Demostrador d'un sistema de calefacció de la llar basat en OpenMote, OpenWSN (IEEE 802.15.4e) i thethings.iO

Roberto Romero Jotel

Màster Universitari en Enginyeria de Telecomunicació (UOC-URL)

Pere Tuset Peiró

Juny de 2015

FITXA DEL TREBALL FINAL

Títol del treball:	*Demostrador d'un sistema de calefacció de la llar basat en OpenMote, OpenWSN (IEEE 802.15.4e) i thethings.iO*
Nom de l'autor:	Roberto Romero Jotel
Nom del consultor:	Pere Tuset Peiró
Data de lliurament:	*06/2015*
Àrea del Treball Final:	*Telemàtica*
Titulació:	*Màster Universitari en Enginyeria de Telecomunicació*

Resum del Treball:

El concepte de *Internet of Things* (IoT) es consolida cada cop més en l'actual era digital. Les grans multinacionals de tecnologia generen nous objectes quotidians intel·ligents amb connexió a Internet sense complir els estàndards de IoT. Un exemple d'aplicació de IoT és obtenir dades dels sensors per automatitzar el control d'accions dels objectes. Les dades dels sensors generen una gran quantitat d'informació i el núvol (*cloud*) és el lloc més indicat per emmagatzemar-les pel seu alt grau de disponibilitat, flexibilitat, seguretat i facilitat d'accés.

S'ha desenvolupat un demostrador del sistema de calefacció de la llar utilitzant la plataforma de hardware OpenMote, la implementació *open-source* de la pila de protocols basada en els estàndards de IoT, OpenWSN, i la plataforma de *cloud* per IoT, thethings.iO. El demostrador consta d'un element central i una xarxa de sensors (WSN) amb OpenWSN (basat en la tecnologia IEEE 802.15.4e) sobre les motes OpenMote.

L'element central està format per una Raspberry Pi amb connexió a Internet i dels

mòduls de hardware OpenBase i la mota OpenMote-CC2538, connectats pel port USB a la Raspberry Pi, per permetre la comunicació amb la WSN. A més, disposa d'una pantalla tàctil connectada a la Raspberry Pi amb una interfície gràfica d'usuari (GUI) basada en el *framework* Qt. S'ha desenvolupat el software necessari per realitzar les funcions de recollir dades de temperatura i humitat periòdicament de la WSN, emmagatzemar-les a la plataforma thethings.iO a través del protocol CoAP (Constrained Application Protocol) i decidir si s'envia ordre d'actuació a una mota per controlar el sistema de calefacció. La comunicació amb la WSN es realitza a través del protocol CoAP. Tanmateix, ha de permetre a l'usuari configurar paràmetres del demostrador, visualitzar l'històric de temperatura i humitat, així com obtenir i mostrar dades de temperatura i humitat de forma instantània de qualsevol mota.

La WSN es basa en el *firmware* de OpenWSN i el hardware OpenBattery i la mota OpenMote-CC2538 funcionant com a sistema autònom. S'ha desenvolupat el programari necessari per poder rebre peticions sobre el protocol CoAP, actuar sobre el sistema de calefacció per mitjà d'un GPIO i adquirir dades de temperatura i humitat relativa del sensor integrat en OpenBattery.

Per últim, s'ha verificat i validat el correcte funcionament del demostrador desenvolupat.

Abstract (in English):

The concept of Internet of Things (IoT) consolidates increasingly in the current digital age. Large technology companies generate new smart quotidian objects with Internet connection without complying with IoT standards. An example of IoT application is to obtain data from sensors to automate the control actions of objects. The sensors data generate a large amount of information and the cloud is the most suitable place to store them for their high availability, flexibility, security and usability.

Has develop a home heating system demonstrator using OpenMote hardware platform, open-source implementation of protocol stack based on IoT standards, OpenWSN, and thethings.iO cloud platform for IoT. The demonstrator consists of a central element and a Wireless Sensor Network (WSN) with OpenWSN (based on IEEE

802.15.4e technology) on OpenMote motes.

The central element is formed by a Raspberry Pi with Internet connection and OpenBase and OpenMote-CC2538 mote hardware modules, connected by USB port to the Rapsberry Pi, to allow communication with WSN. It also has a touch screen connected to Raspberry Pi with a graphical user interface (GUI) based on Qt framework. Has developed the software required to perform the functions of collect temperature and humidity data periodically from WSN, store them on thethings.iO platform through CoAP (Constrained Application Protocol) protocol and decide whether action order is sent to a mote to control heating system. WSN communication is via CoAP protocol. However, should allow the user to configure demonstrator settings, view temperature and humidity history, as well as obtain and display temperature and humidity data instantly from any mote.

WSN is based on firmware OpenWSN and OpenBattery and OpenMote-CC2538 mote hardware operating as autonomous system. Has developed required software to receive requests through CoAP protocol, act on the heating system via a GPIO and acquire temperature and humidity data from OpenBattery integrated sensor.

Finally, has verified and validated the proper functioning of the developed demonstrator.

Paraules clau:

Internet of Things (IoT), OpenMote, OpenWSN, thethings.iO, OpenMote-CC2538, CoAP, Raspberry Pi, IEEE802.15.4e

Agraïments

Als meus familiars i amics pel seu recolzament.

Al consultor, Pere Tuset Peiró, per les seves indicacions per resoldre incidències i els seus ànims.

A en Marc Pous, CEO de thethings.iO, per facilitar un compte PREMIUM i explicar el funcionament de la seva plataforma.

A l'equip del TCM NetLab per prestar-me el mòdul de relé.

Índex de continguts

Índex d'il·lustracions

Índex de taules

1 Introducció

El demostrador per la gestió del sistema de calefacció de la llar utilitza la plataforma de hardware per IoT, OpenMote; la implementació *open-source* de la pila de protocols basada en els estàndards de IoT, OpenWSN; i la plataforma de *cloud computing* per IoT, thethings.iO.

El demostrador consta d'un element central basat en un dispositiu Raspberry Pi amb pantalla tàctil per la interfície gràfica d'usuari (GUI) i una xarxa de sensors (WSN) desplegada a la llar. La WSN es basa en OpenWSN (basat en tecnologia IEEE802.15.4e) sobre les motes OpenMote. Hi ha motes amb capacitat de prendre mesures de temperatura i humitat relativa, i una mota per actuar sobre un relé que controla el sistema de calefacció en funció de les mesures periòdiques de temperatura. Les mesures de temperatura i humitat capturades periòdicament mitjançant la WSN s'emmagatzemen a la plataforma thethings.iO per accedir-hi posteriorment amb l'objectiu de generar gràfics de l'històric de les dades.

1.1 Motivació

Els motius de l'elecció del present Treball Final de Màster (TFM) són diversos i es detallen tot seguit:

- El concepte de *Internet of Things* (IoT) és un dels més rellevants i innovadors en el sector de les telecomunicacions tal i com es va poder veure en el passat *Mobile World Congress* (MWC15). Aquest concepte està en fase de desenvolupament i en els pròxims anys serà capdavanter en el sector de les Tecnologies de la Informació i la Comunicació (TIC). Per tant, considero molt interessant realitzar el TFM en aquesta línea de treball.
- Comprendre el funcionament i saber utilitzar OpenWSN, implementació *open-source* que compleix amb els protocols dels estàndards internacionals de *Internet of Things* (IoT).
- Participar en el desenvolupament de software lliure com ho és el projecte OpenWSN.

- Conèixer un nou sistema encastat, la plataforma de hardware OpenMote amb els seus respectius components.
- Des del meu punt de vista, el TFM és l'única oportunitat durant el Màster per desenvolupar productes innovadors en una vessant molt pràctica (demostrador/aplicació) a partir d'adquirir un nou coneixement teòric. Aquest TFM és un exemple clar de la meva visió.

1.2 Objectius

L'objectiu del present Treball Final de Màster és el desenvolupament d'un demostrador de *Internet of Things* (IoT) per la gestió (monitoratge i control) del sistema de calefacció de la llar emprant la plataforma de hardware OpenMote, el projecte OpenWSN (basat en la tecnologia IEEE 802.15.4e) i la plataforma de *cloud* per IoT, thethings.iO.

Per assolir l'objectiu general detallat anteriorment cal complir amb els següents objectius:

- Conèixer els diversos components de la plataforma de hardware oberta OpenMote.
- Estudiar l'arquitectura i el mode de funcionament del projecte OpenWSN, així com comprendre el funcionament de les diverses aplicacions de software associades (OpenVisualizer i llibreria CoAP).
- Entendre el funcionament de la plataforma de thethings.iO i la forma d'utilitzar les seves llibreries.
- Desenvolupar una plataforma de software utilitzant OpenMote, OpenWSN i Raspberry Pi. Les seves funcions són: recollir les dades de temperatura i humitat de diverses motes de forma periòdica, així com controlar el sistema de calefacció de forma automàtica en funció de les dades.
- Implementar un programa de software en la Raspberry Pi amb la finalitat d'emmagatzemar les dades de temperatura i humitat en la plataforma thethings.iO a través del protocol CoAP.

- Desenvolupar una interfície gràfica d'usuari (GUI) amb el *framework* Qt per la Raspberry Pi. Les sever funcions són: configuració de paràmetres del demostrador, visualització de l'històric de temperatura i humitat, així com obtenir i mostrar dades de temperatura i humitat de forma instantània de qualsevol mota.
- Verificar i validar el correcte funcionament del demostrador desenvolupat en els casos d'ús.

1.3 Enfocament i mètode seguit

La metodologia per realitzar el Treball Final de Màster proporciona les mínimes garanties per assolir el desenvolupament del mateix en el termini establert i amb la màxima qualitat.

El desenvolupament s'ha realitzat utilitzant les metodologies d'un cicle de vida clàssic o en cascada i àgil. La metodologia en cascada s'ha emprat en les primeres etapes del projecte en una visió més global perquè es disposa d'uns requisits que no varien en la durada del projecte. Mentre que la metodologia àgil s'ha utilitzat després dels desenvolupaments inicials en el procés d'aprenantge del funcionament de les diverses aplicacions de software utilitzades.

Les etapes de la metodologia en cascada emprades han estat anàlisi de requisits, disseny global i desenvolupament inicial (primers usos de la llibreria Python CoAP de OpenWSN).

Els conceptes utilitzats en la metodologia àgil són:

- **Desenvolupament iteratiu i incremental:** desenvolupament basat en petites millores, una darrere altres.
- **Codi simple:** implementació del codi de forma simple i anar afegint noves funcionalitats un cop verificades les simples.
- **Proves continues:** proves per verificar el correcte funcionament després de cada petit desenvolupament i la usabilitat de la interfície gràfica d'usuari (GUI).

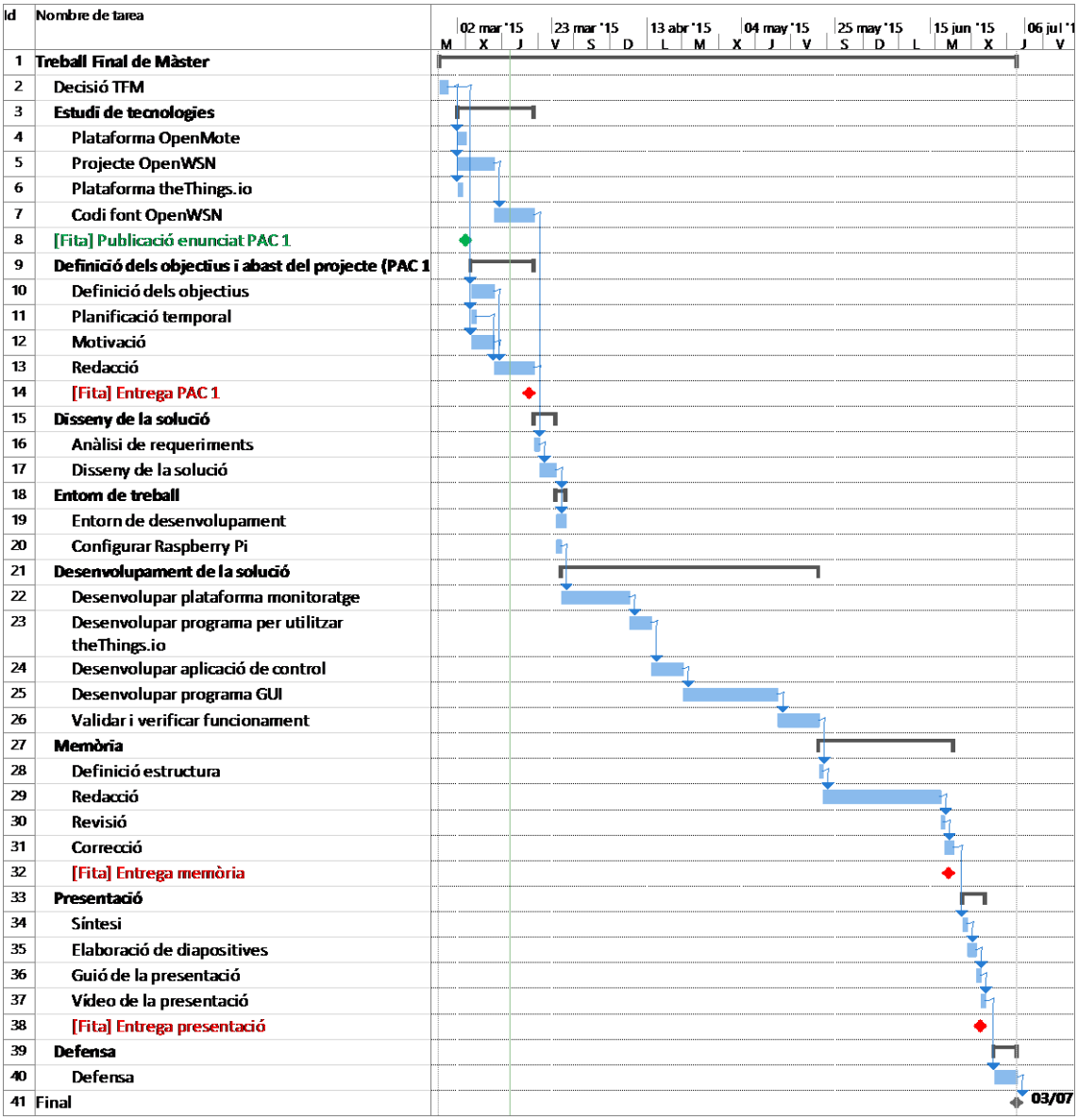
1.4 Planificació del treball

En la següent taula es pot observar la planificació temporal del present Treball Final de Màster dividida en les principals tasques a realitzar.

Nom tasca	Duració	Inici	Final
Treball Final de Màster	**92 dies**	**26/02/2015**	**03/07/2015**
Decisió TFM	**2 dies**	**26/02/2015**	**27/02/2015**
Estudi de tecnologies	**13 dies**	**02/03/2015**	**18/03/2015**
Plataforma OpenMote	2 dies	02/03/2015	03/03/2015
Projecte OpenWSN	6 dies	02/03/2015	09/03/2015
Plataforma thethings.iO	1 dia	02/03/2015	02/03/2015
Codi font OpenWSN	7 dies	10/03/2015	18/03/2015
[Fita] Publicació enunciat PAC 1	0 dies	04/03/2015	04/03/2015
Definició dels objectius i abast del projecte (PAC 1)	**10 dies**	**05/03/2015**	**18/03/2015**
Definició dels objectius	3 dies	05/03/2015	09/03/2015
Planificació temporal	1 dia	05/03/2015	05/03/2015
Motivació	3 dies	05/03/2015	09/03/2015
Redacció	7 dies	10/03/2015	18/03/2015
[Fita] Entrega PAC 1	0 dies	18/03/2015	18/03/2015
Disseny de la solució	**3 dies**	**19/03/2015**	**23/03/2015**
Anàlisi de requeriments	1 dia	19/03/2015	19/03/2015
Disseny de la solució	2 dies	20/03/2015	23/03/2015
Entorn de treball	**2 dies**	**24/03/2015**	**25/03/2015**
Entorn de desenvolupament	2 dies	24/03/2015	25/03/2015
Configurar Raspberry Pi	1 dia	24/03/2015	24/03/2015
Desenvolupament de la solució	**41 dies**	**25/03/2015**	**20/05/2015**
Desenvolupar plataforma monitoratge	11 dies	25/03/2015	08/04/2015
Desenvolupar programa per utilitzar thethings.iO	3 dies	09/04/2015	13/04/2015
Desenvolupar aplicació de control	5 dies	14/04/2015	20/04/2015
Desenvolupar programa GUI	15 dies	21/04/2015	11/05/2015
Validar i verificar funcionament	7 dies	12/05/2015	20/05/2015
Memòria	**22 dies**	**21/05/2015**	**19/06/2015**
Definició estructura	1 dia	21/05/2015	21/05/2015
Redacció	18 dies	22/05/2015	16/06/2015
Revisió	1 dia	17/06/2015	17/06/2015
Correcció	2 dies	18/06/2015	19/06/2015
[Fita] Entrega memòria	0 dies	19/06/2015	19/06/2015
Presentació	**5 dies**	**22/06/2015**	**26/06/2015**
Síntesi	1 dia	22/06/2015	22/06/2015
Elaboració de diapositives	2 dies	23/06/2015	24/06/2015
Guió de la presentació	1 dia	25/06/2015	25/06/2015
Vídeo de la presentació	1 dia	26/06/2015	26/06/2015
[Fita] Entrega presentació	0 dies	26/06/2015	26/06/2015
Defensa	**5 dies**	**29/06/2015**	**03/07/2015**
Defensa	5 dies	29/06/2015	03/07/2015
Final	**0 dies**	**03/07/2015**	**03/07/2015**

Taula 1: *Planificació temporal del TFM*

El diagrama de Gantt de la planificació temporal del TFM és:



Il·lustració 1: *Diagrama de Gantt de la planificació temporal del TFM*

1.5 Sumari de productes obtinguts

Els productes obtinguts en la realització del present Treball Final de Màster són:

- Projecte de software openDemo que inclou la llibreria Python CoAP de OpenWSN i les aplicacions *openDemo* i *openDemoGUI*.
- Projecte de software thethingsio-coap-openDemo format per la llibreria Node.js CoAP de thethings.iO i les aplicacions Write.js i Read.js.

- Projecte de software openwsn-fw-openDemo que consta del projecte OpenWSN amb les diverses aplicacions CoAP creades per realitzar les funcions del demostrador.

1.6 Descripció dels altres capítols de la memòria

L'estructura del present document és:

- **Capítol 2. Internet of Things (IoT):** es realitza un descripció general del concepte IoT, s'especifiquen en detall els estàndards IoT per capa i s'explica de forma breu els sistemes operatius per IoT.
- **Capítol 3. OpenWSN:** s'explica de forma global el projecte OpenWSN i la seva pila de protocols, així com les eines associades OpenVisualizer i la llibreria Python CoAP.
- **Capítol 4. Plataformes de hardware i software:** es detallen les característiques tècniques de les plataformes de hardware OpenMote i Raspberry Pi i s'explica la plataforma de software thethings.iO, així com la forma d'utilitzar la seva llibreria Node.js CoAP.
- **Capítol 5. Configuracions prèvies:** s'especifica com instal·lar i configurar les diverses eines de software necessàries pel desenvolupament del present projecte tant en l'entorn de desenvolupament com en el de producció (Raspberry Pi). A més, es detallen els passos a seguir per sincronitzar una xarxa OpenWSN sobre OpenMote.
- **Capítol 6. Desenvolupament:** es detalla la solució final a partir d'una visió global amb la justificació del disseny realitzat per acabar en una explicació detallada del desenvolupament per a cadascuna de les funciones del demostrador. Per últim, s'especifica el procediment de desplegament de la solució a l'entorn de producció.
- **Capítol 7. Verificació:** s'explica la verificació del correcte funcionament del desenvolupament realitzat.
- **Capítol 8. Conclusions:** es presenten les conclusions del present TFM i les possibles línies futures d'actuació que es deriven del projecte.

2 Internet of Things (IoT)

2.1 Introducció

El primer cop que es va anunciar el concepte de IoT va ser en 1.999 pel visionari britànic Kevin Ashton en el context de la gestió de cadena de subministrament [1].

Actualment el paradigma de IoT es pot definir com una xarxa d'objectes o "coses" intel·ligents que permet intercanviar informació, adquirir dades de sensors i actuar sobre altres objectes sense la interacció humana, és a dir, M2M (*machine-to-machine*) [1][2].

Els objectes són sistemes encastats amb capacitat de processar, adquirir dades de sensors, actuar sobre "coses" i comunicar sense fils a certa distància. Aquestes qualitats permeten la consideració de la xarxa d'objectes com una xarxa de sensors sense fils (*Wireless Sensor Network* o WSN). En WSN als objectes se'ls anomena *motes*. Les *motes* han de tenir els següents requisits:

- **Baix consum:** les *motes* funcionen com a sistemes autònoms amb bateria, en conseqüència, el consum ha de ser el més baix possible per allargar la vida útil de la bateria. Els elements que impacten més en el consum de les *motes* són el microcontrolador i el transceptor de ràdio. Per tant, és necessari que el microcontrolador tingui un baix consum en mode actiu (desenes de mA) i l'opció del mode *sleep* amb consum de l'ordre de µA. Així mateix, el transceptor de ràdio ha de tenir baix consum en els modes transmissió i recepció, així com disposar de l'opció del mode *sleep*.
- **Baixa velocitat de transmissió:** habitualment les aplicacions de WSN transmeten poca quantitat d'informació, i per tant, no són necessàries taxes de velocitat elevades.
- **Baixa capacitat de processat:** en general, el cost computacional de les aplicacions de WSN és baix, i en conseqüència, no és necessari processadors molt potents. La potència de processament impacte negativament en el consum, és a dir, processadors de baixa potència consumeixen menys.

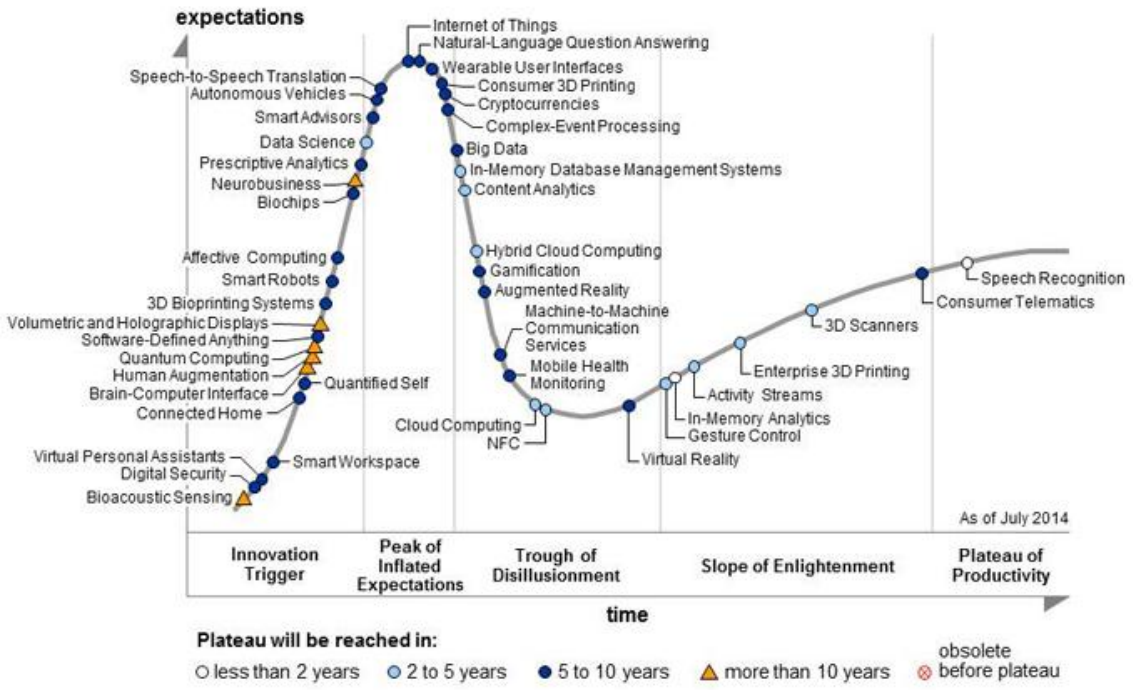
- **Capacitat d'adquirir dades i actuar:** disposar d'entrades/sortides digitals i analògiques, així com busos de comunicació (I2C, RS-485, RS-232) per permetre la comunicació amb sensors (monitoratge), actuadors (control) i altres objectes (M2M).
- **Mida reduïda:** la mida reduïda facilita la integració de les motes en el disseny d'objectes quotidians.
- **Baix cost de producció:** un preu reduït de les motes permet que el cost de l'objecte intel·ligent final no sigui massa elevat, i per tant, facilita la seva integració en la societat.

Com s'ha comentat anteriorment, els objectes poden adquirir dades de qualsevol sensor que ens envolta, això provoca la generació d'una gran quantitat de dades i el núvol (*cloud*) és la millorar infraestructura pel seu emmagatzematge, processat i visualització [1].

Els objectes intel·ligents tenen un identificador únic operant en la infraestructura d'Internet, és a dir, amb una adreça IP assignada. Segons la consultora Gartner [3] el 2020 hi haurà 25 bilions anglosaxons de "coses" connectades, fet que el desenvolupament de IoT va lligat a IPv6.

A més de la predicció d'objectes connectats, Gartner considera el concepte de IoT capdavanter en el món de les tecnologies de la informació i la comunicació (TIC), com es pot observar en la il·lustració següent *Hype Cycle for Emerging Technologies* [4]. En aquest gràfic es pot veure com IoT es troba en el punt més elevat del *Peak of Inflated Expectations*.

En l'actualitat es parla molt del paradigma IoT en el sector de les TIC, com per exemple en el passat *Mobile World Congress* (MWC15), per la gran quantitat d'entorns d'aplicació on es pot desplegar. En els pròxims anys es veurà si aquesta gran expectació es complirà o es desinflarà.



Il·lustració 2: *Gràfic Hype Cycle for Emerging Technologies, Gartner [4]*

Alguns possibles escenaris d'aplicació de IoT són:

- **Agricultura:** entorns d'agricultura intel·ligent. Alguns exemples d'aplicacions poden ser el monitoratge de l'estat del sòl (sec o humit), el control del rec, el monitoratge de temperatura i humitat en un hivernacle, etc.
- **Automoció:** les aplicacions d'aquest escenari poden ser l'automòbil connectat per interactuar amb elements de l'entorn (carreteres, semàfors, etc.) i vehicles d'activitat logística amb l'objectiu de reduir costos.
- **Control industrial:** les aplicacions d'aquest escenari poden ser:
    - Aplicacions de control de la producció.
    - Monitoratge de la temperatura i la qualitat de l'aire.
    - Localització en interiors.
- **Domòtica:** aquest escenari presenta aplicacions d'automatització i monitoratge de la llar com el control remot d'aplicacions, sistema de detecció d'intrusos, automatització de persianes i tendals, etc.
- **Logística:** les aplicacions de l'escenari poden ser monitoratge de la qualitat de les condicions de l'enviament, localització d'articles, seguiment de flotes, etc

- **Salut:** aquest escenari inclou aplicacions com el monitoratge de les constants vitals dels pacients en hospitals, l'assistència a persones grans o discapacitades que viuen soles, monitoratge de la radiació ultraviolada,etc.
- **Seguretat:** aplicacions per la detecció de fuites de gasos i radiació, per controlar l'accés a àrees restringides, etc.
- ***Smart City*:** aplicacions per ciutats intel·ligents, algunes aplicacions són:
    - Aparcament intel·ligent indicant places disponibles en la ciutat.
    - Monitoratge de l'estructura d'edificis, ponts i monuments històrics.
    - Mapes de soroll en temps real.
    - Il·luminació intel·ligent mitjançant la presència de ciutadans.
    - Recollida intel·ligent de residus, detecció dels nivells de brossa en el contenidors per optimitzar rutes de recollida.
- ***Smart Metering:*** les aplicacions poden ser control i gestió del consum d'energia elèctrica, lectura remota de comptadors d'aigua i gas, etc.

2.2 Estàndards IoT

Els protocols estàndards de IoT presenten tres requeriments [5]:

- **Pila de comunicació de baix consum:** habitualment els objectes intel·ligents utilitzen bateries com a font d'alimentació. Per tant, la pila de comunicació ha de ser de baix consum. Per exemple, protocols d'Internet com HTTP i TCP no estan optimitzats per comunicacions de baix consum.
- **Pila de comunicació altament fiable:** Internet proporciona mecanismes de fiabilitat en la comunicació d'extrem a extrem. Per combinar perfectament IoT amb Internet és necessari que IoT ofereixi la mateixa fiabilitat en la comunicació que proporciona Internet.
- **Pila de comunicació amb Internet:** Els objectes intel·ligents han de comunicar-se de la mateixa forma que qualsevol altre dispositiu connectat a Internet, és a dir, mitjançant el protocol IP.

Els estàndards de *Internet of Things* utilitzats en el present Treball Final de Màster són [5] [6]:

- IEEE802.15.4-2006
- IEEE802.15.4e
- *IPv6 over Low power Wireless Personal Area Networks* (6LoWPAN)
- *IPv6 Routing Protocol for Low power and Lossy Networks* (RPL)
- *Constrained Application Procotol* (CoAP)

Tot seguit s'explica les característiques principals de cadascun d'ells.

2.2.1 IEEE802.15.4-2006

IEEE802.15.4-2006 és la segona versió del estàndard IEEE802.15.4. Defineix la capa física i la capa de control d'accés al medi (*Medium Access Control,* MAC). Està dissenyat per *Low Power* i *Low Rate Wireless Personal Area Networks* (WPANs) i s'ha convertit en un estàndard de *facto* per WSN [7].

La base de IEEE802.15.4 és permetre baix consum, comunicació ubiqua de baixa velocitat entre dispositius propers amb poca o sense infraestructura. El seu mode de funcionament bàsic té un abast de 10 metres amb taxes de 250 kbps.

2.2.1.1 Capa física

Les característiques de la capa física són [7]:

- Baix consum d'operació
- Baixa velocitat de transmissió 250 a kbps
- Suport multicanal amb temps de resposta ràpid 192 µs
- Funcions de gestió del consum com la qualitat de l'enllaç i la detecció d'energia.

Es defineixen quatre capes físiques [7]:

- 868/915 MHz: tècnica DSSS amb modulació BPSK.
- 868/915 MHz: tècnica DSSS amb modulació O-QPSK.
- 2450 MHz: tècnica DSSS amb modulació O-QPSK.
- 868/915 MHz: tècnica PSSS amb modulació ASK.

La banda de 868 MHz (2 canals) s'utilitza a Europa i la banda de 915 MHz (30 canals) a Nord Amèrica i Austràlia. La banda ISM de 2.4 GHz es defineixen 16 canals de 2 MHz d'amplada separats 5 MHz per minimitzar interferències entre canals adjacents.

L'estàndard requereix que els transceptors de ràdio tinguin una sensibilitat mínima de -85 dBm o millor, en la pràctica, els transceptors tenen sensibilitats entre els valors de -95 dBm i -100 dBm.

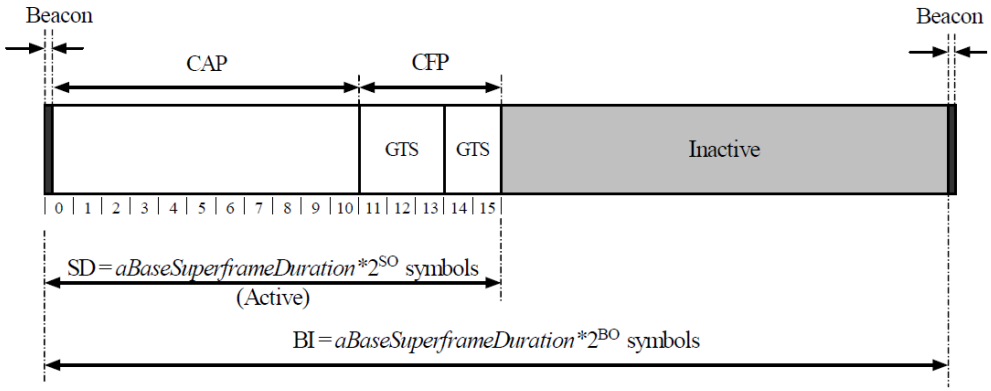
2.2.1.2 Capa MAC

Les característiques de la capa MAC són [7]:

- Temps real adequat per la reserva garantides d'intervals de temps.
- Prevenció de col·lisions a través de CSMA/CA.
- Suport integrat per a comunicacions segures (xifrat AES de 128 bits).
- Construcció de topologies en estrella i en malla
- Suport a baixes latències i dispositiu d'adreçament dinàmic.

Els protocols MAC defineixen dos mètodes d'accés al canal [7]:

- **Sense *Beacon*:** s'utilitza l'estàndard CSMA per resoldre el conflictes d'accés al medi. Els paquets rebuts amb èxit són reconeguts positivament.
- **Amb *Beacon*:** es segueix una estructura de supertrama, on un coordinador de la xarxa transmet balises a intervals predeterminats. Després de la balisa ve un període de contenció d'accés (*Contention Access Period* o CAP) on els veïns del coordinador poden parlar amb ell a través d'un esquema d'accés al medi Aloha ranurat [7]. Durant aquest període, els nodes poden sol·licitar un espai de temps dedicat al període *Guaranteed Time Slot* (GTS). Posteriorment, s'inicia el *Contention Free Period* (CFP) on els nodes poden realitzar transmissions amb latència limitada en el seu espai de temps dedicat assignat anteriorment. Un cop finalitzat el CFP comença el període inactiu on cap node pot transmetre. Tots els nodes es troben en mode *sleep* per estalviar energia. Passat un cert període de temps es torna a transmetre la balisa i el procés es explicat es torna a repetir. En la il·lustració de sota es pot observar l'estructura de supertrama detallada anteriorment.



Il·lustració 3: *Estructura supertrama [7]*

IEEE802.15.4-2006 defineix tres tipus de rols dels dispositius:

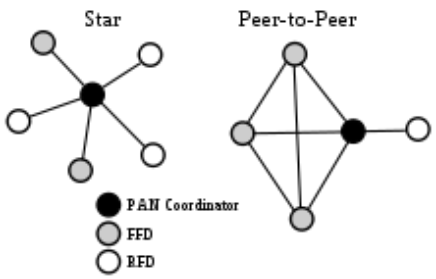
- **Coordinador PAN (*Personal Area Network*):** coordinador que és el controlador principal de la PAN. Una xarxa només té un coordinador PAN.
- **Coordinador:** node final amb capacitat de proporcionar coordinació i altres serveis a la xarxa.
- **Node final:** dispositiu amb capacitat d'accedir al medi sense fils a través de les definicions de l'estàndard IEEE802.15.4, a nivell físic i MAC.

Així mateix, especifica dos tipus de nodes [7]: *Full Function Device* (FFD) i *Reduced Function Device* (RFD).

FFD són nodes amb capacitat de participar en qualsevol topologia de xarxa i poden realitzar les funcions de tots els rols definits per l'estàndard (coordinador PAN, coordinador i node final). A més, poden comunicar-se amb qualsevol altre node de la xarxa.

RFD són nodes simples amb recursos de hardware i software limitats. Només es poden comunicar amb FFD, per tant, mai poden ser coordinadors, són nodes finals. Només poden participar en una xarxa amb topologia estrella.

Es defineixen dos tipus de topologia: en estrella i punt a punt. En la il·lustració següent es pot veure un exemple de les topologies estrella i punt a punt.



Il·lustració 4: *Topologies estrella i punt a punt [8]*

La MAC permet la formació de topologies en estrella extensa *multi-hop*.

2.2.2 IEEE802.15.4e

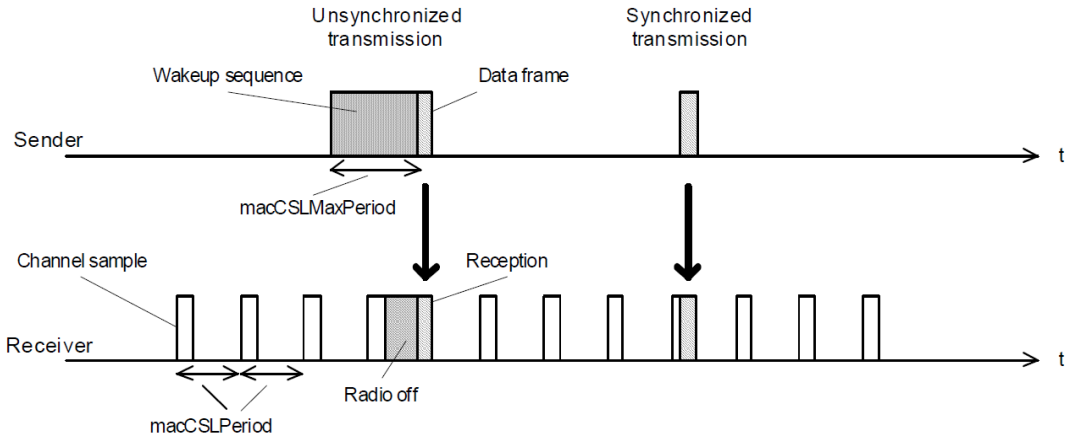
Desenvolupat pel grup de treball *IEEE 802.15 WPAN Task Group 4* en 2012. Defineix una esmena de la capa MAC de l'estàndard IEEE802.15.4-2006 amb l'objectiu de donar suport als mercats industrials.

Incorpora nous modes d'operació [9] per donar suport a diferents tipus d'aplicacions de la xarxa:

- ***Deterministic & Synchronous Multi-channel Extension* (DSME):** per aplicacions d'alta disponibilitat, eficiència, escalabilitat i robustesa.
- ***Low Latency Deterministic Network (LLDN):*** per aplicacions de baixa latència com robots, grues, etc.
- ***Time Slotted Channel Hopping (TSCH):*** per aplicacions destinades a entorns industrials on el consum d'energia ha de ser reduït i la comunicació ha de ser robusta enfront interferències.
- ***Radio Frequency Identification Blink (RFID Blink):*** per aplicacions d'identificació d'objectes o persones, localització o seguiment.
- ***Asynchronous Multi-Channel Adaption (AMCA):*** per a xarxes de grans dimensions i dispersió geogràfica com xarxes de monitoratge d'infraestructures i de control de processos.

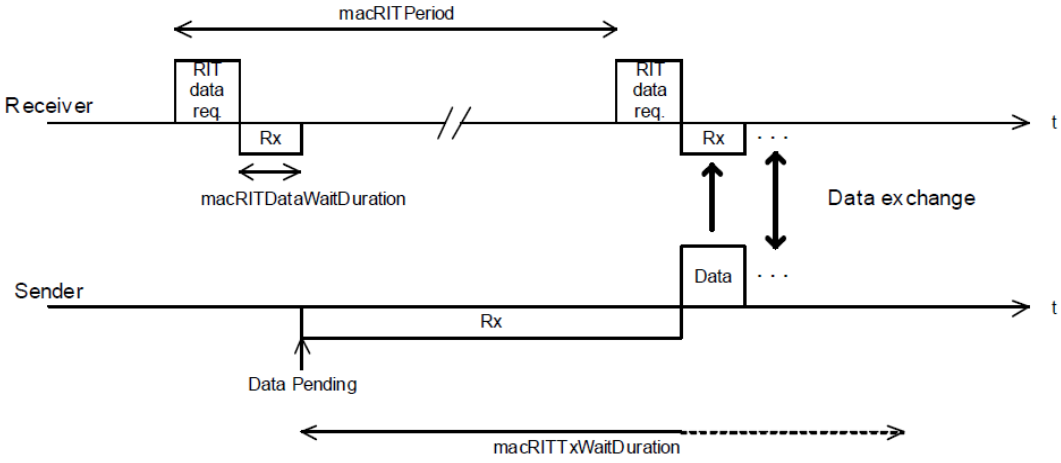
Defineix el protocol *Low Energy* (LE) que permet obtenir cicles de treball al voltant del 1 %. LE incorpora dos mecanismes per reduir el consum d'energia [9]:

- ***Coordinated Sampled Listening (CSL):*** el receptor escolta el canal periòdicament verificant si hi ha sol·licituds de transmissió. Si durant el període d'escolta rep una trama *wakeup* amb la seva adreça MAC, el receptor passa a mode *sleep* durant un temps específic (*redezvous* (RZ) Time) inclòs en la trama *wakeup*. RZ Time és el temps entre el final de la trama *wakeup* i l'inici de la trama de dades a enviar al receptor. En la il·lustració es pot observar el funcionament del mecanisme CSL.



Il·lustració 5: *Funcionament del mecanisme CSL [9]*

- ***Receiver Initiated Transmissions (RIT):*** s'utilitza en xarxes PAN que no empren balises. Es basa en què el receptor envia trames *datareq* de forma periòdica a través de CSMA/CA no ranurat, després de cada enviament escolta el canal durant període curt de temps. El transmissor escolta el canal a l'espera de rebre una trama *datareq* per transmetre immediatament una trama de dades. En la següent il·lustració es pot veure el funcionament del mecanisme RIT.



Il·lustració 6: *Funcionament del mecanisme RIT [9]*

L'estàndard IEEE802.15.4e defineix els *Information Elements* (IEs) com un mètode flexible, extensible i senzill d'implementar per l'encapsulament d'informació [10].

Un IE està format per un camp identificador (ID), un camp de longitud i un camp de contingut. Els IEs proporcionen contenidors d'informació de sincronització, estat de la xarxa, etc. i permet afegir noves definicions de IEs en futures versions del estàndard.

El contingut d'un IE és un camp variable en funció del mode de funcionament, funcionalitats i modes de comportament (CSL, RIT, balises, etc.).

La trama MAC de l'estàndard IEEE802.15.4e està formada per tres parts principals [10]:

- ***MAC Header (MHR):*** està formada pel camps:
  - *Frame Control:* tipus de trama.
  - *Sequence Number:* número únic a la trama.
  - *Adressings Fields:* identificadors i adreces origen i destí de la PAN.
  - *Auxiliary Security Header:* opcional, extensió del mecanismes de seguretat de la informació de nivells superiors.
  - *Capçaleres Information Elements (IEs):* dades addicionals orientats al funcionament i comportament específic de determinats elements.
- ***MAC Payload o MAC Service Data Unit (MSDU):*** *payload* dels IEs i de la trama (pot correspondre amb les PDUs de nivells superiors) **.**
- ***MAC Footer (MFR)***: dos bytes per l'emmagatzematge de la seqüència de control de la trama (*Frame Control Sequence* o FCS) per verificar i validar la integritat de les dades.

En la il·lustració de sota es mostra el format d'una trama MAC IEEE802.15.4e.

<table>
<tr><td>Octets:
1/2</td><td>0/1</td><td>0/2</td><td>0/1/2/8</td><td>0/2</td><td>0/1/2/8</td><td>0/1/5/6/1
0/14</td><td colspan="2">variable</td><td>variable</td><td>2</td></tr>
<tr><td rowspan="2">Frame Control</td><td rowspan="2">Sequence Number</td><td>Destination PAN Identifier</td><td>Destination Address</td><td>Source PAN Identifier</td><td>Source Address</td><td rowspan="2">Auxiliary Security Header</td><td colspan="2">Information Elements</td><td rowspan="2">Frame Payload</td><td rowspan="2">FCS</td></tr>
<tr><td colspan="4">Addressing fields</td><td>Header IEs</td><td>Payload IEs</td></tr>
<tr><td colspan="7">MHR</td><td colspan="2">MAC Payload</td><td>MFR</td></tr>
</table>

Il·lustració 7: *Format de la trama MAC IEEE802.15.4e [10]*

L'estàndard IEEE802.15.4e introdueix dues noves versions de les balises que s'utilitzen en els modes de funcionament DSME i TSCH [9] i el seu contingut està format a partir de IEs. Aquestes balises són [10]:

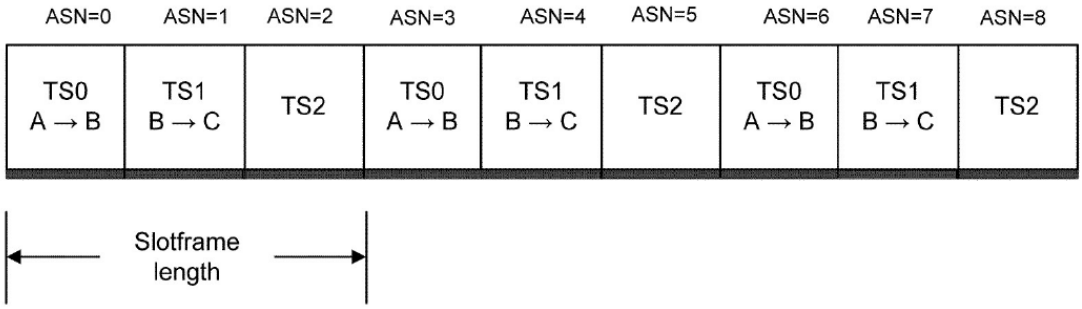
- ***Enhanced Beacon (EB):*** proporcionen més flexibilitat en contingut que les balises de l'estàndard IEEE802.15.4-2006. Els EBs tenen el valor "0b10" en camp versió de la trama.
- ***Enhanced Beacon Requests (EBR):*** s'utilitzen per sol·licitar un EB especificant filtres en la resposta perquè el EB respongui només la informació demanda. D'aquesta forma es redueix la mida de la trama enviada.

2.2.2.1 TSCH

TSCH és l'element clau que permet a l'estàndard IEEE802.15.4e donar suport al mercat industrial [11] perquè presenta més robustesa enfront a interferències externes i a *multi-path fading*. És una tècnica d'accés al medi que utilitza comunicacions sincronitzades en el temps per realitzar operacions de baix consum i salts de canal de freqüència per proporcionar alta fiabilitat [12].

2.2.2.1.1 Slotframes *i* slots

TSCH divideix el temps en ranures d'ús predefinit per evitar col·lisions en la comunicació. Es basa en una estructura de *slotframe*. El *slotframe* és un conjunt de *timeslots* que es repeteixen en el temps [13]. Cada *timeslot* té la duració suficient com perquè el transmissor enviï una trama de mida màxima i el receptor contesti una trama ACK. El nombre de *timeslots* determinen la periodicitat de repetició dels mateixos. El nombre total de *timeslots* que ha transcorregut des de l'inici de la xarxa s'anomena *Absolute Slot Number* (ASN), per tant, si el *timeslot* es repeteix també s'incrementa el seu valor. En la següent il·lustració es pot veure un exemple de *slotframe* amb 3 *timeslots* (TS0,TS1 i TS2). En l'exemple es mostra com es repeteixen els *timeslots* i com es va incrementant el valor de ASN.



Il·lustració 8: *Exemple slotframe amb 3 timeslots [9]*

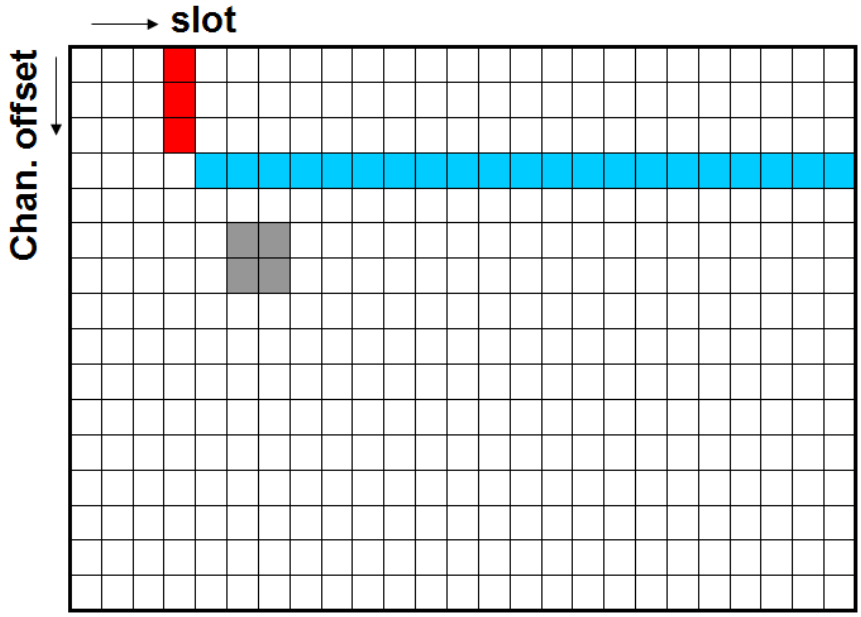
Els nodes s'han de sincronitzar amb l'estructura de *slotframe* de la xarxa, si un node no es sincronitza amb la xarxa no podrà comunicar-se. Cada node disposa d'un planificador que decideix quina acció ha de realitzar en cada *timeslot*. Les possibles accions són: transmetre i rebre dades i dormir.

Quan un node ha de transmetre un paquet (generat en una capa superior), el passa a la seva capa MAC on s'afegeix a una cua de transmissió. En cada *timeslot* la capa MAC revisa si hi ha cap paquet en la cua per aquell *timeslot*, si és així es transmet el paquet. En cas contrari, el node passa a mode *sleep*.

En el cas de rebre dades, el node encén la ràdio abans de l'instant esperat d'arribada del paquet. Verifica si és el destinatari del paquet, en cas afirmatiu envia un ACK al transmissor, apaga la ràdio i passa el paquet rebut a la capa superior perquè sigui processat. En cas contrari, el node passa a mode *sleep*.

En un *timeslot* on el node està dormit no s'encén la ràdio en cap moment, reduint així el consum d'energia.

El *slotframe* està dividit en temps i en canals de freqüència, per tant, es poden realitzar les transmissions de tres formes diferents. En la següent il·lustració es mostra una matriu canals x *slot* amb aquestes tres opcions.



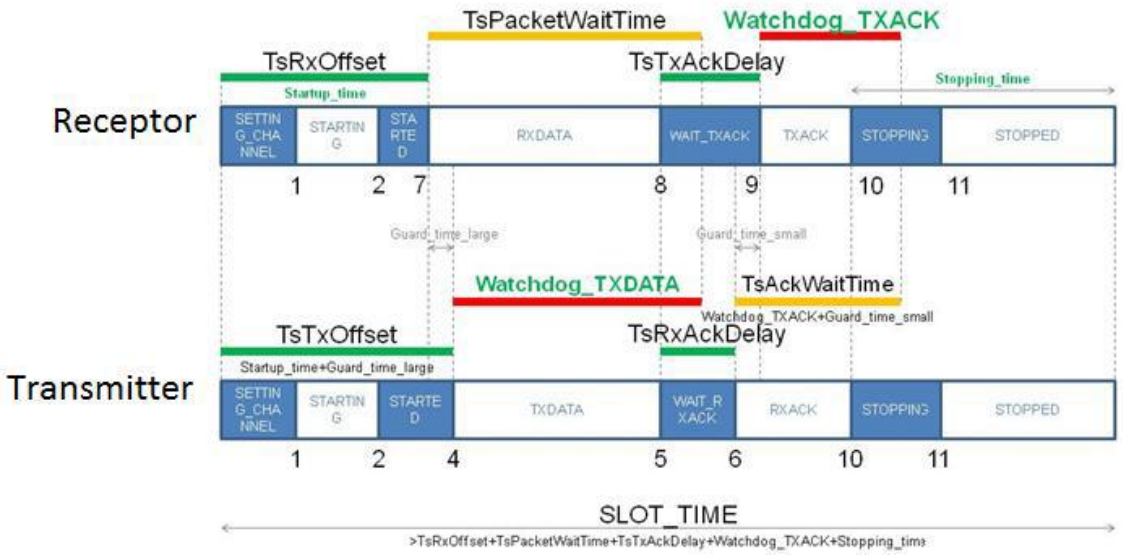
Il·lustració 9: *Estructura slotframe [9]*

Una opció és que les transmissions es realitzin en un mateix *slot* però per diferents canals (caselles vermelles). Un altre és realitzar la transmissió per un mateix canals en diferents *slots* (caselles blaves). Per últim, la transmissió es realitza per diferents canals i diferents ranures (caselles grises).

Durant un *timeslot*, un node transmissor envia una trama a un node receptor, si aquest últim rep correctament la trama contesta al transmissor amb una trama ACK. Si el node transmissor no rep la trama ACK passat un període de temps d'espera, tornarà a transmetre la trama esperant al següent *timeslot* de transmissió assignat per aquell node receptor.

En la següent il·lustració es pot observar l'organització del *timeslot*.

El *timeslot* comença en T=0 des de la perspectiva del dispositiu transmissor. El transmissor espera TsTxOffset µs, després inicia la transmissió del paquet. El transmissor espera TsRxAckDelay µs, posteriorment entra en mode recepció a l'espera de rebre la trama ACK del receptor. Si la trama ACK no arriba passats TSAckWaitTime µs, el node apaga la ràdio i la trama ACK no arribarà. Des de l'inici de la ranura, el receptor espera TsRxOffset µs, llavors encén el transceptor de ràdio. Es manté així fins passats TsPacketWaitTime µs o fins rebre un paquet. Després de rebre el paquet espera durant TsTxAckDelay µs i contesta al transmissor amb una trama ACK.



Il·lustració 10: *Organització timeslot [11]*

2.2.2.1.2 Channel Hopping

El càlcul del canal per el qual s'ha de transmetre o rebre dades es realitza a través de la següent expressió [5]:

canal =(ASN + channelOffset) mod N_{ch}; on N_{ch} és el nombre de canals.

En la banda de 2.4 GHz en què s'utilitzen 16 canals, el rang de valors del canal està comprès entre 0 i 15. El càlcul del canal es realitza amb l'expressió:

canal =(ASN + channelOffset) mod 16

2.2.2.1.3 Sincronització

La sincronització entre nodes és necessària per mantenir la connexió amb els veïns en una xarxa basada en *slotframe*. Hi ha dos mètodes [13] per a sincronitzar un node en la xarxa:

- **Sincronització basada en ACK:** el node receptor calcula la delta entre el temps esperat d'arribada de la trama i el temps actual d'arribada de la trama. Aquest valor s'envia al node transmissor en les trames ACK per mantenir la sincronització amb el node receptor.
- **Sincronització basada en trama:** el node receptor calcula la delta entre el temps esperat d'arribada de la trama i el temps actual d'arribada de la trama. Aquest

valor l'utilitza el receptor per ajustar el seu rellotge, així el receptor es sincronitza amb el rellotge del node transmissor.

El procés permet que els nodes es sincronitzin en poques desenes de µs, valor molt petit en comparació amb el temps de guarda del protocol (1 ms). Els rellotges dels nodes són oscil·ladors de quarts amb derives d'aproximadament 10 ppm, és a dir, passat un segon pot haver diferències de 10 µs. Això fa que els nodes han de resincronitzar-se cada vegada. Si hi ha trànsit en la xarxa els nodes es van resincronitzant. En el cas que no hi hagi trànsit passats 30 segons, el nodes intercanvien paquets sense informació (*keep-alive messges*) per tornar a sincronitzar.

Un node només es sincronitza amb el temps del seu pare, formant així una topologia en arbre de sincronització. Això permet que tots els nodes de la xarxa tenen en comú el sentit del temps.

2.2.2.1.4 Formació de la xarxa

La xarxa IEEE802.15.4e TSCH està formada per la combinació de dispositius FFD i RFD.

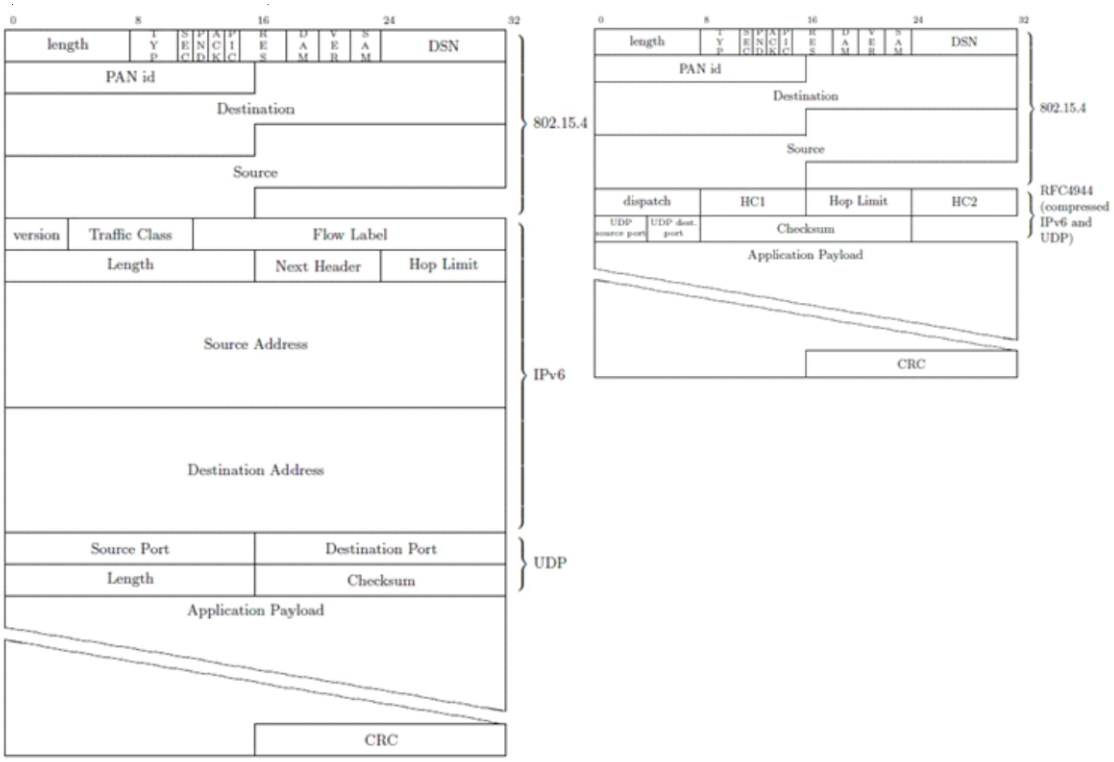
El procediment de formació de la xarxa té dos components [11]: *advertising* i *joining*. En el cas de *advertising*, només els nodes FFD que ja són membres de la xarxa poden enviar trames d'instrucció per anunciar de la presència de la xarxa. Els nous nodes que intenten afegir-se a la xarxa escolten aquestes trames i s'uneixen a la xarxa enviant al node anunciador una trama *Join request*. Després, el node anunciador activa als nous nodes enviant una trama d'activació.

Una nova xarxa s'inicia quan el coordinador PAN comença a anunciar, en aquest instant, es converteix en el primer node de la xarxa. El coordinador PAN inicia almenys un *slotframe* perquè altres nodes de la xarxa es puguin sincronitzar.

2.2.3 IETF 6LoWPAN

6LoWPAN és una capa d'adaptació desenvolupada per el 6LoWPAN *Working Group* de la IETF [14]. Aquesta capa proporciona mecanismes d'encapsulament, compressió de capçaleres i fragmentació per permetre enviar i rebre paquets IPv6 sobre xarxes basades en l'estàndard IEEE802.15.4.

Aquest mecanisme són imprescindibles per poder enviar i rebre paquets IPv6 en xarxes IEEE802.15.4. Si s'agafa com exemple el paquet de la il·lustració següent, es pot observar que hi ha *overhead* en les capçaleres IPv6 amb longitud de 40 bytes i UDP amb longitud de 8 bytes. Aquest *overhead* en les capçaleres impacte negativament sobre l'amplada de banda en la capa física [14]. Per tant, es realitza una compressió de les capçaleres basada en eliminar camps de les capçaleres IPv6 i UDP amb valors coneguts, o que es poden deduir dels camps de la capçalera de IEEE802.15.4 [15].

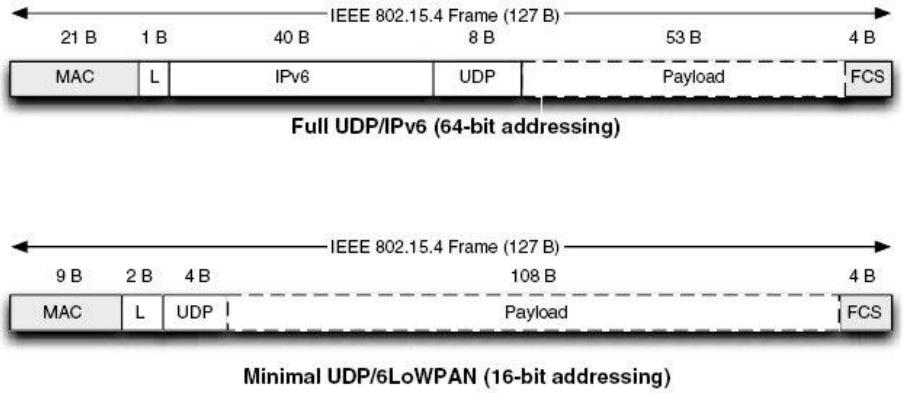


Il·lustració 11: *Paquet amb totes les capçaleres i paquet comprimit amb 6LoWPAN [15]*

Així en la capçalera de IPv6, el camp versió és sempre 6, la classe de trànsit i l'etiqueta de flux no s'usen mai i el camp de longitud és sempre igual a la longitud del camp IEEE802.15.4 menys la longitud de la capçalera IPv6. Per tant, tots aquests camps es poden eliminar. Així mateix, el *Next Header* apunta a UDP o TCP, aquest camp de 8 bits pot ser substituït per un camp de 2 bits com a part del camp HC1 de la capçalera 6LoWPAN. Per últim, el RFC2464 defineix que es poden obtenir les adreces IPv6 de 128 bits a partir de les adreces MAC de 64 bits a través dels camps origen i destí de IEEE802.15.4. En conseqüència, els camps de les adreces d'origen i destí de IPv6 es poden eliminar. En definitiva, només el camp *Hop Limit* està en la capçalera de

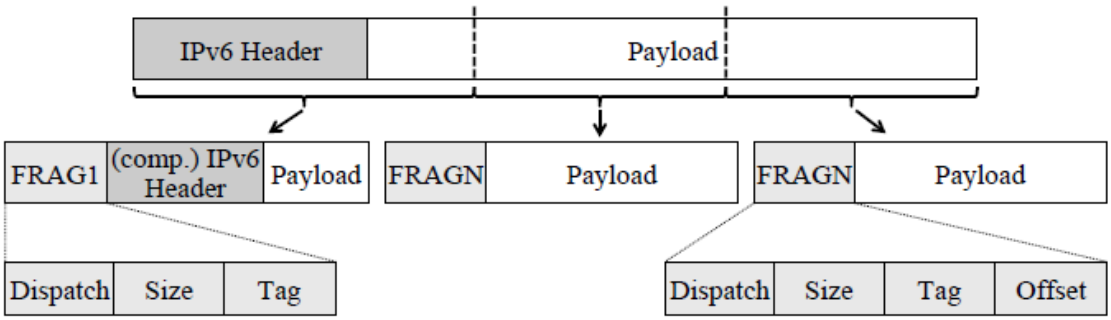
6LoWPAN. De la mateixa manera per UDP, la longitud es pot calcular a partir del camp de longitud de IEEE802.15.4. En el cas més comú, només s'utilitza un nombre limitat de ports i amb 4 bits de longitud és suficient [15].

En la il·lustració de sota es poden veure dues trames IEEE802.15.4. La primera amb totes les capçaleres de UDP i IPv6 (adreça 64 bits), deixant que al càrrega útil del paquet sigui 53 bytes. La segona després d'aplicar la màxima compressió possible del protocol 6LoWPAN. En aquest cas la informació útil pot tenir una mida màxima de 108 bytes, el doble que en el cas sense la compressió de les capçaleres IPv6 i UDP.



Il·lustració 12: *Màxima compressió capçaleres amb 6LoWPAN [16]*

En IPv6 el mínim MTU per defecte és 1280 bytes, mentre que per IEEE802.15.4 és de 127 bytes, per tant, és necessari la fragmentació de les trames IPv6 perquè puguin ser transmeses en xarxes basades en IEEE802.15.4. En la il·lustració següent es pot veure el procés de fragmentació d'un paquet IPv6 en paquets 6LoWPAN amb l'encapsulació de les capçaleres corresponents en cada cas.



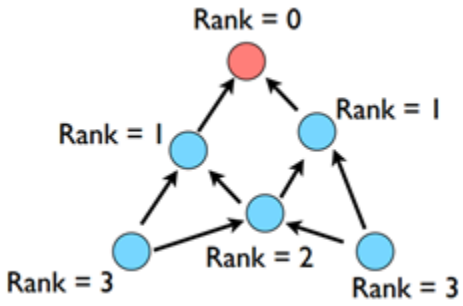
Il·lustració 13: *Exemple procés fragmentació [17]*

2.2.4 IETF RPL

RPL és un protocol d'encaminament de vector de distància definit en el RFC6550. Opera en la capa d'enrutament, per sobre de IEEE802.15.4, i és responsable de la transmissió de paquets a través de múltiples salts que separen els nodes origen i destí. La seva funció es divideix en dues parts, per una banda, disposa d'un motor de reenviament que utilitza una taula d'enrutament per decidir quin node veí és el següent salt per al paquet. D'altra banda, un protocol d'encaminament que omple les dades de la taula d'enrutament descrita anteriorment [18].

RPL està dissenyat per *Low Power and Lossy Wireless Networks* com WSN. Per tant, està optimitzat per xarxes de recollida (on els nodes envien les dades capturades periòdicament per un nombre petit de punts de recollida) amb comunicació de poc freqüent des del punt de recollida als nodes. RPL suporta trànsit de recollida *Multi-Point-to-Point* (MP2P) i trànsit de configuració *Point-To-Multi-Point* (P2MP). No suporta bé el trànsit *Point-To-Point* (P2P), encara que es poden desenvolupar solucions en aquest sentit [19].

Per al trànsit MP2P, RPL construeix un gradient a la xarxa, amb connexió a terra en els punts de recollida anomenat *Low power and lossy network Border Router* (LBR). Cada node se li assigna un *rank* que va augmentant a mesura que el node està lluny de la LBR. El gradient en RPL s'anomena *Destination Oriented Directed Acyclic Graph* (DODAG). En la il·lustració de sota es pot veure un exemple de com el *rank* es va incrementant cada vegada que el node està més lluny del LBR.



Il·lustració 14: *Exemple assignació valor rank [20]*

Els nodes que executen RPL intercanvien informació de senyalització per configurar i mantenir el DODAG a través d'un nou tipus de missatges ICMPv6 anomenat RPL *Control Message*. El codi 1B en la capçalera ICMPv6 s'utilitza per diferenciar entre els diversos subtipus. El subtipus utilitzat per construir el DODAG s'anomena DAG *Information Object* (DIO). Tots els nodes de la xarxa publiquen regularment un DIO que serveix per indicar el seu *rank*. L'instant en què un node emet un DIO es regeix pel temporitzador Trickle que es restableix només quan es presenta una inconsistència. D'aquesta forma, quan la topologia és estable l'intercanvi de DIOs és molt petit.

P2MP disposa de dos modes: *Storing Mode* i *Non-Storing Mode*. El *Non-Storing Mode*, l'adreçament origen s'utilitza quan s'emet un missatge a un node específic de la xarxa, el LBR anteposa la seqüència de nodes que cal passar per arribar a la destinació del paquet. Per aprendre aquesta seqüència, cada node de la xarxa ha de transmetre un *Destination Advertisement Object* (DAO, un altre subtipus de ICMPv6 RPL *Control Message*) al LBR indicant els seus pares i fills. El *Storing Mode* utilitza taules intermèdies en certs nodes de la xarxa. Anàlogament al comportament de *Distributed Hash Table* (DHT), les rutes són calculades parcialment en cada node intermedi que és capaç de trobar la ruta dins de la seva subxarxa fins el destí [18].

Les possibles mètriques i limitacions d'aquest protocol d'encaminament són: energia del node, nombre de salts, rendiment de l'enllaç, latència, fiabilitat de l'enllaç i color de l'enllaç. El terme color de l'enllaç en RPL es refereix a les propietats dels enllaços, en funció del color es poden utilitzar o no els enllaços pels camins [5].

2.2.5 IETF CoAP

CoAP és un protocol especialitzat en transferència web utilitzat en nodes i xarxes restringides a IoT. Està dissenyat per aplicacions M2M com *smart energy* i automatització d'edificis [21].

En juny de 2014 es va estandarditzar com RFC 7252 on el grup de treball del IETF *Constrained RESTful Environments* (CORE) va realitzar la major part del treball d'estandardització. L'objectiu del grup CORE per obtenir CoAP va ser desenvolupar un protocol amb arquitectura REST en entorns restringits (nodes i xarxes).

Una de les diferencies entre els protocols CoAP i HTTP és que CoAP envia els missatges de forma asíncrona a través del protocol UDP, mentre que HTTP estableix connexions TCP per enviar peticions i respostes.

El model de funcionament de CoAP és semblant a l'arquitectura client/servidor del protocol HTTP. La diferència en què en el model CoAP els papers de client i servidor no són tan clars. Això és perquè utilitza el protocol UDP, asíncron, i els dos punts actuen com a clients i servidors [22].

Les principals característiques del protocol són [22] [23]:

- Protocol web que compleix amb els requeriments M2M en entorns limitats.
- Connexió UDP [RFC0768] amb fiabilitat opcional que suporta sol·licituds *unicast* i *multicast*.
- Intercanvi asíncron de missatges.
- Baix *overhead* en la capçalera i baixa complexitat d'analitzar.
- Limitar la necessitat de fragmentació en xarxes 6LoWPAN.
- URI i suport *Content-type.*
- Capacitat de *chaching* i *proxy* simple.
- Mapatge HTTP sense estat a través de l'ús de *proxies* o assignació directa de les interfícies HTTP a CoAP.
- Descobriment de recursos opcional.
- Mecanisme Observe per a notificacions asíncrones.
- Seguretat vinculant a *Datagram Transport Layer Security* (DTLS) [RFC6347].

2.2.5.1 Arquitectura CoAP

CoAP té una arquitectura de dos capes [5]:

- ***Message layer:*** la seva funció és controlar els intercanvis de missatges a través de UDP entre dos *endpoints*. Les peticions i les respostes comparteixen el mateix format de missatge. Els missatges són identificats per un ID utilitzat per detectar duplicitats i per fiabilitat. Hi ha quatre tipus de missatges:
    - *Confirmable (CON)*: requereix confirmació de recepció (missatge ACK) per part del receptor.

- *Non-Confirmable (NON):* s'utilitza quan no hi ha requeriment perquè la transmissió sigui fiable.
- *Acnowledgement (ACK):* s'envien per confirmar que s'ha rebut un missatge tipus CON.
- *Reset (RST):* resposta del receptor quan no pot processar el missatge (CON o NON) rebut.

Els missatges *multicast* són del tipus NON.

- ***Request/Response layer:*** Les semàntiques de les peticions i respostes de CoAP es realitzen en els missatges de CoAP que inclouen un codi de mètode o de resposta, respectivament. Opcionalment (o per defecte) la informació de la petició i la resposta, com la URI i el tipus de contingut *payload* es realitza com opcions de CoAP. Un *Token Option* s'utilitza perquè coincideixi amb les respostes a peticions pendents de confirmar. Com CoAP s'executa sobre UDP, transport no fiable, els missatges poden arribar desordenats, duplicats o perduts sense tenir coneixement. Per tant, CoAP necessita implementar un mecanisme de fiabilitat amb les següents característiques:
  - Retransmissió fiable *stop-and-wait* amb *back-off* exponencial per missatges CON.
  - Detecció de missatges duplicats pels tipus CON i NON.
  - Suport *multicast*.

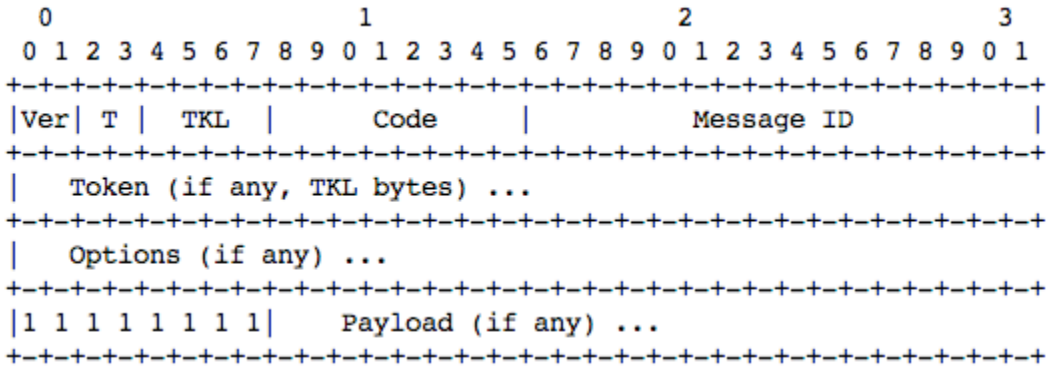
2.2.5.2 Format dels missatges CoAP

Els missatges CoAP estan codificats en un format binari simple. Un missatge consisteix en una capçalera CoAP de longitud fixa de 4 bytes seguit pel camp *Token*, les opcions en el format *Type-Length-Value* (TLV) i un *payload* (contingut del missatge). El *payload* està format pels bytes després de les opcions, si n'hi hi, i la seva longitud es calcula a partir de la longitud del datagrama. Els principals camps del missatges són [23]:

- ***Version (Ver):*** 2 bits que indiquen la versió de CoAP. Establir el valor 1 en l'actual versió.

- ***Type (T):*** 2 bits per especificar el tipus de missatge: CON (0), NON (1), ACK (2) i RST (3).
- ***Token Length (TKL):*** 4 bits que indiquen la longitud del camp *Token*.
- ***Code:*** 8 bits per definir si el missatge és una petició o una resposta. En el cas de petició, especifica els mètodes de petició (GET, POST, etc.). En el cas de resposta, indica el codi de resposta (2.01, 4.03,etc.).
- ***Message ID:*** 16 bits per definir un identificador únic, utilitzat per la detecció de duplicitats i mapejar els tipus de missatges ACK/RST a tipus de missatge CON/NON.
- ***Token:*** si existeix la seva longitud és variable fins a 8 bytes i està definida en el camp TKL. El valor del camp *Token* s'utilitza per correlar les peticions o les respostes.
- ***Options:*** opcions del missatge.
- ***Payload:*** contingut del missatge.

En la il·lustració es mostra el format dels camps d'un missatge CoAP.



Il·lustració 15: *Format d'un paquet CoAP [23]*

2.2.5.3 Mètodes CoAP

CoAP ofereix els mètodes per a una arquitectura RESTful [5].

- **GET:** permet recuperar una representació de la informació corresponent al recurs identificat per la sol·licitud URI.

  Si la petició és vàlida es contesta amb un codi de resposta 2.05 (*Content*) o 2.03 (*Valid*).

  Presenta un funcionament segur i idempotent.

- **POST:** requereix que la representació del recurs inclosa en la petició URI sigui processada. Normalment el resultat és la creació d'un nou recurs o l'actualització del recurs indicat.

  Si el recurs s'ha creat en el servidor contesta amb un codi de resposta 2.01 (*Created*) i hauria d'incloure en la resposta la URI del nou recurs. Si la petició és correcte però no s'ha pogut crear el nou recurs, el codi de resposta hauria de ser 2.04 (*Changed*). Si la petició és vàlida i el recurs ha estat eliminar, el codi de resposta ha de ser 2.02 (*Deleted*).

  No és ni segur ni idempotent.
- **PUT:** el recurs indicat en la petició URI ha de ser actualitzat o creat amb la representació inclosa en el missatge. El format d ela representació està definit pel tipus de *media* inclòs en la opció Content-Type.

  Si el recurs de la petició URI existeix, la representació inclosa ha de modificar la versió d'aquest recurs i respondre amb el codi 2.04 (*Changed*). Si el recurs no existeix, el servidor ha de crear un nou recurs amb aquesta URI i enviar el codi de resposta 2.01 (*Created*). Si el recurs no es pot crear o modificar, s'envia el codi d'error corresponent.

  No és segur però si idempotent.
- **DELETE:** el recurs indicat en la petició URI ha de ser eliminat.

  Si s'ha pogut eliminar el recurs o no existia abans de la petició s'envia un codi de resposta 2.02 (*Deleted*).

  No és segur però si idempotent.

Com s'ha pogut veure en els mètodes CoAP, les respostes estan identificades per uns codis de forma similar als codis d'estat del protocol HTTP. Es poden classificar en tres grups [5]:

- **Petició correcte:** codis 2.XX, indica que la petició ha estat rebuda, entesa i acceptada.
- **Error del client:** codis 4.XX, especifica que el client ha comès algun error.
- **Error intern del servidor:** codis 5.XX, indica que el servidor no és capaç d'atendre la petició.

2.2.5.4 Esquema URI

Les URIs de CoAP són molt similar a les de HTTP. L'esquema URI ha estat identificat per recursos CoAP i per proporcionar els mitjans necessaris per localitzar els recursos. De la mateixa forma que en les arquitectures RESTful, els recursos estan organitzats jeràrquicament en un potent servidor que escolta peticions per un determinat port [5]. L'esquema URI CoAP [24] és:

coap://host:port/recurs

El *host* pot ser una adreça IP (IPv4 i IPv6) o un nom que pugui ser resolt utilitzant un servei DNS. El *port* és el port UDP per on escolta el servidor. Per últim, *recurs* és el nom del recurs al qual es vol realitzar alguna acció. Un exemple [24] d'una URI CoAP amb IPv6 és:

coap:// fe80::c30c:0000:0000:0002:5684/HelloWorld

2.3 Sistemes operatius per IoT

La majoria de les motes són sistemes encastats de baix consum i els sistemes operatius tradicionals no són adequats. Per aquest motiu s'han desenvolupat nous sistemes operatius per aquesta arquitectura i que compleixen amb alguns dels estàndards de IoT detallats anteriorment. Molts d'ells s'han generat en entorn universitari amb llicència de codi obert. Els més destacats són: TinyOS, Contiki, OpenWSN i RiOT.

2.3.1 TinyOS

TinyOS és un sistema operatiu *open-source* desenvolupat a U.C. Berkeley per dispositius sense fils de baixa potència, per tant, d'aplicació a WSN. Està implementat utilitzant abstracció de components de programació que proporciona codi modular i facilitat de reutilització de components [6]. Està desenvolupat en el llenguatge de programació nesC, dialecte del llenguatge C.

Il·lustració 16: *Logotip TinyOS [25]*

Proporciona implementacions de capa MAC per xarxes IEEE802.15.4 i pels estàndards 6LoWPAN, RPL i CoAP a través de BLIP2.0 (Berkeley *Low-Power* IP *Stack*) [6].

TinyOS s'ha desplegat en nombroses plataformes de hardware [26] durant els últims deu anys, com per exemple, Zolertia Z1 i TMote Sky. Tots dos exemples utilitzen el microcontrolador MSP430 i la ràdio CC2420 de Texas Instruments.

2.3.2 Contiki

Contiki és un sistema operatiu *open-source* per WSN i sistemes encastats desenvolupat en 2002 pel Swedish Institute of Computer Science [6]. Es basa en un planificador multitasca col·laboratiu que utilitza abstracció *protothread*. L'ús de *protothreads* és similar a un planificador cooperatiu. Gestiona de forma eficient la memòria i el consum per incrementar la vida de les bateries.

Proporciona diverses llibreries per accedir a les funcionalitats de comunicacions. Implementa la seva pròpia capa MAC basat en CSMA/CA *preamble-sampling* utilitzant *wake-ups* periòdicament per escoltar la transmissió de paquets de les motes veïnes. La llibreria µIPv6 proporciona les funcionalitats de 6LoWPAN i RPL, a més, implementa el protocol CoAP [6].

Contiki corre sobre diverses plataformes de hardware [27], les més destacades són:

- OpenMote-CC2538: SoC (Cortex-M3 + CC2520)
- Zolertia Z1: MSP430 + CC2420 de Texas Instruments
- WiSMote: MSP430 + CC2520 de Texas Instruments
- mb851: STM32w de ST Microelectronics amb ràdio integrada a 2.4 GHz compatible amb IEEE 802.15.4.
- SEED-EYE: pic32mx795f512l + mrf24j40 de Microchip

- MICAz: Atmel ATmega128L + TI CC2420

2.3.3 RIOT

RIOT és un sistema operatiu de codi obert per a Internet of Things llicenciat sota LGPLv2. El projecte s'inicia per persones de la Freie University of Berlin, INRIA i HAW Hamburg.

Il·lustració 17: *Logotip RIOT [28]*

RIOT [28] es basa en una arquitectura *microkernel*, desenvolupat en el llenguatge C o C++ i minimitza la dependència del codi amb el hardware emprat. Proporciona mecanismes per maximitzar l'eficiència energètica, capacitat en temps real i planificador amb prioritat (preemptiu), així com suport a multifil.

Proporciona la implementació de protocols i tecnologies IoT: IEEE802.15.4, 6LoWPAN, RPL i CoAP.

RIOT es pot desplegar en plataformes de 8 bits, per exemple Arduino Mega 2560, de 16 bits, per exemple MSP430, i 32 bits, per exemple ARM. En concret, les arquitectures compatibles són:

- MSP430
- ARM7
- Cortex-M0/M3/M4
- X86

Les plataformes més significatives són:

- Arduino Due
- Arduino Mega 2560
- OpenMote-CC2538: SoC (Cortex-M3 + CC2520)

- STM32F4DISCOVERY
- TelosB
- Zolertia Z1

2.3.4 OpenWSN

OpenWSN no és exactament un sistema operatiu sinó una implementació completa *open-source* de la pila de protocols dels estàndards de *Internet of Things* (IoT). Està desenvolupat en el llenguatge de programació C per la U.C. Berkeley. Utilitza dos nivells d'abstracció [6]: hardware i Berkeley Socket.

Il·lustració 18: *Logotip OpenWSN [29]*

La pila de protocols està distribuïda en diverses capes amb el seu respectiu estàndard com es pot veure en la següent taula:

Capa	Estàndard
Aplicació	CoAP, HTTP
Transport	UDP, TCP
IP	IETF RPL
Adaptació	IETF 6LoWPAN
Accés al medi	IEEE802.15.4e
Física	IEEE802.15.4-2006

Taula 2: *Protocol estàndard per capa de OpenWSN [29]*

Algunes de les plataformes de hardware compatibles són [30]:

- TelosB
- GINA

- WSN430
- Zolertia Z1
- OpenMote-CC2538
- OpenMoteSTM
- USP Mote CC2538
- IoT-LAB_M3
- AgileFox

A més, és compatible amb els següents sistemes operatius de IoT:

- TinyOS
- Contiki
- RIOT

3 OpenWSN

En aquest apartat es realitza una explicació més detallada del projecte OpenWSN a la realitzada en l'apartat 1.3.4 OpenWSN.

OpenWSN és una implementació completa *open-source* de la pila de protocols dels estàndards de *Internet of Things* (IoT) com IEEE802.15.4e, 6LoWPAN, RPL i CoAP. OpenWSN és la primera implementació de codi obert de l'estàndard IEEE802.15.4e amb el mode TSCH[6].

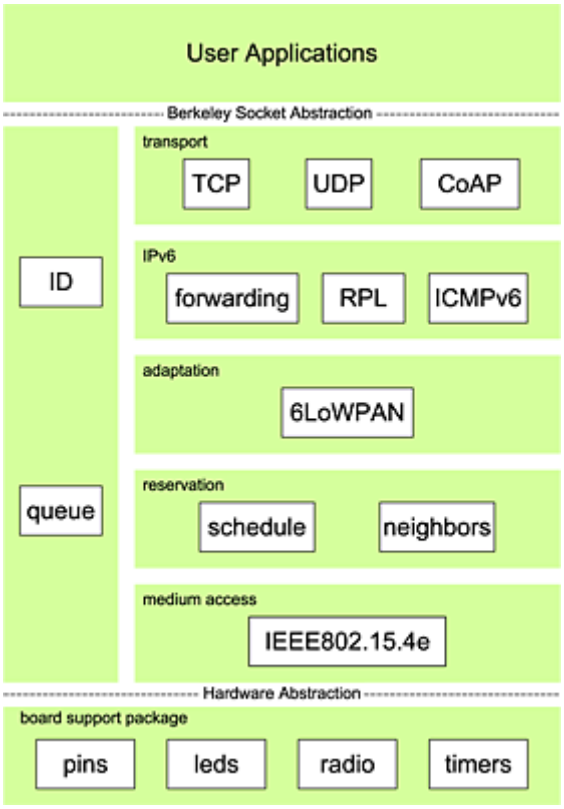
OpenWSN permet desplegar xarxes mallades amb alta fiabilitat i baix consum totalment integrades a Internet. La pila de protocols resultant serà la pedra angular de la propera revolució M2M.

3.1 Pila de protocols

Utilitza dos nivells d'abstracció [6]:

- **Berkeley Socket:** va ser desenvolupat com a part del desenvolupament del sistema operatiu Berkeley Software Distribution (BSD). El BSD va ser adoptat per tots els sistemes operatius i avui és el cor d'Internet. Considera que aplicacions en dos hosts d'Internet es comuniquen entre elles a través d'un *socket* identificat de forma unívoca per les seves adreces IP i pels ports corresponents de cada aplicació. La pila OpenWSN respecta aquesta abstracció, per tant, el desenvolupament d'una aplicació a la part superior de la pila OpenWSN és molt similar al desenvolupament d'una aplicació de qualsevol host.
- **Hardware:** consisteix a agrupar totes les funcions amb accés a hardware, per exemple funcions d'escriptura de registres, en un grup d'arxius anomenats *Board Support Package* (BSP). Això permet que la gran majoria del codi es compartit entre totes les plataformes. Hi ha un BSP per a cada plataforma suportada, la resta del codi de la pila es comparteix amb totes les implementacions.

En la següent il·lustració es pot veure els nivells d'abstracció comentats anteriorment.



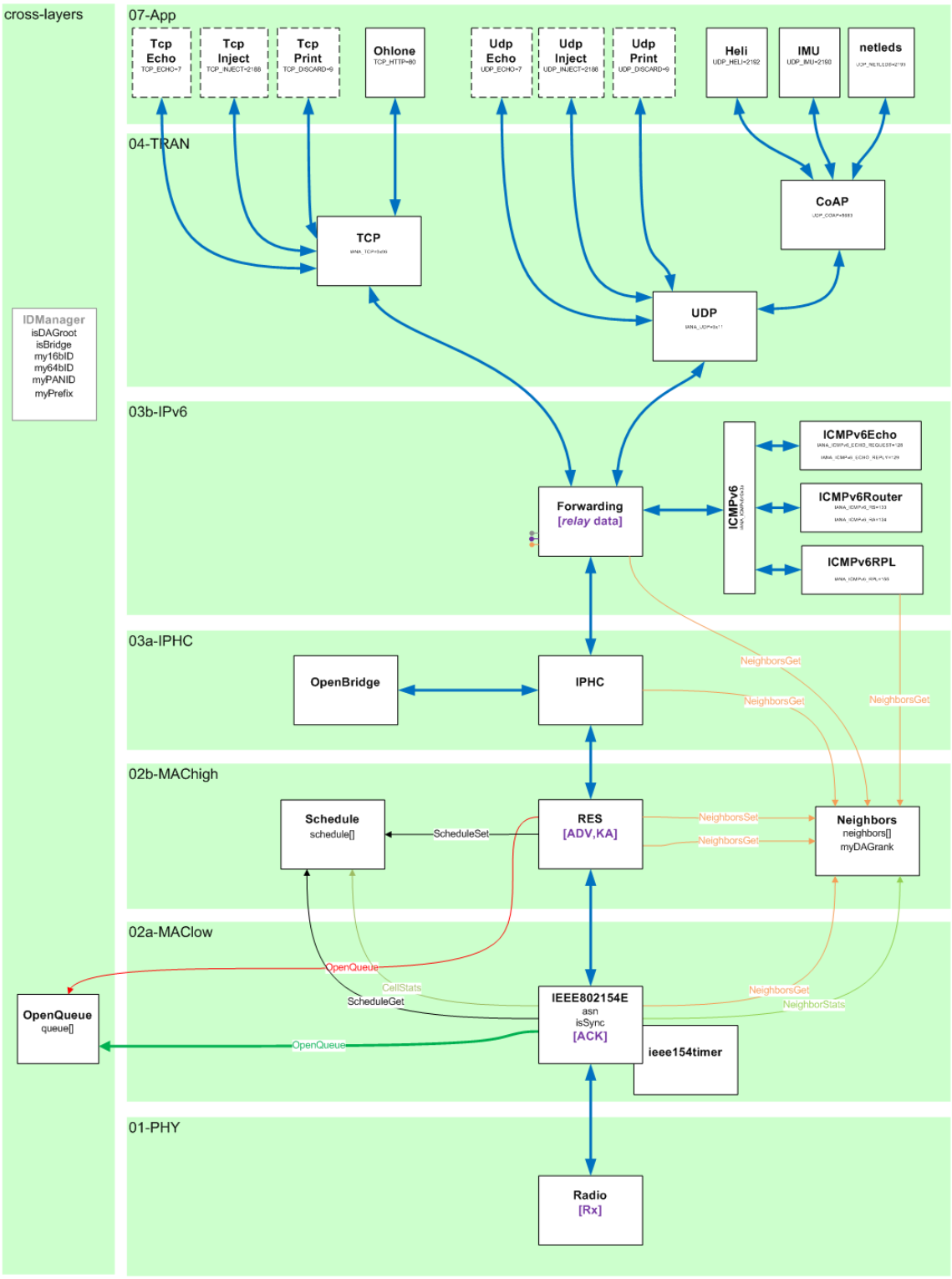
Il·lustració 19: *Nivells d'abstracció OpenWSN [6]*

La pila de protocols està distribuïda en diverses capes amb la implementació del seu respectiu estàndard com es pot veure en la següent taula:

Capa	Estàndard
Aplicació	CoAP, HTTP
Transport	UDP, TCP
IP	IETF RPL
Adaptació	IETF 6LoWPAN
Accés al medi	IEEE802.15.4e
Física	IEEE802.15.4-2006

Taula 3: *Protocol estàndard per capa de OpenWSN [29]*

En la il·lustració de sota es pot observar l'organització de la pila de protocols de OpenWSN.



Il·lustració 20: *Organització de la pila de protocols de OpenWSN [31]*

Cal destacar que la capa MAC es divideix en dues subcapes [10]:

- **02a-MAClow:** capa MAC inferior, implementa els temporitzadors associats i IEEE802.15.4e amb una màquina d'estats finits (FSM). S'alimenta de la cua de

paquets a enviar (mòdul *OpenQueue*) i del planificador de la capa MAC superior. El seu mode d'execució és a través d'interrupcions o ISR (*Interrupt Service Routine*).

- **02b-MAChigh:** capa MAC superior, és la responsable de la gestió de la planificació. S'executa en mode de tasca.

3.2 OpenOS

OpenOS és el planificador principal desenvolupat com a part del projecte OpenWSN. Les interrupcions de hardware i dels *timers* ordenen les tasques en funció de la seva prioritat i s'afegeixen a una llista de tasques.

Si hi ha tasques en la llista, el planificador crida a la funció *callback* associada a cada tasca i l'elimina del llistat. En la situació que la llista estigui buida, el planificador canvia al microcontrolador a estat *deep sleep*, a l'espera d'una nova interrupció per ser afegida a la llista de tasques.

OpenOS és col·laboratiu, és a dir, el planificador no desallotja la tasca que s'està executant fins a finalitzar el seu temps d'execució. La pila OpenWSN no està directament lligada al planificador OpenOS, per tant, la pila es pot executar com a part d'un sistema operatiu diferent.

3.3 OpenVisualizer

OpenVisualizer és un programa de visualització i depuració desenvolupat en llenguatge Python que s'executa en un ordinador per interactuar amb les motes OpenWSN connectades a ell.

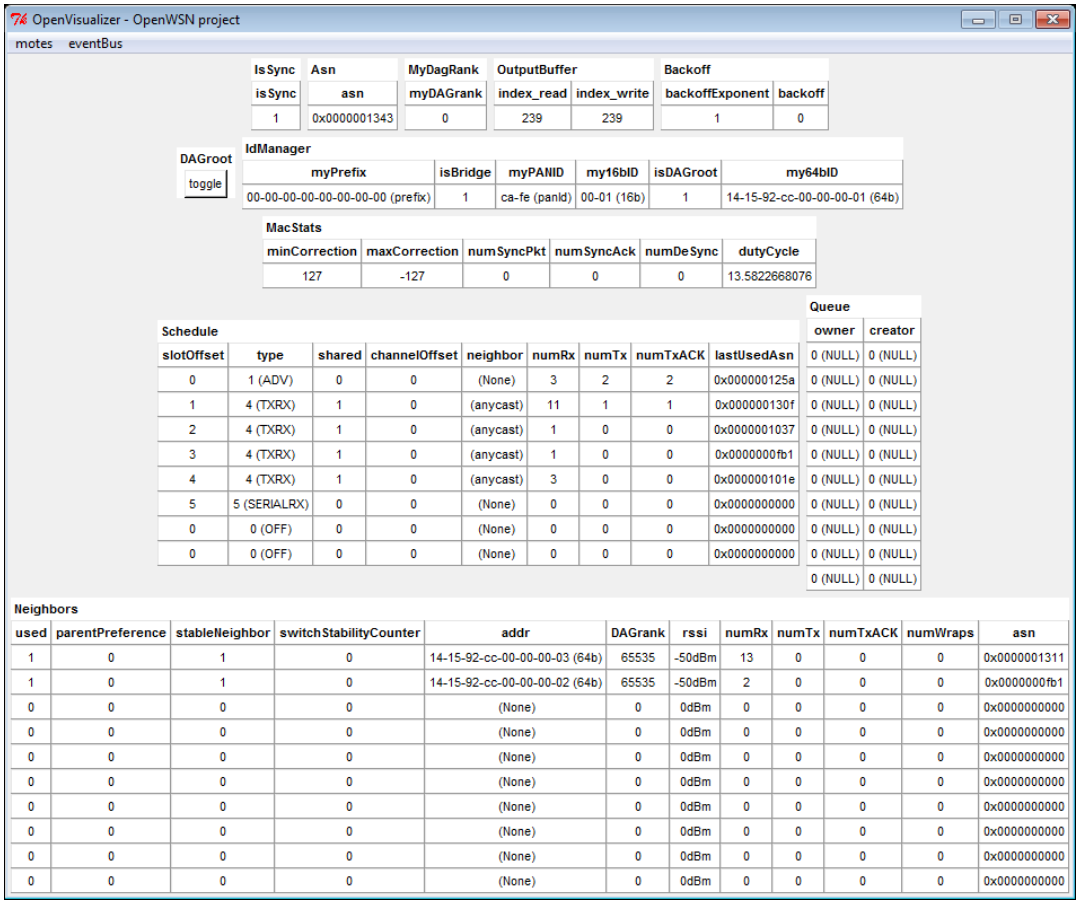
Les principals característiques són [32]:

- Connecta la xarxa OpenWSN a Internet a través d'una interfície virtual (Windows i Linux).
- Compatible amb la majoria de sistemes operatius.
- Mostra l'estat intern de cada mota connectada físicament (port sèrie) a l'ordinador on s'executa.
- Visualitza errors transmesos per les motes.

- Permet configurar DAG root [33].
- Compressió 6LoWPAN [33].
- Calcula la ruta origen RPL [33].

La comunicació amb una mota es realitza connectant la mota al ordinador a través del port sèrie. OpenVisualizer mostra informació de la xarxa i de l'estat intern de la mota (connectivitat, taules de nodes veïns i del planificador i estat de les cues), gràfics de connectivitat multi-salt i codis d'error i depuració de la mota. En la informació de connectivitat s'especifiquen les dades si la xarxa està sincronitzada, ASN, el DAG *rank*, si és DAG *root*, l'adreça IPv6, etc. En la taula de nodes veïns es mostren dades com l'adreça IPv6, el DAG *rank*, RSSI i ASN.

En la següent il·lustració es pot visualitzar un exemple de l'execució de OpenVisualizer amb tota la informació que proporciona.



Il·lustració 21: *Exemple de funcionament de OpenVisualizer [32]*

OpenVisualizer consta de mòduls bàsics i diversos tipus d'interfície d'usuari [32]:

- Interfície gràfica d'usuari (GUI)
- Interfície per línea de comandes (CLI)
- Interfície web

Per tant, hi ha tres opcions per executar OpenVisualizer:

1. Situar-se en ~/openwsn-sw/software/OpenVisualizer.
2. Executar la següent comanda en funció de la interfície:

```
sudo scons rungui|runcli|runweb
```

Il·lustració 22: *Comandes per executar OpenVisualizer*

3.4 Llibreria Python CoAP

Els desenvolupadors de OpenWSN proporcionen una llibreria CoAP desenvolupada en el llenguatge de programació Python. Implementa un client CoAP i un servidor CoAP amb un nombre arbitrari de recursos.

La llibreria permet [20]:

- Enviar missatges de tipus *Confirmable* (CON) o *Non-Confirmable* (NON).
- Parametritzar el nombre de reintents i els *timeouts*
- Suport a peticions concurrents.

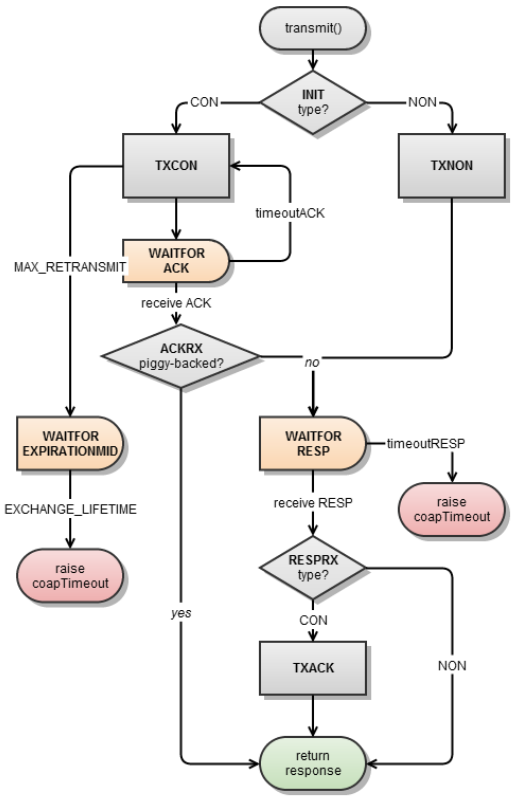
La llibreria no dona suport a funcionalitats de *caching*, *proxying*, seguretat DTLS o *multicast*.

En la següent il·lustració es pot observar la màquina d'estat de la llibreria quan ha de transmetre una petició de qualsevol dels mètodes (GET, PUT, POST i DELETE) CoAP en funció del tipus de missatge CON o NON.

La transmissió del missatge tipus CON es basa en esperar durant un període de temps (*timeout* ACK) a rebre un paquet ACK. Si dintre d'aquest període es rep el paquet ACK es retorna la resposta. Si passat el *timeout* no s'ha rebut el paquet ACK es torna a

repetir la transmissió del missatge tipus CON. La repetició de la transmissió del missatge tipus CON finalitza en arribar al nombre màxim de reintents.

La transmissió del missatge tipus NON consisteix també en esperar per rebre un paquet ACK durant el *timeout* ACK. Però en aquest cas, no hi ha repetició de la transmissió sinó arriba el paquet ACK.



Il·lustració 23: *Màquina d'estat per la transmissió de paquets CoAP de la llibreria Python CoAP de OpenWSN [34]*

4 Plataformes de hardware i software

En aquest apartat es detallen les plataformes de hardware i software utilitzades en el TFM.

4.1 Plataformes de hardware

Les plataformes de hardware utilitzades en el present TFM són OpenMote i Raspberry Pi. En els següents apartats s'expliquen les característiques més importants.

4.1.1 Plataforma OpenMote

La plataforma oberta de hardware per IoT, OpenMote [35], està formada per la mota OpenMote-CC2538, el OpenBase i el OpenBattery.

Il·lustració 24: *Logotip OpenMote [35]*

4.1.1.1 OpenMote-CC2538

La mota OpenMote-CC2538 [36] és el nucli de la plataforma de hardware OpenMote. És un SoC (*System on Chip*) de Texas Instruments (TI) format per un microcontrolador Cortex i un transceptor de ràdio. Es pot connectar als altres dos components de la plataforma de hardware OpenMote: OpenBase i OpenBattery. Si es connecta al OpenBase permet depurar els desenvolupaments de software desplegats en la mota, interconnectar diverses xarxes i treballar com a *gateway*. Si es connecta al OpenBattery, permet a la mota treballar de forma autònoma i accedir a la informació dels sensors de la OpenBattery. Està suportada pels desenvolupaments de codi obert Contiki, OpenWSN i FreeRTOS.

La mota OpenMote-CC2538 té els següents components:

CC2538: SoC de Texas Instruments format per un ARM Cortex-M3 de 32 bits i el transceptor de ràdio CC2520. El microcontrolador treballa a la freqüència de 32 MHz,

té 32 KB de RAM i 512 KB de Flash, així com perifèrics habituals (GPIOs, ADC, *Timers*, etc.). La ràdio treballa a la banda de 2.4 GHz i és completament compatible amb l'estàndard IEEE 802.15.4-2006.

TPS62730: convertidor/reductor de corrent continu de Texas Instruments amb dos modes de funcionament: regulat i derivat (*bypass*). El mode derivat connecta la tensió d'entrada de la bateria (normalment 3 V) a tot el sistema. El mode regulat redueix la tensió d'entrada (normalment 3 V) a 2.1 V. El benefici d'aquest mode de funcionament és la millora de l'eficiència global del sistema en condicions d'alta o baixa càrrega, és a dir, quan el sistema es troba dormint o quan la ràdio està transmeten o rebent dades.

ABM8G: Cristall de quars de 32 MHz de l'empresa Abracon Corporation, proporciona el senyal de rellotge del microcontrolador i del transceptor de ràdio. Té una precisió de 30ppm i el rang de temperatures de treball és de -20 °C i +70 °C.

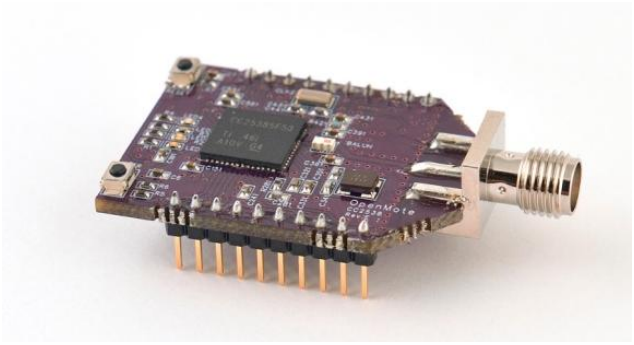
ABS07: Cristall de quars de 32.768 kHz de l'empresa Abracon Corporation, proporciona el senyal de rellotge RTC (*Real Time Clock*) del microcontrolador. Té una precisió de 10ppm i el rang de temperatures de treball és de -40 °C i +85 °C.

LEDs: 4 LEDs de diferents colors (vermell, verd, groc i taronja) de l'empresa Rohm Semiconductor i s'utilitzen per la depuració.

Botons: 2 botons de l'empresa Omron, un s'utilitza com a *reset* de la placa i l'altre està connectat a una línia de GPIO amb l'objectiu de despertar el microcontrolador quan està en mode *sleep* a través de la interrupció que es genera en prémer el botó.

Connector d'antena: connector SMA femella que permet connectar antenes externes a la placa.

XBee layout: OpenMote és totalment compatible amb factor de forma Xbee, per tant, es pot connectar fàcilment amb un ordinador a través de *Xbee Explorer Dongle* de *Sparkfun*. No obstant això, no és físicament compatible amb el *Xbee Explorer USB* donat que els pins del JTAG/USB interfereixen amb el connector USB.



Il·lustració 25: *Placa OpenMote-CC2538 [36]*

4.1.1.2 OpenBase

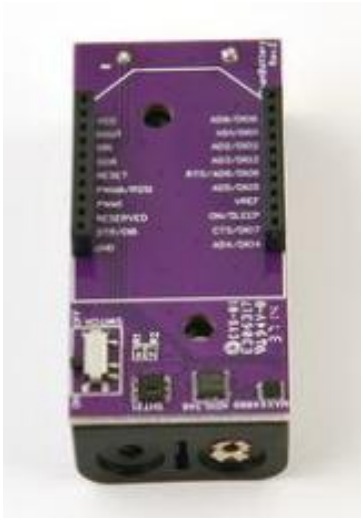
OpenBase [37] és una placa d'interfície per la mota OpenMote-CC2538. OpenMote-CC2538 es connecta a OpenBase mitjançant les capçaleres de Xbee. Inclou una sonda per mesurar el corrent consumit per OpenMote-CC2538.

OpenBase presenta tres propòsits diferents. Primer, permet programar i depurar el codi amb suport a *breakpoints* utilitzant un ARM JTAG estàndard amb connector de 10 pins (per exemple, Segger J-Link). Segon, possibilitar la comunicació amb l'ordinador a través dels ports sèrie o USB amb la finalitat de programació i depuració. El port sèrie, controlat per mitjà del xip FTDI FT232R, permet la programació a través de BSL i la depuració utilitzant la funció *printf*. El port USB, suport natiu, només possibilita la depuració utilitzant la funció *printf*. Per últim, permet la comunicació amb una xarxa Ethernet 10/100 Mbps proporcionant a la mota OpenMote-CC2538 connectivitat a Internet sense necessitat d'un ordinador. La connexió Ethernet es basa en el xip Microchip ENC28J60 i un connector estàndard RJ-45 amb protecció magnètica del circuit.

Il·lustració 26: *Placa OpenBase [37]*

4.1.1.3 OpenBattery

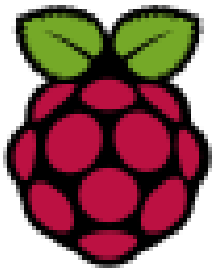
OpenBattery [38] és una placa d'interfície per la mota OpenMote-CC2538. OpenMote-CC2538 es connecta a OpenBattery mitjançant les capçaleres de Xbee. Té un porta piles 2xAAA per proporcionar la energia suficient perquè la mota OpenMote-CC2538 funcioni de forma autònoma. Disposa d'un interruptor de potència per encendre/apagar la placa, a més de tres sensors digitals: un sensor SHT21 (temperatura i humitat), un acceleròmetre (ADXL346) i un sensor de llum (MAX44009). Els sensors es connecten a la mota OpenMote-CC2538 a través del bus I2C.



Il·lustració 27: *Placa OpenBattery [38]*

4.1.2 Raspberry Pi

Raspberry Pi [39] és una sèrie d'ordinadors de placa única (SBC en anglès) de mida d'una targeta de crèdit per promoure l'ensenyament d'informàtica en les escoles.



Il·lustració 28: *Logotip Raspberry Pi [39]*

En l'actualitat hi ha tres models diferents de Raspberry Pi [40]:

- Raspberry Pi 1 Model A+
- Raspberry Pi 1 Model B+
- Raspberry Pi 2 Model B

Qualsevol model es basa en un SoC de Broadcom que conté un processador central (CPU en anglès) ARM, un processador gràfic (GPU en anglès) VideoCore IV de Broadcom i la memòria RAM (la mida varia en funció del model). Utilitza una targeta SD pel emmagatzematge permanent i disposa de connexió a Ethernet, sortida d'àudio i vídeo, port USB i GPIOs.

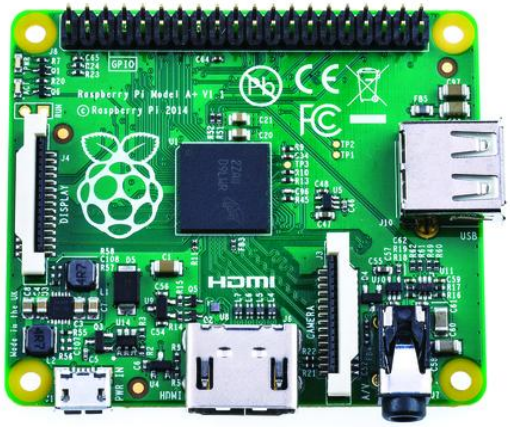
La majoria de sistemes operatius de Raspberry Pi estan basats en el *kernel* de Linux. Els més coneguts són:

- Raspbian (Debian Wheezy)/NOOBS (per a principiants)
- Android
- Firefox OS
- OpenWebOS
- OPENELEC

Tot seguit es detallen les especificacions tècniques de cada model Raspberry Pi:

4.1.2.1 Raspberry Pi 1 Model A+:

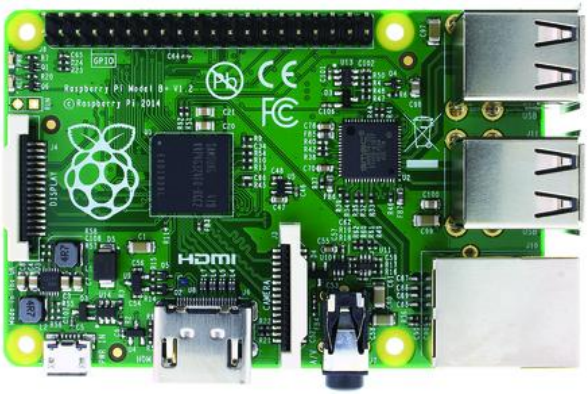
SoC	Broadcom BCM2835 (CPU + GPU + DSP + SDRAM + port USB)
CPU	ARM1176JZF-S a 700 MHz (família ARM11)
GPU	Broadcom VideoCore IV OpenGL ES 2.0 MPEG2, VC-1, 1080p30 H.254/MPEG-4 AVC
Memòria (SDRAM)	256 MB (compartit amb GPU)
Ports USB 2.0	1 (directe del xip BCM2835)
Entrada de vídeo	Connector MIPI CSI
Sortides de vídeo	Analògica: connector TRRS (PAL i NTSC) Digital: HDMI (rev 1.3 i 1.4) Interfície DSI
Sortides d'àudio	Analògica: connector de 3.5 mm Digital: HDMI
Emmagatzematge	MicroSD
Connexió de xarxa	No
Pins GPIO	40
Consum	200 mA (1W)
Alimentació	5 V via Micro USB o GPIO header
Dimensions	65 mm x56.5 mm
Pes	23 g



Il·lustració 29: *Raspberry Pi 1 Model A+ [41]*

4.1.2.2 Raspberry Pi 1 Model B+:

SoC	Broadcom BCM2835 (CPU + GPU + DSP + SDRAM + port USB)
CPU	ARM1176JZF-S a 700 MHz (família ARM11)
GPU	Broadcom VideoCore IV OpenGL ES 2.0 MPEG2, VC-1, 1080p30 H.254/MPEG-4 AVC
Memòria (SDRAM)	512 MB (compartit amb GPU)
Ports USB 2.0	4
Entrada de vídeo	Connector MIPI CSI
Sortides de vídeo	Analògica: connector TRRS (PAL i NTSC) Digital: HDMI (rev 1.3 i 1.4) Interfície DSI
Sortides d'àudio	Analògica: connector de 3.5 mm Digital: HDMI
Emmagatzematge	MicroSD
Connexió de xarxa	Ehternet 10/100 Mbit/s
Pins GPIO	40
Consum	600 mA (3 W)
Alimentació	5 V via Micro USB o GPIO header
Dimensions	85.6 mm x56.5 mm
Pes	45 g



Il·lustració 30: *Raspberry Pi 1 Model B+ [42]*

4.1.2.3 Raspberry Pi 2 Model B:

SoC	Broadcom BCM2836 (CPU + GPU + DSP + SDRAM + port USB)
CPU	Quad-core ARM Cortex A7 a 900 MHz
GPU	Broadcom VideoCore IV OpenGL ES 2.0 MPEG2, VC-1, 1080p30 H.254/MPEG-4 AVC
Memòria (SDRAM)	1 GB (compartit amb GPU)
Ports USB 2.0	4
Entrada de vídeo	Connector MIPI CSI
Sortides de vídeo	Analògica: connector TRRS (PAL i NTSC) Digital: HDMI (rev 1.3 i 1.4) Interfície DSI
Sortides d'àudio	Analògica: connector de 3.5 mm Digital: HDMI
Emmagatzematge	MicroSD
Connexió de xarxa	Ehternet 10/100 Mbit/s
Pins GPIO	40
Consum	800 mA (4 W)
Alimentació	5 V via Micro USB o GPIO header
Dimensions	85.6 mm x56.5 mm
Pes	45 g

Il·lustració 31: *Raspberry Pi 2 Model B [43]*

4.2 Plataforma de software

En aquest apartat es detallen les principals característiques de la plataforma de software thethings.iO.

4.2.1 thethings.iO

thethings.iO és una plataforma de *Cloud Computing* per IoT.

Il·lustració 32: *Logotip thethings.iO [44]*

Les seves principals característiques són [44]:

- Suport a quatre protocols IoT: REST API, MQTT, CoAP i Websockets.
- Solució *cloud* a gran escala.
- Emmagatzematge de dades amb garantia de protecció accessibles fàcilment en poc temps.
- Interoperabilitat dels objectes.
- Proporciona eines analítiques potents que es poden personalitzar per entregar informació.
- Alt grau de disponibilitat, flexibilitat, seguretat i escalabilitat.
- Accés gratuït restringit per desenvolupadors.

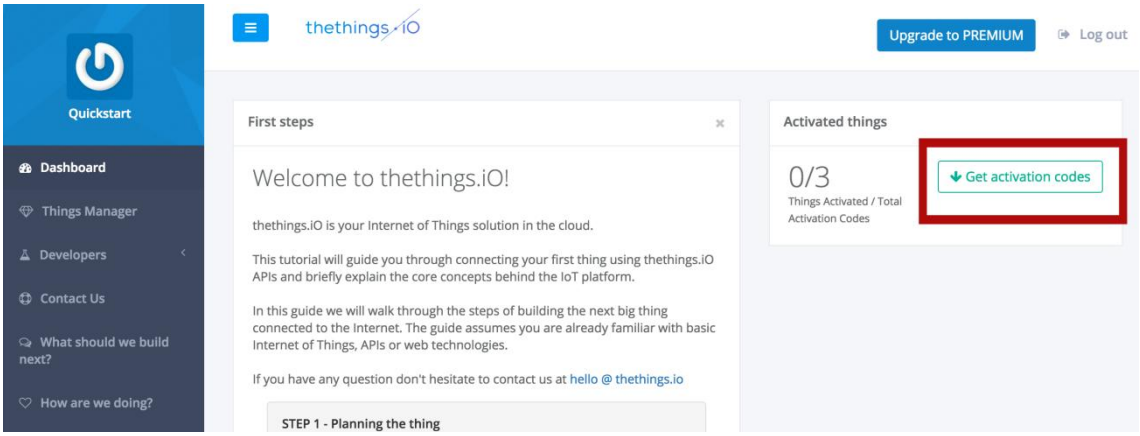
4.2.1.1 Interacció amb thethings.iO

El primer pas per utilitzar thethings.iO és crear un compte d'usuari, hi ha tres modalitats [44]: *Developer* (gratuït), *Startup* i *Forever.*

En el present Treball Final de Màster s'utilitza el protocol CoAP per la comunicació amb la plataforma de software thethings.iO. thethings.iO proporciona *endpoint* CoAP amb els mètodes [45]: *Activate*, *ThingWrite*, *ThingRead* i *ThingSubscribe*. La URL *endpoint* CoAP de thethings.iO és *coap.thethings.iO* i utilitza el port 5683.

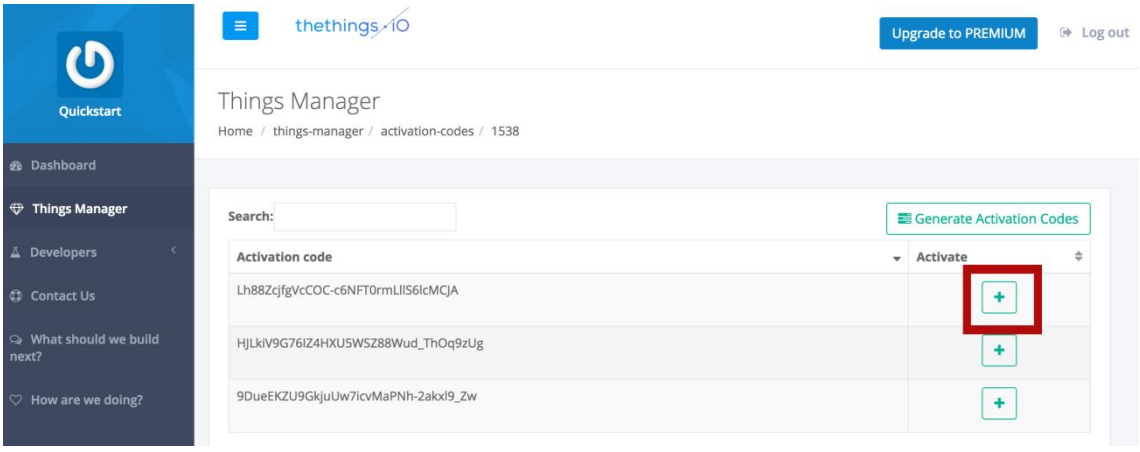
Per poder comunicar dades a la plataforma thethings.iO és necessari activar "la cosa"a partir d'un codi d'activació.

Un codi d'activació és un codi únic associat a un dispositiu que permet obtenir el seu *thing token* corresponent. Per obtenir el codi d'activació és necessari accedir al *Dashboard* de thethings.iO fent LOGIN en la seva web. Una vegada l'usuari ha accedit, ha de prémer el botó "*Get activation codes*". En la següent il·lustració [46] es pot veure el *Dashboard* de thethings.iO i el botó que cal fer clic.



Il·lustració 33: *Finestra Dashboard de thethings.iO*

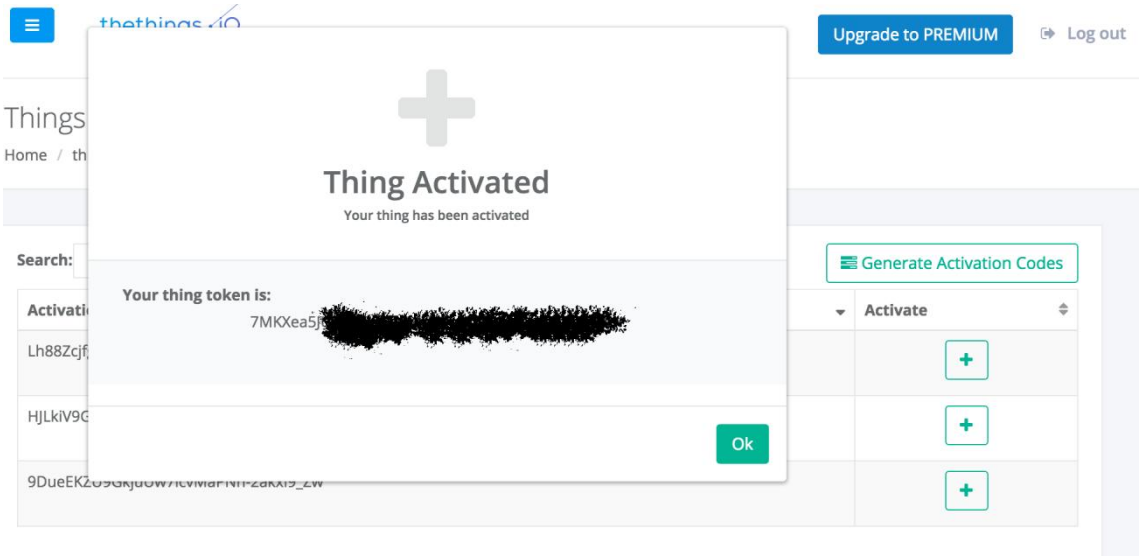
Tot seguit es mostra a la pantalla un llistat de codis d'activació, tal i com es pot visualitzar en la il·lustració [46] de sota.



Il·lustració 34: *Finestra Thing Manager de thethings.iO amb codis d'activació*

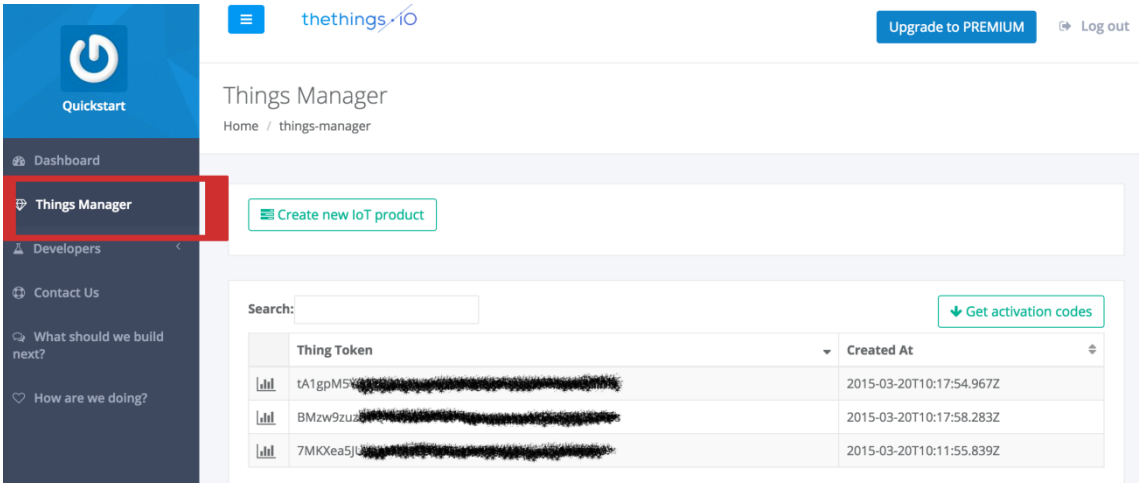
El *thing token* és també un codi únic associat a un dispositiu que el permet identificar-se en el *endpoint* CoAP. El *thing token* es pot obtenir de dues formes:

- **Panell de la consola:** accedir al *Panel* (*Things Manger*) de thethings.iO i fer clic en el botó "+" del codi d'activació desitjat, com es veu en la il·lustració anterior. Llavors s'obre una finestra indicant que l'objecte s'ha activat i el seu *thing token*, com es visualitza en la il·lustració [46] següent.



Il·lustració 35: *Finestra d'activació del "thing" amb el thing token*

En aquest moment si es torna a accedir a *Things Manger* el codi d'activació desapareix i es mostra el *thing token* amb la seva data de creació, en la il·lustració de sota es pot observar aquest fet.



Il·lustració 36: *Finestra Thing Manger de thethings.iO amb thing tokens*

- **Mètode *Activate* [45]:** realitzar la petició POST a l'adreça *coap://coap.thethings.iO/v2/things/ (endpoint* CoAP) amb el següent *payload*:

```
{
  "activationCode" : "{ {activation Code} }"
}
```

Il·lustració 37: *Payload de la petició POST d'activació*

On *activation Code* és un codi d'activació obtingut del *Panel* de thethings.iO.

Si resultat de la petició és correcte retorna el *thing token*, en cas contrari notifica que s'ha produït un error i quin és. Un exemple de resposta és:

```
# Everything has gone well
{
  "status": "created",
  "thingToken": "NwFbEqwEQ-ucyW8aET34EWi6L_tte6fumxTSU-Pkw"
}

# Error
{
  "status": "error",
  "message": "Error creating thing"
}
```

Il·lustració 38: *Exemple de resposta a una petició POST d'activació*

Una vegada activades "les coses" es pot emmagatzemar i llegir dades en la plataforma thethings.iO a través del mètodes ThingWrite i ThingRead respectivament.

- **ThingWrite:** emmagatzema registres d'informació de "la cosa" corresponent al *thing token* especificat a través d'una petició POST a l'adreça *coap://coap.thethings.iO/v2/things/{{THING_TOKEN}}* amb el següent *payload* [45]:

```
{
  "values":
   [
    {
      "key": "fun",
      "value": "9000"
    }
   ]
}
```

Il·lustració 39: *Exemple de payload per una petició POST del mètode ThingWrite*

On el valor de "key" correspon amb un recurs CoAP.

Si la petició és correcta, retorna el següent missatge [45]:

```
# Everything has gone well
{
  "status": "Success",
  "message": "Created"
}
```

Il·lustració 40: *Resposta a una petició correcta del mètode ThingWrite*

- **ThingRead:** retorna els valors especificats en la KEY (recurs) corresponents amb "la cosa" especificada en el *thing token* a través d'una petició GET a l'adreça *coap://coap.thethings.iO/v2/things/{{THING_TOKEN}}/resources/{{KEY}}.* Tot seguit es mostra un exemple de resposta [45]:

```
[{
   "key": "fun",
   "value": "9000",
   "datetime": "2015-01-30T15:35:33.000Z"
  }, ... (more values) ...
]
```

Il·lustració 41: *Exemple de resposta a una petició GET amb mètode ThingRead*

El mètode ThingRead suporta diversos paràmetres de cerca, per exemple, *coap://coap.thethings.iO/v2/things/{{thingToken}}/resources/{{key}}?limit=10*, retorna els últims 10 valors. En la següent taula es mostra els diversos paràmetres [45] possibles amb els seus valors per defecte, formats i descripcions.

Paràmetre	**Valor per defecte**	**Format**	**Descripció**
limit	1	Número	Quantitat de resultats (màx. 100)
startDate	1 Gener 1970	YYYYMMDDHHmmss	Data inicial
endDate	Fi del temps (JavaScript)	YYYYMMDDHHmmss	Data final

Taula 4: *Possibles paràmetres de cerca del mètode ThingRead*

El mètode ThingSubscribe no s'explica en aquest Treball Final de Màster perquè no s'utilitza.

4.2.1.2 Llibreria CoAP de thethings.iO

La llibreria CoAP de thethings.iO està desenvolupada en el llenguatge de programació Node.js i permet la connexió amb el *endpoint* de thethings.iO.

Per poder utilitzar la llibreria és necessari configurar un fitxer anomenat *config.json*, hi ha dues formes, una a través del codi d'activació [47]:

```
{
  "activationCode" : "one of your activation codes"
}
```

Il·lustració 42: *Configuració per defecte del fitxer config.json amb codi d'activació*

I l'altra amb el *thing token* [47]:

```
{
  "thingToken" : "one of your thing tokens"
}
```

Il·lustració 43: *Configuració per defecte del fitxer config.json amb thing token*

En el següent exemple d'ús de la llibreria, s'activa de forma automàtica "la cosa" i es llegeixen els últims 15 valors de la clau '*voltage'* [47].

```
var theThingsCoAP = require('thethingsio-coap');

var client = theThingsCoAP.createClient();

var params = {limit:15}
client.on('ready', function () {
   client.thingRead('voltage', params, function (error, data) {
      console.log(error ? error : data)
   })
})
```

Il·lustració 44: *Exemple d'ús de la llibreria CoAP de thethings.iO*

5 Configuracions prèvies

5.1 Entorn de desenvolupament

- **Màquina virtual:** sistema operatiu Ubuntu 14.04 LTS amb arquitectura de 32 bits.
- **Codi font OpenWSN:** es realitza una còpia actualitzada dels repositoris centrals de OpenWSN, https://github.com/openwsn-berkeley/openwsn-fw, a través de l'eina GIT.

```
sudo apt-get install git
git clone https://github.com/openwsn-berkeley/openwsn-fw
```

Il·lustració 45: *Instal·lació de l'eina GIT i descàrrega del firmware OpenWSN*

- **Scons:** eina de codi obert pels sistemes operatius basats en Unix amb l'objectiu de construir i instal·lar software per mitjà de *scripts* desenvolupats en Python. La seva funció és similar a la utilitat tradicional *make* i l'eina *autoconf*. La instal·lació es realitza a través de la següent comanda:

```
sudo apt-get install scons
```

Il·lustració 46: *Instal·lació eina Scons*

- **GNU ARM Gcc toolchain:** *toolchain* per la construcció i depuració de *firmware* en microcontroladors d'arquitectura ARM, OpenMote-CC2538 integra un microcontrolador ARM M3. El procediment d'instal·lació es detalla tot seguit:
  - Descarrega del paquet, versió Linux, des de la web https://launchpad.net/gcc-arm-embedded
  - Còpia del paquet en el directori /opt, descompressió i desempaquetat del mateix.

```
sudo cp gcc-arm-none-eabi-4_9-2014q4-20141203-linux.tar.bz2 /opt
cd /opt
sudo tar xjf gcc-arm-none-eabi-4_9-2014q4-20141203-linux.tar.bz2
```

Il·lustració 47: *Descàrrega i descompressió de GNU ARM Gcc toolchain*

- Afegir la localització del *toolchain* en les variables d'entorn.

```
cd /etc/profile.d/
sudo nano myenvvars.sh
ARM_GCC_HOME=/opt/gcc-arm-none-eabi-4_9-2014q4
export ARM_GCC_HOME
PATH=$JAVA_HOME/bin:$ARM_GCC_HOME/bin:$PATH
export PATH
```

Il·lustració 48: *Instal·lació GNU ARM Gcc toolchain*

- **SEGGER J-Link Driver:** *Driver* format per diverses eines com el servidor de depuració per permetre la carrega i depuració del software desenvolupat en la plataforma OpenMote. El procediment d'instal·lació és:
  - Descarrega del *driver* des de la web https://www.segger.com/jlink-software.html
  - Es selecciona la versió de Linux i l'arquitectura de 32 bits.
  - La instal·lació del paquet del *driver* pel JTAG Segger J-Link es realitza de la següent forma:

```
sudo dpkg --install jlink_4.96.12_i386.deb
```

Il·lustració 49: *Instal·lació SEGGER j-Link Driver*

El *driver* s'instal·la en el directori */opt/SEGGER/JLink*.

- **Eclipse:** programa informàtic de codi obert desenvolupat en Java per realitzar desenvolupaments de software en diversos llenguatges de programació. Disposa

de diferents entorns integrats de desenvolupament (IDE en anglès) en funció del llenguatge de programació disponibles en la web https://www.eclipse.org/downloads/. Els passos per realitzar la instal·lació són:

- Descarrega i instal·lació de *Eclipse for C/C++ Developers* pel sistema operatiu Linux i arquitectura 32 bits.

- **Configuració per desenvolupar el *firmware* de les motes:**

- Executar Eclipse i anar a *Help>Install new Software*

- Afegir un repositori amb les següents dades:

  - Name: *GNU ARM Eclipse Plug-ins*

  - Location: *http://gnuarmeclipse.sourceforge.net/updates*

- Seleccionar el nou repositori i instal·lar el *plugin* "GNU ARM C/C++ Cross Development Tools".

- Afegir una nova variable d'entorn anomenada SCONS amb valor "scons", *Window>Preferences>C/C++>Build>Environment*

- Assignar a la variable *jlink_path* la ruta d'instal·lació de SEGGER J-Link i a la variable *jlink_gdbserver* el valor "JLinkGDBServer", *Window>Preferences>Run/Debug>String Substitution*

- **Configuració per desenvolupar aplicació Python:**

- Executar Eclipse i anar a *Help>Install new Software*

- Afegir un repositori amb les següents dades:

  - Name: *Pydev*

  - Location: *http://pydev.org/updates*

- Seleccionar el nou repositori i instal·lar el *plugin* "Pydev for Eclipse".

- **OpenVisualizer:** eina principal per connectar la xarxa OpenWSN a Internet. La instal·lació consisteix en realitzar una còpia del codi font des dels repositoris centrals de OpenWSN i instal·lar les seves dependències.

```
git clone https://github.com/openwsn-berkeley/openwsn-sw
```

Il·lustració 50: *Descàrrega de OpenVisualizer del repositori de OpenWSN*

Per poder instal·lar les dependències cal instal·lar l'eina *pip* (per instal·lar paquets Python).

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

Il·lustració 51: *Instal·lació de l'eina pip*

Les dependències s'instal·len de la següent forma:

```
sudo pip install PyDispatcher
sudo pip install bottle
sudo apt-get install python-tk
```

Il·lustració 52: *Instal·lació de les dependències de OpenVisualizer*

La dependència *bottle* (*framework* web Bottle) és requeriment per utilitzar la web de OpenVisualizer, mentre que *python-tk* per la GUI.

- **Llibreria Python CoAP:** llibreria en llenguatge Python per la comunicació amb les motes a través del protocol CoAP. La instal·lació consisteix en realitzar una còpia del codi font des dels repositoris centrals de OpenWSN.

```
git clone https://github.com/openwsn-berkeley/coap
```

Il·lustració 53: *Descàrrega de la llibreria Python CoAP del repositori de OpenWSN*

- **Node.js:** entorn de desenvolupament basat en el llenguatge de programació ECMAScript. La instal·lació es realitza amb la següent comanda:

```
sudo apt-get install nodejs
```

Il·lustració 54: *Instal·lació de Node.js en l'entorn de desenvolupament*

- **Llibreria CoAP thethings.iO:** llibreria CoAP de thethings.iO en llenguatge Node.js. El procediment d'instal·lació és:

```
sudo apt-get install npm
sudo npm install thethingsio-coap
```

Il·lustració 55: *Descàrrega de la llibreria CoAP de thethings.iO*

- **Framework Qt:** *framework* per desenvolupar aplicacions d'interfície gràfica d'usuari. Disponible en llenguatges de programació C++ i Python. La instal·lació es realitza a través de la següent comanda:

```
sudo apt-get install python-qt4
```

Il·lustració 56: *Instal·lació del framework Qt en l'entorn de desenvolupament*

- **Programa Qt Creator:** Qt Cretor és un IDE multiplataforma per desenvolupar aplicacions amb el *framework* Qt. La instal·lació es pot realitzar des de Ubuntu Software Center.
- **Llibreria Matplotlib:** és una llibreria per generar gràfics en el llenguatge de programació Python. Abans de la instal·lació de la llibreria Matplolib, cal instal·lar les següents dependències:

```
sudo apt-get install libpng-dev
sudo apt-get install libjpeg8-dev
sudo apt-get install libfreetype6-dev
```

Il·lustració 57: *Instal·lació de les dependències de Matplotlib*

La instal·lació de la llibreria Matplolib es realitza a través de l'eina *pip*:

```
sudo pip install matplotlib
```

Il·lustració 58: *Instal·lació de Matplotlib en l'entorn de desenvolupament*

5.2 Raspberry Pi

- **Carregar imatge sistema operatiu en la targeta SD:** descarregar la imatge Raspbian des de la web http://www.raspberrypi.org/downloads/ i utilitzar l'eina *Win32DiskImager* per gravar-la en la targeta SD.

- **Configuració inicial:** configuració inicial d'una nova instal·lació de Raspberry Pi. La primera vegada que arrenca la imatge carregada en la targeta SD cal realitzar una expansió del sistema de fitxers.

```
sudo raspi-config
sudo apt-get install rpi-update
sudo rpi-update
sudo reboot
```

Il·lustració 59: *Configuració inicial de la imatge Raspbian*

- **Actualitzar el software i el sistema operatiu:**

```
sudo apt-get update
sudo apt-get upgrade
sudo reboot
```

Il·lustració 60: *Actualització imatge Raspbian*

- **Instal·lar *driver* PiScreen:** els passos per la instal·lació són:

  - Instal·lació del *driver* des del repositori de Notro:

```
sudo REPO_URI=https://github.com/notro/rpi-firmware rpi-update
```

Il·lustració 61: *Instal·lació del driver de PiScreen*

  - Activar SPI: afegir "dtparam=spi=on" en el fitxer */boot/config.txt*

- Carregar el *driver* en el procés d'arrencada: afegir els següents valors en el fitxer */etc/modules* i reiniciar el sistema.

```
flexfb width=320 height=480 regwidth=16 init=-1,0xb0,0x0,-1,0x11,-2,250,-
1,0x3A,0x55,-1,0x36,0x28,-1,0xC2,0x44,-1,0xC5,0x00,0x00,0x0,0x0,-
1,0xE0,0x0F,0x1F,0x1C,0x0C,0x0F,0x08,0x48,0x98,0x37,0x0A,0x13,0x04,0x11,0x0D,
0x00,-
1,0xE1,0x0F,0x32,0x2E,0x0B,0x0D,0x05,0x47,0x75,0x37,0x06,0x10,0x03,0x24,0x20,
0x00,-
1,0xE2,0x0F,0x32,0x2E,0x0B,0x0D,0x05,0x47,0x75,0x37,0x06,0x10,0x03,0x24,0x20,
0x00,-1,0x11,-1,0X29,-3
fbtft_device debug=3 rotate=90 name=flexfb speed=16000000
gpios=reset:25,dc:24,led:22
ads7846_device gpio_pendown=17 verbose=3 x_plate_ohms=100
pressure_max=255 swap_xy=1
```

Il·lustració 62: *Valors a afegir a /etc/modules*

- **Activar X windows en PiScreen:** el procediment d'activació és:
  - Desactivar el *driver* de *framebuffer*: comentar la línea de text "`Option "fbdev" "/dev/fb0" "` afegint un coixinet (#) al principi del fitxer */usr/share/X11/xorg.conf.d/99-fbturbo.conf*
  - Instal·lar els prerequisits pel calibratge:

```
sudo apt-get install libtool libx11-dev xinput autoconf libx11-dev libxi-dev
x11proto-input-dev -y
```

Il·lustració 63: *Instal·lació prerequisits calibratge pantalla tàctil*

  - Descarregar i instal·lar *xinput_calibrator*:

```
git clone https://github.com/tias/xinput_calibrator
cd xinput_calibrator/
./autogen.sh
make
sudo make install
```

Il·lustració 64: *Descàrrega i instal·lació eina xinput_calibrator*

- Descarregar i configurar el *script* de calibratge:

```
wget http://ozzmaker.com/piscreen/xinput_calibrator_pointercal.sh
sudo cp ~/xinput_calibrator_pointercal.sh
/etc/X11/Xsession.d/xinput_calibrator_pointercal.sh

sudo sh -c 'echo "sudo /bin/sh
/etc/X11/Xsession.d/xinput  calibrator  pointercal.sh" >>
```

Il·lustració 65: *Descàrrega i configuració script calibratge*

- Inici X Windows:

```
FRAMEBUFFER=/dev/fb1 startx
```

Il·lustració 66: *Configurar inici X Windows*

- **Iniciar automàticament X windows en PiScreen en el procés d'arrencada:** per realitzar aquesta configuració cal seguir els següents passos:
  - Modificar el fitxer */etc/inittab*: comentar amb coixinet (#) la línea "1:2345:respawn:/sbin/getty 115200 tty1" i afegir la següent *"1:2345:respawn:/bin/login –f pi tty1 /dev/tty1 2>&1"*
  - Modificar el fitxer */etc/rc.local*: afegir la següent línea *"su –l pi –c "env FRAMEBUFFER=/dev/fb1 startx &""* al final del fitxer, abans de la sentència *exit*.
- **OpenVisualizer:** eina principal per connectar la xarxa OpenWSN a Internet. La instal·lació consisteix en realitzar una còpia del codi font des dels repositoris centrals de OpenWSN i instal·lar les seves dependències.

```
git clone https://github.com/openwsn-berkeley/openwsn-sw
```

Il·lustració 67: *Descàrrega de OpenVisualizer del repositori de OpenWSN*

Per poder instal·lar les dependències cal instal·lar l'eina *pip* (per instal·lar paquets Python).

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

Il·lustració 68: *Instal·lació de l'eina pip*

Les dependències s'instal·len de la següent forma:

```
sudo pip install PyDispatcher
sudo pip install bottle
```

Il·lustració 69: *Instal·lació de les dependències de OpenVisualizer*

- **Activar IPv6:** per activar automàticament IPv6 en el procés d'arrencada de Raspbian és necessari afegir *ipv6* en l'ultima línea del fitxer */etc/modules.*

- **Node.js:** la instal·lació es realitza amb la següent comanda:

```
sudo wget http://node-arm.herokuapp.com/node_latest_armhf.deb
sudo dpkg -i node_latest_armhf.deb
```

Il·lustració 70: *Instal·lació de Node.js en la Raspberry Pi*

- **Framework Qt:** La instal·lació es realitza a través de la següent comanda:

```
sudo apt-get install python-qt4
```

Il·lustració 71: *Instal·lació del framework Qt en la Raspberry Pi*

- **Llibreria Matplotlib:** La instal·lació es realitza a través de la següent comanda:

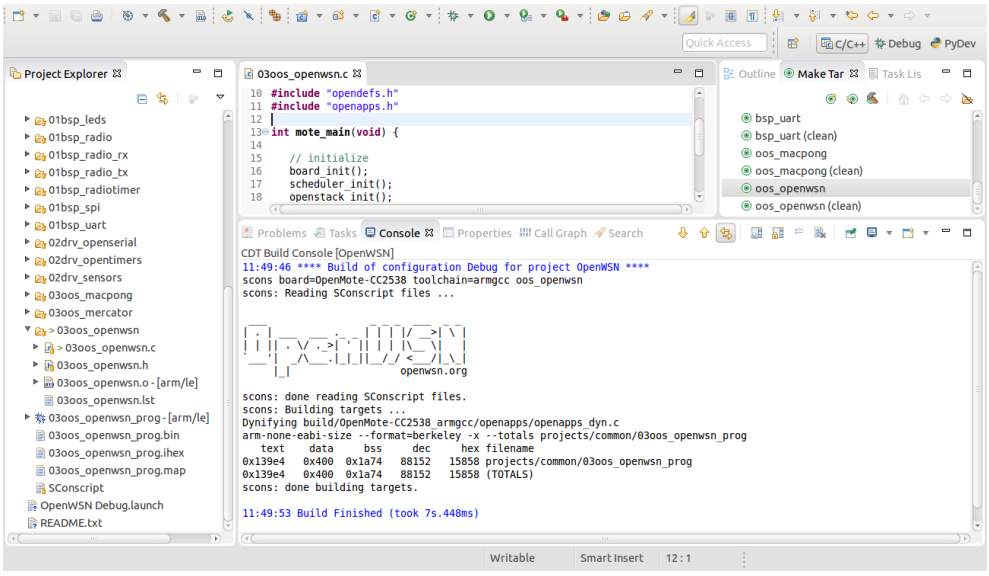
```
sudo apt-get install python-matplotlib
```

Il·lustració 72: *Instal·lació de Matplotlib en la Raspberry Pi*

5.3 Sincronitzar xarxa OpenWSN

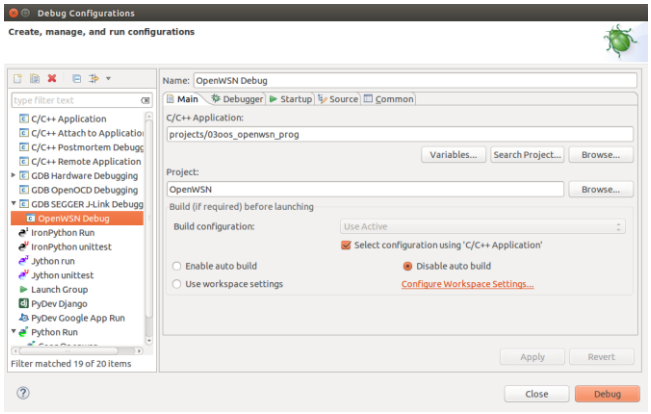
Per poder sincronitzar la xarxa OpenWSN primer és necessari carregar el *firmware* a les OpenMote-CC2538. Per carregar el *firmware* cal seguir els següents passos:

- Copiar la carpeta *openwsn-fw* amb el codi font de OpenWSN en el *workspace* de Eclipse.
- Obrir Eclipse i accedir a l'opció *Import* a través de *File*.
- Seleccionar *Existing Projects into Workspace* i fer clic en *Next*.
- En la opció de *Select root directory* prémer en el botó *Browse* i seleccionar *openwsn-fw/projects/OpenMote-CC2538/OpenWSN*. Després fer clic a *Finish*.
- Compilar el projecte cal anar a la pestanya de "Make Target" i fer doble clic sobre el *target oos_openwsn*. Si es vol realitzar un *clean* del projecte es fa doble clic sobre el *target oos_openwsn(clean).*



Il·lustració 73: *Entorn de desenvolupament amb l'eina Eclipse*

- Una vegada el codi està compilat es configura el GDB SEGGER J-Link Debugging de la següent forma:
  - Menú *Run > Debug Configurations*
  - Seleccionar *GDB SEGGER J-Link Debugging* i crear un nou llançador de configuració fent clic en el primer botó de l'esquerra de dalt.
  - Omplir els camps com es mostra en la il·lustració següent:

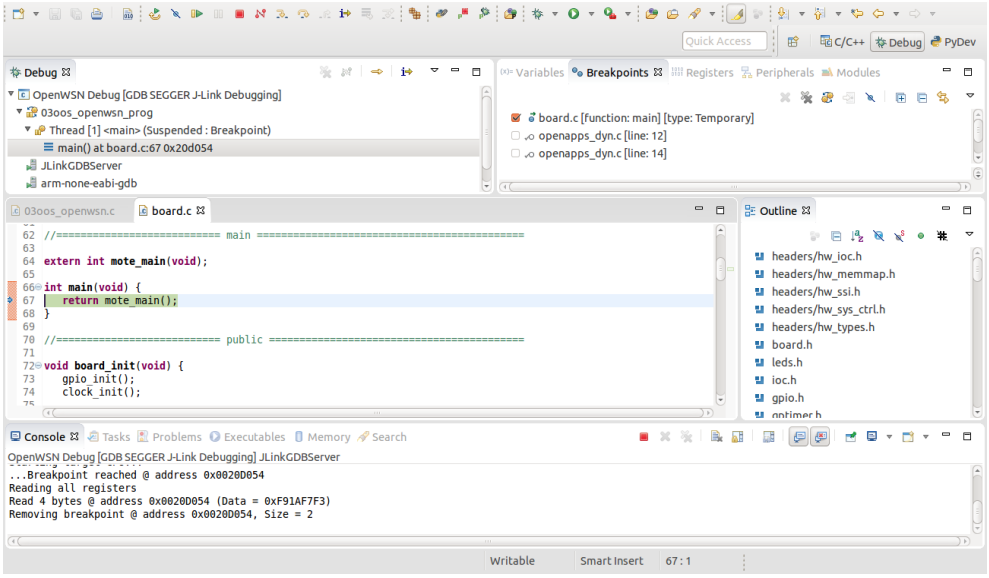


Il·lustració 74: *Finestra de configuració de l'opció Debug*

- Connectar el JTAG J-Link a l'ordinador a través del port USB.
- Connectar un OpenMote-CC2538 a un OpenBase, connectar el OpenBase al ordinador a través del port USB_FTDI i seleccionar en el *switch* USB_FTDI, així com connectar el JTAG J-Link al OpenBase a través dels pins JTAG.

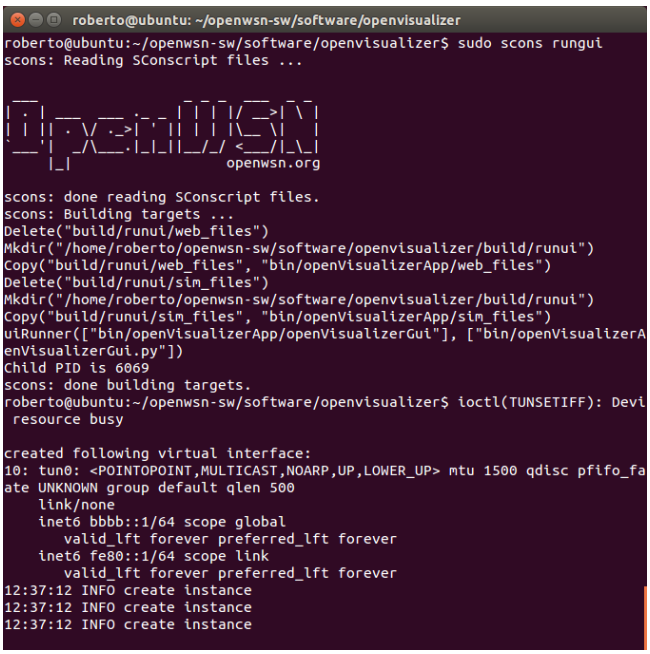
Il·lustració 75: *Connexió del JTAG J-Link amb la placa OpenBase*

- En Eclipse realitzar el *debug* de OpenWSN Debug. Continuar l'execució quan es queda parat en el punt d'interrupció fent clic a F8.



Il·lustració 76: *Finestra de Eclipse amb la perspectiva Debug*

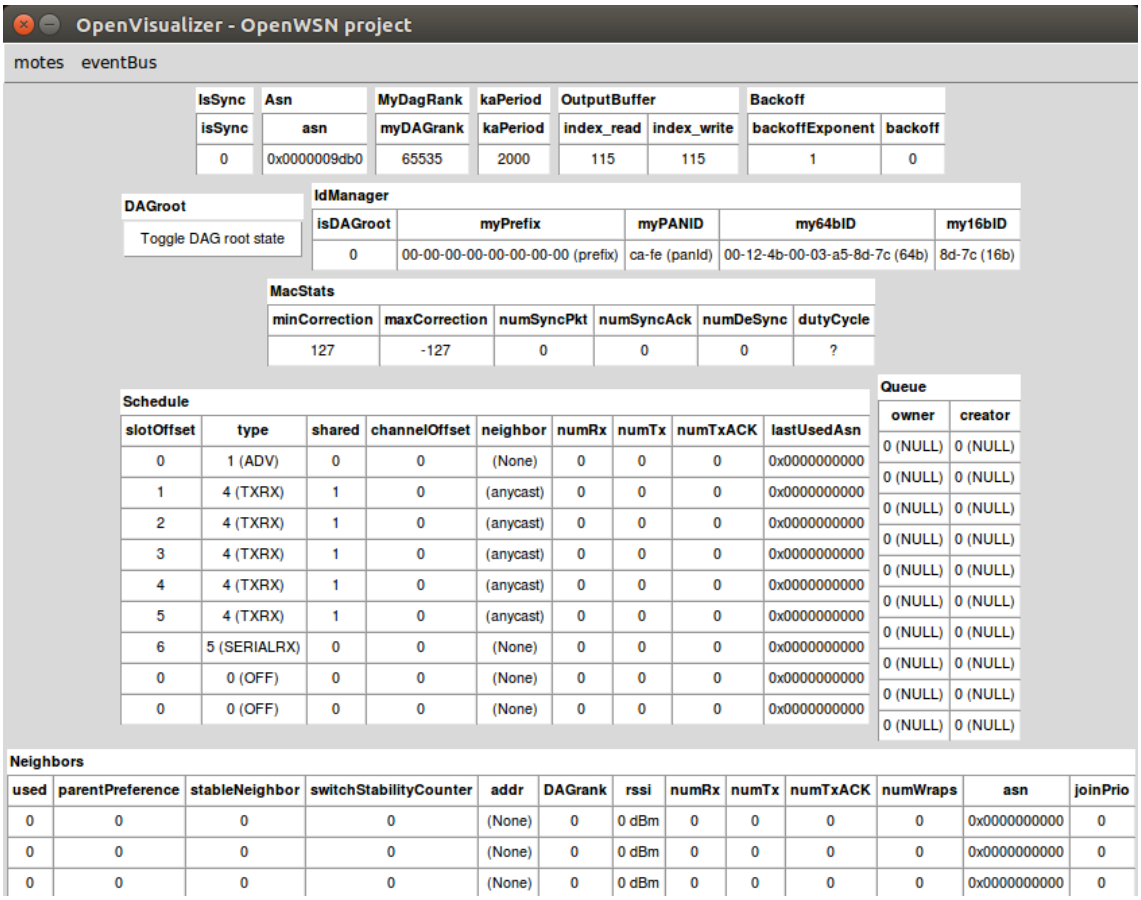
Després és necessari que dues motes OpenMote-CC2538 tinguin el codi carregat. Una mota es connecta al OpenBase, el OpenBase al ordinador a través del port USB_FTDI i seleccionar en el *switch* USB_FTDI. L'altre mota es connecta a un OpenBattery amb piles i es selecciona ON en el *switch*.



Il·lustració 77: *Execució de OpenVisualizer en mode GUI*

Posteriorment s'executa OpenVisualizer, per exemple en mode GUI. En la il·lustració de dalt s'observa com OpenVisualizer crea una interfície virtual anomenada *tun0* amb

adreçament IPv6. Posteriorment s'obre una finestra de l'aplicació, es selecciona la mota connectada i OpenVisualizer obre una nova finestra com es mostra en la il·lustració següent:

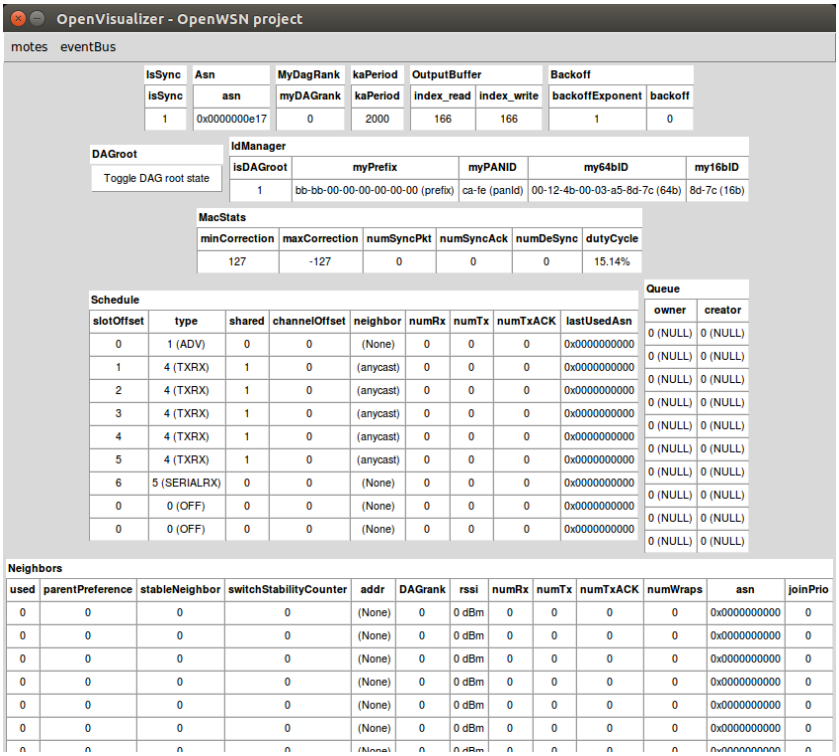


Il·lustració 78: *Finestra de OpenVisualizer*

En la il·lustració de sota es pot veure com les dues motes només tenen encesos els LEDs de color verd. Això indica que les dues plaques estan en funcionament però la xarxa OpenWSN no està sincronitzada.

Il·lustració 79: *Xarxa OpenWSN sense sincronitzar*

Per últim, es fa clic en botó *Toggle DAG root state* de OpenVisualizer. En aquest moment es modifica els valors de *IsSync=1*, *myDAGrank=0* (DAGroot) i *isDAGroot=1*. Encara no hi ha cap veí en la taula de veïns. En la il·lustració de sota es poden veure aquests canvis en OpenVisualizer.



Il·lustració 80: *Finestra OpenVisualizer amb DAGroot configurat*

Una vegada configurada la mota connectada a OpenVisualizer com a *DAGroot*, s'inicia la xarxa OpenWSN amb només aquesta mota. Aquest canvi provoca que s'encenguin

els LEDs taronja i vermell, com es mostra en la il·lustració de següent. El LED taronja indica que la mota està sincronitzada, mentre que el LED vermell indica el funcionament del transceptor de ràdio.

Il·lustració 81: *Xarxa OpenWSN en procés de sincronització*

Després d'uns instants, s'afegeix a OpenVisualizer un veí a la taula de veïns amb informació de la IPv6, DAGrank, RSSI, etc. com es pot observar en la il·lustració següent:

Neighbors

used	parentPreference	stableNeighbor	switchStabilityCounter	addr	DAGrank	rssi	numRx	numTx	numTxACK	numWraps
1	0	1	0	00-12-4b-00-03-a5-90-ea (64b)	65535	-23 dBm	2	0	0	0
0	0	0	0	(None)	0	0 dBm	0	0	0	0
0	0	0	0	(None)	0	0 dBm	0	0	0	0

Il·lustració 82: *Finestra OpenVisualizer amb un veí*

En la següent il·lustració es pot veure com la mota connectada a OpenBattery també té encesos els LEDs de color vermell i taronja.

Il·lustració 83: *Xarxa OpenWSN sincronitzada*

En aquest moment la xarxa OpenWSN de dues motes està sincronitzada. Per verificar-ho es realitza un *ping* a l'adreça IP de la mota veïna.

```
roberto@ubuntu: ~
roberto@ubuntu:~$ ping6 bbbb::0012:4b00:03a5:90ea
PING bbbb::0012:4b00:03a5:90ea(bbbb::12:4b00:3a5:90ea) 56 data bytes
64 bytes from bbbb::12:4b00:3a5:90ea: icmp_seq=1 ttl=64 time=317 ms
64 bytes from bbbb::12:4b00:3a5:90ea: icmp_seq=2 ttl=64 time=303 ms
64 bytes from bbbb::12:4b00:3a5:90ea: icmp_seq=3 ttl=64 time=296 ms
64 bytes from bbbb::12:4b00:3a5:90ea: icmp_seq=4 ttl=64 time=285 ms
64 bytes from bbbb::12:4b00:3a5:90ea: icmp_seq=5 ttl=64 time=274 ms
64 bytes from bbbb::12:4b00:3a5:90ea: icmp_seq=6 ttl=64 time=267 ms
64 bytes from bbbb::12:4b00:3a5:90ea: icmp_seq=7 ttl=64 time=261 ms
^C
--- bbbb::0012:4b00:03a5:90ea ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6001ms
rtt min/avg/max/mdev = 261.858/286.537/317.023/18.630 ms
roberto@ubuntu:~$
```

Il·lustració 84: *Ping a la mota veïna*

6 Desenvolupament

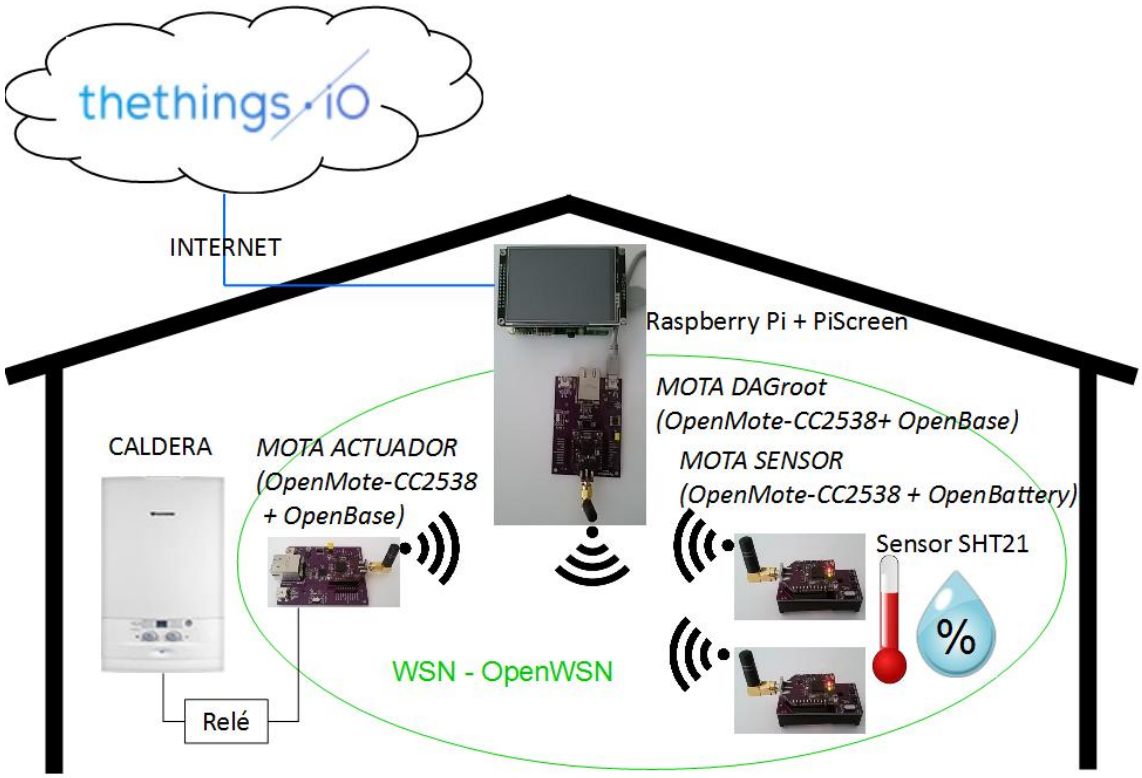
El desenvolupament del present Treball Final de Màster consisteix en un demostrador per gestionar el sistema de calefacció de la llar utilitzant la plataforma de hardware OpenMote, la implementació *open-source* de la pila de protocols basada en els estàndards de IoT, OpenWSN, i la plataforma de *cloud* per IoT, thethings.iO. El demostrador consta d'un element central i una xarxa de sensors (WSN) desplegada en la llar amb capacitat de prendre mesures de temperatura i humitat relativa. La WSN es basa en OpenWSN sobre les motes OpenMote. Les mesures capturades s'emmagatzemen en la plataforma thethings.iO.

L'element central està format per una Raspberry Pi amb connexió a Internet i dels mòduls de hardware OpenBase i la mota OpenMote-CC2538 units. Aquesta unió es connecta pel port USB a la Raspberry Pi, per permetre la comunicació amb la xarxa de sensors i realitzar la funció de *DAGroot* dintre de la WSN. A més, disposa d'una pantalla tàctil (PiScreen) connectada a la Raspberry Pi.

La xarxa de sensors consta de dos tipus de motes, una formada pel hardware OpenBattery i la mota OpenMote-CC2538 funcionant com a sistema autònom, a partir d'aquest moment s'anomena a aquest tipus de mota, *mota sensor*. La *mota sensor* obté valors de temperatura i humitat del sensor SHT21 integrat en OpenBattery. L'altra mota consta del hardware OpenBase i la mota OpenMote-CC2538, en endavant s'anomenarà *mota actuador*. La *mota actuador* realitza funcions d'actuació sobre el sistema de calefacció a través d'un relé controlat per un GPIO de la mota. El demostrador està format per dos *motes sensors* i una *mota actuador*.

La plataforma thethings.iO identifica cada mota amb un identificador únic anomenat *thing token*, el qual és necessari per qualsevol comunicació amb la plataforma.

En la següent il·lustració es pot veure un esquema de l'arquitectura del demostrador desenvolupat explicada anteriorment.



Il·lustració 85: *Esquema general del demostrador*

En la Raspberry Pi s'executa OpenVisualizer de OpenWSN en mode web que realitza la funció de *Low-power Border Router* (LBR), una aplicació en llenguatge Python anomenada *openDemo*, una aplicació basada en el *framework* Qt en Python com interfície gràfica d'usuari (GUI) anomenada *openDemoGUI* i dues aplicacions desenvolupades en Node.js per interactuar amb la plataforma thethings.iO. Una per emmagatzemar i una altra per llegir les dades anomenades Write.js i Read.js respectivament. A més, disposa d'una base de dades SQLite on s'emmagatzema el valor de temperatura de consigna; l'adreça IPv6, el *thing token*, el tipus de mota i la descripció per a cada node de la WSN.

L'aplicació *openDemo* utilitza la llibreria Python CoAP de OpenWSN per comunicar-se amb les motes. Les seves funcions són:

- Capturar cada minut les dades de temperatura i humitat realitzant peticions CoAP amb mètode GET a cada *mota sensor*.
- Enviar les dades capturades en format JSON a l'aplicació Write.js a través d'un *socket*.

- Controlar el sistema de calefacció a partir de les dades de temperatura i de l'estat del sistema de calefacció, enviant una petició CoAP amb mètode PUT amb l'acció a fer a la *mota actuador*.
- Afegir de forma automàtica en la base de dades l'adreça IPv6 de les motes.

L'aplicació Write.js rep les dades de l'aplicació *openDemo* i les envia a la plataforma thethings.iO pel seu emmagatzematge a través de la llibreria Node.js CoAP de thethings.iO.

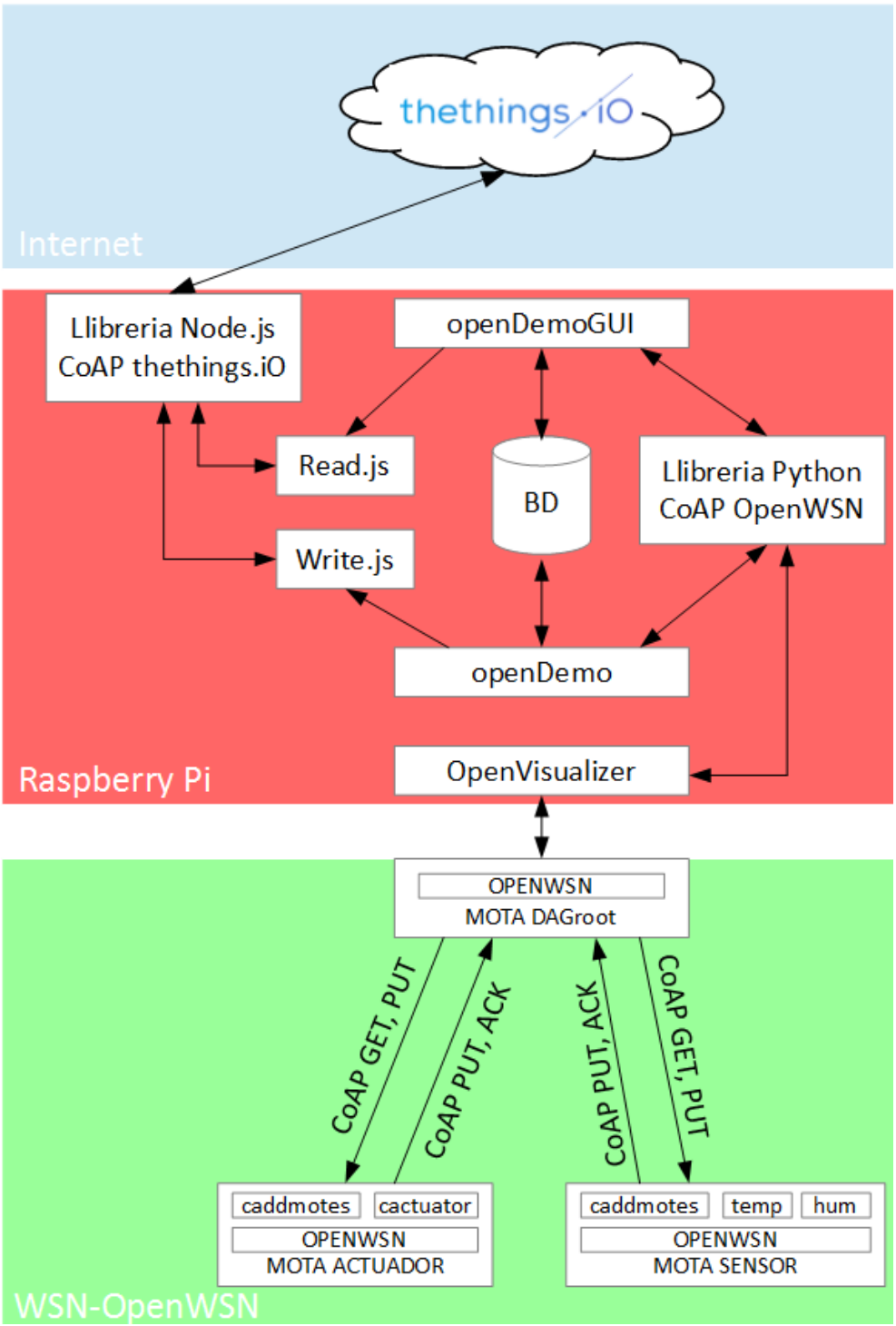
openDemoGUI permet visualitzar, configurar i emmagatzemar en la base de dades el valor de temperatura de consigna; el *thing token*, el tipus de mota i la descripció per a cada node de la WSN. A més, permet visualitzar l'històric de temperatura i humitat relativa de cada node obtenint les dades de la plataforma thethings.iO mitjançant la comunicació amb un *socket* amb l'aplicació Read.js. Així com, capturar dades de temperatura i humitat de forma instantània a través de peticions CoAP amb mètode GET (llibreria Python CoAP de OpenWSN) a qualsevol mota.

Les motes de la WSN executen quatre aplicacions CoAP per donar resposta a les funcions del demostrador. Les aplicacions CoAP s'han desenvolupat per sobre del *firmware* de OpenWSN aprofitant el nivell d'abstracció *Berkeley Socket*. Les aplicacions CoAP són:

- ***ctemp:*** permet rebre peticions CoAP amb mètode GET i la seva resposta (ACK) és el valor de temperatura obtingut del sensor SHT21.
- ***chum:*** permet rebre peticions CoAP amb mètode GET i la seva resposta (ACK) és el valor d'humitat relativa obtingut del sensor SHT21.
- ***cactuator:*** permet rebre peticions CoAP amb mètode GET per conèixer l'estat del sistema de calefacció i amb mètode PUT per actuar sobre el mateix.
- ***caddmotes:*** envia peticions CoAP amb mètode PUT cada 30 segons perquè l'aplicació *openDemo* afegeixi la seva adreça IPv6 en la base de dades.

Les motes del tipus *mota sensor* utilitzen les aplicacions *ctemp*, *chum* i *caddmotes*. La *mota actuador* empra les aplicacions *caddmotes* i *cactuator*.

En la il·lustració de sota es pot observar un esquema conceptual de les diferents aplicacions desenvolupades, de les aplicacions de software de OpenWSN (OpenVisualizer i *firmware* OpenWSN) emprades i de les llibreries CoAP (OpenWSN i thethings.iO) utilitzades.



Il·lustració 86: *Esquema general del software del demostrador*

A mode de resum, les funcions més importants del demostrador són:

- Captura de dades de temperatura i humitat. Inclou alta automàtica d'adreces IPv6 de les motes.
- Emmagatzematge de les dades a thethings.iO.
- Control del sistema de calefacció.
- Visualització i configuració de dades a través de la GUI.

En els apartats següents a Justificacions del disseny, s'expliquen amb detall el desenvolupament realitzat per cadascuna de les funcions.

6.1 Justificacions del disseny

6.1.1 Captura de dades

El cas d'ús habitual de OpenWSN en una aplicació de captura de dades consisteix en què les motes envien les dades capturades dels sensors periòdicament a un servidor d'emmagatzematge al *cloud*. Però, aquest cas d'ús presenta els següents problemes en el demostrador:

- Les dades s'envien al servidor a través del protocol CoAP amb el tipus de missatges *Non-Confirmable (NON)*, és a dir, sense fiabilitat que les dades han arribat al servidor. Per tant, es poden donar situacions en què les dades no arribin. Per aquest demostrador on les dades s'utilitzen per controlar el sistema de calefacció la pèrdua d'aquestes pot provocar un mal funcionament del demostrador.

- Per emmagatzemar les dades en la plataforma thethings.iO a través del protocol CoAP cal realitzar una petició POST a una URI amb el següent format: *coap://coap.thethings.iO/v2/things/{{THING_TOKEN}}.* On *THING_TOKEN* és un identificador únic obtingut en la web thethings.iO. A més, el *payload* ha d'estar en format JSON i inclou el nom del recurs i diverses dades. El format definit a OpenWSN per enviar les dades al servidor és una petició PUT a través de CoAP en el format: *coap://ip/recurs*, on les dades són text pla. Per tant, els formats són diferents i seria necessari realitzar canvis en la pila OpenWSN.

Aquests dos problemes impliquen les següents decisions de disseny del demostrador:

- A més d'aquests dos problemes, tenint en consideració que la GUI utilitza la llibreria Python CoAP de OpenWSN i que permet peticions CoAP fiables (*timeout* amb reintents sinó es rep ACK) de tipus *Confirmable (CON)*, es decideix realitzar el disseny de la captura de les dades de temperatura i humitat explicat anteriorment. És a dir, enlloc de què les motes enviïn les dades, aquestes són demanades des de l'aplicació Python de la Raspberry Pi. El fet que l'aplicació Python demani les dades implica que ha de conèixer les adreces IPv6 de les motes.
- Per emmagatzemar les dades en la plataforma thethings.iO és necessari utilitzar la seva llibreria Node.js CoAP i imprescindible que cada mota tingui un *thing token*.
- Cal disposar d'una associació entre l'adreça IPv6 i el *thing token* per cada mota. En l'aplicació *openDemo*, les motes s'identifiquen per la IPv6 mentre que en la plataforma thethings.iO per *thing token*.

Com s'ha comentat anteriorment, la llibreria Python CoAP de OpenWSN s'utilitza en l'aplicació *openDemo* i *openDemoGUI* en diverses vegades. La forma d'enviar peticions CoAP a través de la llibreria és creant un objecte *coap* i fer ús de les seves funcions GET, POST, PUT i DELETE. Quan es crea l'objecte *coap* ocupa el recurs socket per una adreça i un port concret, per tant, si es torna a crear un objecte *coap* es produeix un error perquè el recurs ja està ocupat. Per poder compartir l'objecte *coap* en diverses aplicacions s'ha creat la classe *singleton* ObjectCoAP.py, implementada com un mòdul Python, amb diverses funcions. La base de funcionament és definir l'objecte *coap* com una variable global a *"null"*, quan es crida la funció *init* es verifica si l'objecte és *"null"*, en cas afirmatiu es crea l'objecte *coap.* Aquest tipus de disseny normalment implementa un mecanisme de sincronització de fils perquè no es creï l'objecte *coap* dues vegades, però tal i com està dissenyat el demostrador no és necessari.

A més de la funció *init*, s'han implementat tres funcions més. Una permet afegir en la llibreria Python CoAP de OpenWSN el recurs especificat en el paràmetre, aquesta funció s'anomena *addResource*. L'altra, *sendGET*, envia peticions CoAP amb mètode GET passant com a paràmetres l'adreça IPv6 i el recurs el qual es vol accedir. Per últim,

sendPUT envia peticions CoAP amb mètode PUT passant com a paràmetres l'adreça IPv6, el recurs el qual es vol canviar i el nou valor.

En la taula següent es pot veure el codi font de la classe ObjectCoAP.py.

```
#Imports CoAP OpenWSN
from coapdevelop.coap import coap,coapResource
#Variable global
global c
c= None
#Funció per inicialitzar objecte coap
def init():
   global c
   if c is None:
      c=coap.coap()
#Funció per afegir recursos en l'objecte coap
#@param: strResource: string amb el nom del recurs
def addResource(strResource):
   r=coapResource.coapResource(path = strResource)
   c.addResource(r)
#Funció per enviar peticions GET
#@param:ipv6: string amb l'adreça IPv6
#@param: resource: string amb el nom del recurs que es vol accedir
def sendGET(ipv6,resource):
   return c.GET('coap://[{0}]/{1}'.format(ipv6,resource))
#Funció per enviar peticions PUT
#@param:ipv6: string amb l'adreça IPv6
#@param: resource: string amb el nom del recurs que es vol accedir
#@param: order: string amb l'ordre a realitzar
def sendPUT(ipv6,resource, order):
   c.PUT('coap://[{0}]/{1}'.format(ipv6,resource),payload = [ord(order)],)
```

Taula 5: *Codi font classe ObjectCoAP.py*

Per poder compartir l'objecte *coap* entre les aplicacions *openDemo* i *openDemoGUI*, s'han integrat totes dues en el programa openDemo. Cada aplicació s'executa un fil d'execució diferent en la classe principal del programa openDemo, openDemoApp.py. El codi implementat de la classe openDemoApp.py és:

```
#Imports genèrics
import sys,threading,time
#Imports PyQT
from PyQt4 import QtGui
#Imports openDemo
from openDemo import cDataCollection
from openDemoGUI import MainWindow

#Classe multifil Application
class Application(threading.Thread):
   #Constructor
   def __init__(self):
      threading.Thread.__init__(self)
   #Funció que realitza l'execució
   def run (self):
      cDataCollection.start()

#Classe GUI
class ApplicationGUI():
   #Constructor
   def __init__(self):
      # Crear aplicació Qt
      self.app = QtGui.QApplication(sys.argv)
      # Omplir aplicació Qt
      self.gui = MainWindow.MainWindow()
   #Funció que realitza l'execució
   def start(self):
      #Mostrar aplicació Qt
      self.gui.show()
      # Executar aplicació Qt
      self.app.exec_()

#Funció principal
def main():
   #openDemo Application
   application =Application()
   application.start()
   #Esperar per inicialitzar objectes
   time.sleep(5)
   #GUI Application
   applicationGUI = ApplicationGUI()
   applicationGUI.start()
if __name__ == "__main__":
   main()
```

Taula 6: *Codi font classe openDemoApp.py*

6.1.2 Emmagatzematge permanent de dades

El valor de temperatura de consigna i les dades d'adreça IPv6, *thing token*, tipus de mota i la descripció per a cada node de la WSN s'ha d'emmagatzemar de forma permanent. Per això, hi ha dues opcions possibles en el disseny d'estructura de dades permanent: fitxers i bases de dades. En el cas dels fitxers, les dades s'haurien d'emmagatzemar en un format estructurat com XML o JSON. Cada vegada que es necessita una dada cal llegir tot el fitxer, extreure el contingut (*parser*) en funció del format i cercar la dada sol·licitada. En el cas de les bases de dades, les dades estan estructurades i amb una consulta es pot accedir a la informació desitjada. En tots dos casos la informació està estructurada, però en les base de dades és més fàcil la cerca de la informació sol·licitada. Per tant, es selecciona base de dades.

Hi ha diversos gestors de base dades, per exemple, MySQL, SQLite, PostgreSQL, Oracle, etc. Per una banda, MySQL, PostgreSQL i Oracle disposen de gestors de base de dades amb una alta carrega computacional per donar resposta a entorns d'alta concurrència. Per l'altra banda, SQLite funciona molt bé en entorns de baixa concurrència i té un cost computacional baix.

Les funcions del demostrador no requereixen un alt grau d'accés a la base de dades, per tant, el nivell de concurrència és molt baix. A més, la base de dades ha d'estar en un entorn amb certes limitacions computacionals com la Raspberry Pi. En conseqüència, el gestor de base de dades seleccionat és SQLite.

La base de dades està formada per dues taules amb els seus respectius camps:

- **settings:** taula amb el valor de temperatura de consigna. Camps:
    - ***id:*** identificador de tipus *integer*, és la clau primària.
    - ***temp:*** valor de la temperatura de consigna, tipus *integer*.
- **motes:** taula amb les dades d'adreça IPv6, *thing token*, tipus de mota i descripció per a cada node. Inclou els següents camps:
    - ***id:*** identificador de tipus *integer*, és la clau primària.
    - ***ipv6*:** adreça IPv6, tipus *varchar*.
    - ***thingToken*:** thing token, tipus *varchar.*

- ***type*:** tipus de mota, 0: *mota sensor*; 1: *mota actuador*, tipus *integer*.
- ***description:*** descripció de la ubicació de la mota, tipus *varchar*.

Per facilitar l'accés a la base de dades de les aplicacions *openDemo* i *openDemoGUI* des de diferents classes es defineix la classe Database.py perquè cada objecte tingui una connexió a la base de dades diferent.

Tot seguit es mostra el codi font implementat en la classe Database.py:

```
#Imports genèrics
import sqlite3,os
#Imports openDemo
import Log

#Classe Database per accedir a la base de dades
class Database(object):
  #Constructor de la classe
  def __init__(self):
    self.db=None
    self.db_file = '/opt/openDemo/bd/database.sqlite'
  #Funció que verifica si existeix la base de dades, en cas contrari la crea
  def init(self):
    try:
      array=self.db_file.split('/')
      db_path='/'+array[1]+'/'+array[2]+'/'+array[3]+'/'
      pathexists=os.path.exists(db_path)
      if not pathexists:
        os.makedirs(db_path)
      dbexists = os.path.exists(self.db_file)
      if not dbexists:
          self.createTable()
    except:
      Log.writeError("No es pot inicialitzar la base de dades")
  #Funció que connecta amb la base de dades
  def connect(self):
    isConnected=True
    try:
      self.db=sqlite3.connect(self.db_file)
    except:
      isConnected=False
      Log.writeError("No s'ha pogut connectar amb la base de dades")
      pass
    finally:
      return isConnected
```

```
#Funció que realitza una consulta a la base de dades i retorna el resultat
#@param: sql: string amb la sentència de consulta SQL
def select(self,sql):
    records=None
    try:
        cursor=self.db.cursor()
        cursor.execute(sql)
        records = cursor.fetchall()
        cursor.close()
    except:
        Log.writeError("No s'ha realitzat la consulta: "+sql)
        pass
    finally:
        return records

#Funció que realitza un "insert" a la base de dades i retorna un booleà indicant si
      s'ha realitzat correctament
#@param: sql: string amb la sentència d'inserció SQL
def insert(self,sql):
    okInsert=True
    try:
        cursor=self.db.cursor()
        cursor.execute(sql)
        self.db.commit()
    except:
        self.db.rollback()
        okInsert=False
        Log.writeError("No s'ha pogut guardar el registre")
        pass
    finally:
        if cursor is not None:
            cursor.close()
        return okInsert

#Funció que realitza un "update" a la base de dades i retorna un booleà indicant
      si s'ha realitzat correctament
#@param: sql: string amb la sentència d'actualització SQL
def update(self,sql):
    okUpdate=True
    try:
        cursor=self.db.cursor()
        cursor.execute(sql)
        self.db.commit()
    except:
        self.db.rollback()
        okUpdate=False
        Log.writeError("No s'ha pogut actualitzar el registre")
```

```
        finally:
            if cursor is not None:
                cursor.close()
            return okUpdate

    #Funció que tanca la connexió amb la base de dades
    def close(self):
        try:
            if self.db is not None:
                self.db.close()
        except:
            Log.writeError("No s'ha pogut tancar la connexio amb la base de dades")
            pass
        finally:
            self.db=None

    #Funció que crear les dues taules de la base de dades
    def createTable(self):
        try:
            if self.connect():
                sqlt1="""CREATE TABLE "motes" ("id" INTEGER PRIMARY KEY  NOT NULL ,
                        "ipv6" VARCHAR NOT NULL , "thingToken" VARCHAR DEFAULT null,
                        "type" INTEGER NOT NULL DEFAULT (0), "description" VARCHAR
                        DEFAULT null)"""
                cursor = self.db.cursor()
                cursor.execute(sqlt1)
                sqlt2="""CREATE TABLE "settings" ("id" INTEGER PRIMARY KEY  NOT NULL ,
                        "temp" INTEGER NOT NULL )"""
                cursor.execute(sqlt2)
                cursor.execute("INSERT INTO settings (temp) VALUES (19)")
                self.db.commit()
                cursor.close()
        except:
            self.db.rollback()
            Log.writeError("No s'ha pogut crear la taula")
            pass
        finally:
            self.close()

    #Funció que retorna el valor de temperatura de consigna
    def getTemperaturaConsigna(self):
        tempSet=None
        rec=self.select("SELECT temp FROM settings")
        for row in rec:
            tempSet = row[0]
        return tempSet
```

```
#Funció que retorna un booleà indicant si s'ha pogut actualitzar el valor de
      temperatura de consigna
#@param: temp: valor de consigna de temperatura a actualitzar
def updateTemperaturaConsigna (self,temp):
   return self.update("UPDATE settings SET temp= %s"%temp )

#Funció que retorna un diccionari descripció-ipv6 per a cada mota sensor si el
      camp descripció no és null
def getDiccDescriptionIpv6(self):
   dicc=dict()
   rec=self.select("SELECT description,ipv6 FROM motes WHERE type=0 and
                     description IS NOT NULL and description !=''")
   for row in rec:
      dicc[str(row[0])]=str(row[1])
   return dicc

#Funció que retorna un diccionari descripció-Thing_Token per a cada mota
      sensor si els camps description i thingToken no són null
def getDiccDescriptionToken(self):
   dicc=dict()
   rec=self.select("SELECT description,thingToken FROM motes WHERE type=0
                     and description IS NOT NULL and thingToken IS NOT NULL and
                     description !='' and thingToken !=''")
   for row in rec:
      dicc[str(row[0])]=str(row[1])
   return dicc

#Funció que retorna els camps  description,thingToken,type d'una mota
#@param: id: integer amb el id de la mota en la BD
def getSettingsMota (self, idindex):
   tipus=0
   rec=self.select("SELECT description,thingToken,type FROM motes WHERE
                     id=%s"%idindex)
   for row in rec:
      des=str(row[0])
      token=str(row[1])
      tipus=int(row[2])
   if des is None or des=='None':
      des=""
   if token is None or token=='None':
      token=""
   return des,token,tipus
```

```
#Funció que retorna un booleà indicant si la descripció proporcionada existeix en
      la base de dades
#@param: des: string amb la descripció d'una mota
def descripExist(self,des):
   exist=False
   rec=self.select("SELECT id FROM motes WHERE description='%s'"%des)
   if len(rec)>0:
      exist=True
   return exist

#Funció que retorna un booleà indicant si s'ha pogut actualitzar els paràmetres
      de configuració d'una mota
#@param: id: integer amb el id de la mota en la BD
#@param: des: string amb la descripció de la mota
#@param: token: string amb el Thing_Token de la mota
#@param: tipus: integer amb el tipus de mota, 0-mota sensor; 1-mota actuador
def updateSettingsMota(self,idindex, des, token, tipus):
   return self.update("UPDATE motes SET
      description='%s',thingToken='%s',type=%s  WHERE  id=%s  "%(des,  token,
      tipus,idindex) )
```

Taula 7: *Codi font classe Database.py*

Els mètodes de la classe Database.py es poden dividir en dos tipus, els bàsics i els que faciliten el patró de disseny *Model-Vista* de openDemoGUI.

A continuació s'expliquen els mètode bàsics de la classe Database.py.

- ***init ():*** verifica si existeix la base de dades, sinó existeix crida a la funció *createTable.*
- ***connect ():*** crea una connexió a la base de dades.
- ***select (sql):*** realitza una petició a la base de dades amb la sentència SQL passada com a paràmetre.
- ***insert (sql):*** afegeix un registre a la base de dades amb la sentència SQL passada com a paràmetre.
- ***close ():*** tanca la connexió amb la base de dades.
- ***createTable():*** crea les taules *motes* i *settings*, així com afegeix un registre a la taula *settings* amb el valor de temperatura de consigna igual a 19.

Tot seguit es detallen els mètodes que faciliten el patró *Model-Vista:*

- ***getTemperaturaConsigna():*** retorna el valor de temperatura de consigna.
- ***updateTemperaturaConsigna (temp):*** actualitza el valor de temperatura de consigna amb el valor passat en el paràmetre *temp*.
- ***getDiccDescriptionIpv6():*** retorna un diccionari amb clau descripció i valor IPv6 per a cada mota si el camp *description* no és *null*.
- ***getDiccDescriptionToken():*** retorna un diccionari amb clau descripció i valor *thing token* per a cada mota si els camps *description* i *thingToken* no són *null*.
- ***getSettingsMota(idindex):*** retorna els camps *description*, *thingToken* i *type* de la mota amb el *id* corresponent amb el paràmetre *idindex*.
- ***descripExist(des):*** retorna un booleà si la descripció passada en el paràmetre *des* existeix en la base de dades.
- ***updateSettingsMota(idindex,des,token,tipus):*** actualitza els camps *description*, *thingToken* i *type* amb els valors dels paràmetres *des, token* i *tipus* respectivament en la mota amb *id* corresponent al valor de *idindex*.

6.1.3 Comunicació amb la plataforma thethings.iO

La comunicació, emmagatzemar i llegir dades de temperatura i humitat, amb la plataforma thethings.iO es du a terme a través de la llibreria Node.js CoAP de thethings.iO. Per tant, cal desenvolupar una aplicació en el llenguatge Node.js que utilitzi la llibreria de thethings.iO per comunicar-se amb la plataforma thethings.iO. Així mateix, és necessari que les aplicacions *openDemo* i *openDemoGUI* es comuniquin amb l'aplicació Node.js.

El mode funcionament de la llibreria Node.js CoAP de thethings.iO es basa en crear una aplicació Node.js que crea un client theThingsCoAP que permet accedir a les funcions per emmagatzemar i llegir dades de la plataforma thethings.iO. A més, en el mateix directori que l'aplicació Node.js hi ha d'haver el fitxer *config.json* que inclou el *thing token* de la "cosa" al qual es vol accedir.

Aquest mode de funcionament per emmagatzemar i llegir les dades de diferents "things" provoca possibles casos de concurrència. Aquests casos s'han de gestionar

perquè no esdevinguin en un mal funcionament del demostrador. Aquesta gestió es basa en no poder llegir dades fins que no hagi finalitzat el procés d'emmagatzematge i viceversa. Això pot provocar que la resposta de la GUI a una petició de l'usuari s'allargui més de l'espera't, i per tant, impactar negativament en l'experiència d'usuari. Per aquest motiu es creen dues aplicacions Node.js amb els seus respectius fitxers *config.json*, una per emmagatzemar i una altra per llegir les dades anomenades Write.js i Read.js respectivament.

La comunicació entre les aplicacions ha de ser asíncrona i bidireccional. En el cas de l'aplicació Write.js, ha de proporcionar *feedback* per indicar si l'emmagatzematge de les dades a la plataforma thethings.iO ha estat correcte. A més, ha de permetre la configuració d'un *timeout* d'espera del *feedback* de la plataforma thethings.iO.

Hi ha diversos mètodes de IPC (*inter-process communication*) com fitxers, *named pipes*, *sockets*, etc. Analitzant les característiques de la comunicació i els mètodes hi ha dues opcions possibles *named pipes* i sockets. Node.js presenta millor suport per *sockets* que per *named pipes*, per tant, es selecciona els *sockets* com a mètode per a comunicar les aplicacions Python i Node.js.

6.2 Captura de dades de temperatura i humitat

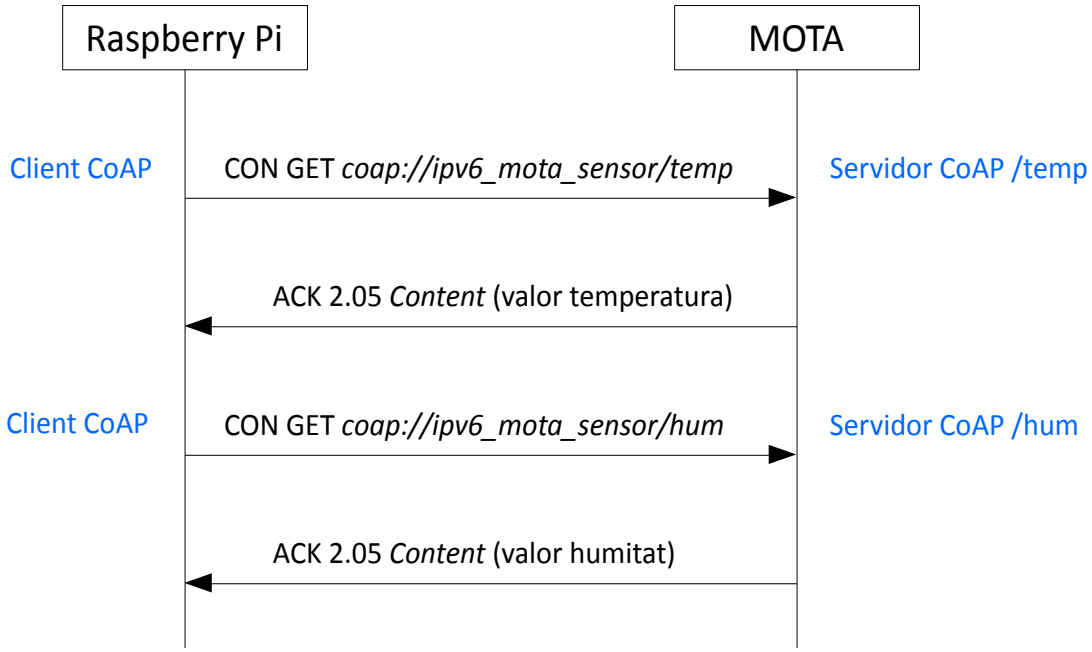
L'aplicació *openDemo* realitza, cada minut, peticions CoAP amb mètode GET a cada *mota sensor* de la WSN per obtenir els valors de temperatura i humitat relativa. Per cada mota, realitza una petició per capturar el valor de temperatura proporcionat pel sensor SHT21 integrat en OpenBattery amb el següent format URI:

coap://ipv6_mota_sensor/temp

I una altra petició pel valor de la humitat adquirit del sensor SHT21 amb el següent format URI:

coap://ipv6_mota_sensor/hum

En la següent il·lustració es pot veure com és la comunicació amb una *mota sensor* per obtenir les dades. L'aplicació *openDemo* funcionant en la Raspberry Pi realitza la funció de client CoAP, mentre que les aplicacions CoAP de la mota són servidors CoAP.



Il·lustració 87: *Esquema del procés de comunicació per capturar les dades*

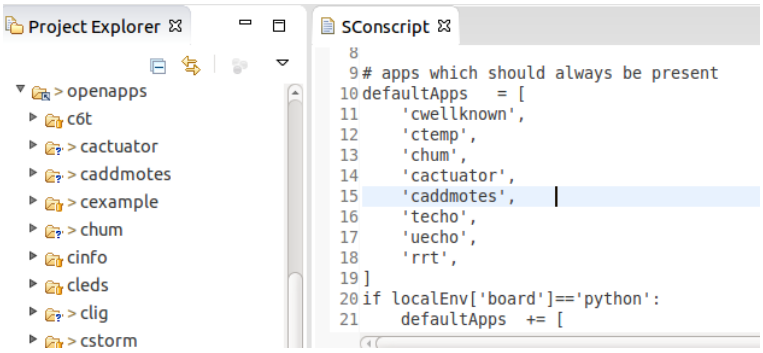
Una vegada obtinguts els valors de temperatura i humitat del sensor SHT21 es calculen els valors reals de temperatura i humitat. Després s'inicia el procés d'emmagatzematge de les dades a thethings.iO en un fil d'execució diferent. Finalitzada la captura de dades de temperatura i humitat s'inicia el procés de control del sistema de calefacció.

En els següents apartats s'explica el desenvolupament realitzat per proporcionar la funció detallada anteriorment en el *firmware* OpenWSN i en openDemo:

6.2.1 *Firmware* OpenWSN

Desenvolupar una aplicació CoAP en el *firmware* de OpenWSN amb capacitat de rebre peticions pel mètode GET en el recurs "*temp*". En rebre una petició s'accedeix al sensor SHT21 a través del bus I2C i es llegeix el valor de temperatura. Aquest valor s'envia com a resposta de la petició. El desenvolupament de l'aplicació CoAP inclou la creació de dues classes ctemp.c i ctemp.h. Per afegir l'aplicació a OpenWSN cal afegir

el nom de la classe ctemp al llistat d'aplicacions de la variable *defaultApps* del fitxer SConscript de la carpeta openapps, tal i com es mostra en la següent il·lustració.



Il·lustració 88: *Configuració del fitxer SConscript del firmware de OpenWSN*

L'estructura de la classe ctemp.c és:

- Importar les llibreries necessàries:

```
#include "opendefs.h"
#include "ctemp.h"
#include "opencoap.h"
#include "openqueue.h"
#include "packetfunctions.h"
#include "sht21.h"
#include "idmanager.h"
```

Taula 8: *Imports de la classe ctemp.c del firmware de OpenWSN*

- Definició de variables i mètodes:

```
const uint8_t ctemp_path0[] = "temp";

ctemp_vars_t ctemp_vars;

owerror_t ctemp_receive(OpenQueueEntry_t* msg, coap_header_iht*
coap_header, coap_option_iht* coap_options);

void ctemp_sendDone(OpenQueueEntry_t* msg, owerror_t error);
```

Taula 9: *Definició de variables i mètodes de la classe ctemp.c del firmware de OpenWSN*

- Funció *init* que es crida quan s'inicia OpenWSN. Inicialitza les variables de l'aplicació CoAP i configura dues funcions *callback*, una quan es rep un paquet CoAP i l'altra quan s'ha enviat un paquet CoAP. Si la mota realitza la funció de *DAGroot* no s'inicialitza l'aplicació.

```
void ctemp_init(void) {
        //No inicialitzar si és DAGroot
        if (idmanager_getIsDAGroot() == TRUE)
                return;
        // register to OpenCoAP module
        ctemp_vars.desc.path0len = sizeof(ctemp_path0) - 1;
        ctemp_vars.desc.path0val = (uint8_t*) (&ctemp_path0);
        ctemp_vars.desc.path1len = 0;
        ctemp_vars.desc.path1val = NULL;
        ctemp_vars.desc.componentID = COMPONENT_CTEMP;
        ctemp_vars.desc.callbackRx = &ctemp_receive;
        ctemp_vars.desc.callbackSendDone = &ctemp_sendDone;
        opencoap_register(&ctemp_vars.desc);
}
```

Taula 10: *Funció init de la classe ctemp.c del firmware de OpenWSN*

- En rebre una petició CoAP s'executa la funció *ctemp_receive.* Si la petició és del mètode GET es llegeix el valor de temperatura del sensor SHT21 i s'envia en format *big endian* com a resposta a la petició.

```
owerror_t ctemp_receive(OpenQueueEntry_t* msg, coap_header_iht*
coap_header, coap_option_iht* coap_options) {
        owerror_t outcome;
        switch (coap_header->Code) {
        case COAP_CODE_REQ_GET:
                // reset packet payload
                msg->payload = &(msg->packet[127]);
                msg->length = 0;
                // add CoAP payload
                packetfunctions_reserveHeaderSize(msg, 3);
                msg->payload[0] = COAP_PAYLOAD_MARKER;
                uint16_t temp = 0x0;
```

```
                //Verificar que el sensor SHT21 és present
                if (sht21_is_present() == true) {
                        //Inicialitza comunicació amb sensor SHT21 a través del
                        bus I2C
                        sht21_init();
                        //Llegir valor temperatura del sensor SHT21
                        temp = sht21_read_temperature();
                }

                // return as big endian
                msg->payload[1] = (uint8_t)(temp >> 8) ;
                msg->payload[2] = (uint8_t) (temp & 0xff);

                // set the CoAP header
                coap_header->Code = COAP_CODE_RESP_CONTENT;
                outcome = E_SUCCESS;
                break;
        default:
                outcome = E_FAIL;
                break;
        }
        return outcome;
}
```

Taula 11: *Funció receive de la classe ctemp.c del firmware de OpenWSN*

- Per finalitzar, la funció que es crida quan s'ha enviat un paquet CoAP és:

```
void ctemp_sendDone(OpenQueueEntry_t* msg, owerror_t error) {
        openqueue_freePacketBuffer(msg);
}
```

Taula 12: *Funció sendDone de la classe ctemp.c del firmware de OpenWSN*

La classe ctemp.h conté les definicions de les capçaleres de ctemp.c.

```
#ifndef __CTEMP_H
#define __CTEMP_H
#include "opencoap.h"
typedef struct {
  coap_resource_desc_t desc;
} ctemp_vars_t;
void ctemp_init(void);
#endif
```

Taula 13: *Codi font de la classe ctemp.h del firmware de OpenWSN*

Desenvolupar una aplicació CoAP en el *firmware* de OpenWSN amb capacitat de rebre peticions pels mètodes GET en el recurs "*hum*". En rebre una petició s'accedeix al sensor SHT21 a través del bus I2C i es llegeix el valor d'humitat relativa. Aquest valor s'envia com a resposta de la petició. Aquesta aplicació té la mateixa estructura que les classes ctemp.c i ctemp.h explicades anteriorment, per tant, només es detallen les diferències més significatives per l'aplicació:

- El nom del recurs és "*hum*".

```
const uint8_t chum_path0[] = "hum";
```

Taula 14: *Configuració nom recurs "hum" de la classe chum.c del firmware de OpenWSN*

- En rebre una petició GET es llegeix el valor de la humitat i s'envia en format *big endian*.

```
uint16_t humSend = 0x0;
//Verificar que el sensor SHT21 és present
if (sht21_is_present() == true) {
        //Inicialitza comunicació amb sensor SHT21 a través del bus I2C
        sht21_init();
        //Llegir valor humitat del sensor SHT21
        humSend = sht21_read_humidity();
        }
        // return as big endian
        msg->payload[1] = (uint8_t) (humSend >> 8);
        msg->payload[2] = (uint8_t) (humSend & 0xff);
```

Taula 15: *Llegir valor d'humitat en la classe chum.c del firmware de OpenWSN*

6.2.2 Aplicació openDemo

En l'aplicació openDemo es crea la seva classe principal, cDataCollection.py, implementada amb el patró de disseny *singleton,* desenvolupada en un mòdul Python. cDataCollection.py conté les funcions *start* i *collectData* (realitza la captura de les dades). Tot seguit s'expliquen aquestes dues funcions.

- En la funció *start* s'inicialitzen diverses variables i es defineix l'estructura per cridar la funció *collectData* de forma periòdica. Algunes de les variables inicialitzades s'utilitzen per realitzar alguna de les altres funcions importants i s'expliquen en la seva funció corresponent en els següents apartats.
    - S'inicialitza la classe Log.py, implementada amb el patró de disseny *singleton* (desenvolupada en un mòdul Python), a través de la funció *init*. La classe Log.py té l'objectiu de proporcionar un fitxer de registres d'errors pel desenvolupament realitzat que no proporciona *feedback* a l'usuari. El fitxer s'anomena *openDemo.log* i està ubicat en **/opt/openDemo /log/.** La classe Log.py disposa de dues funcions, una per inicialitzar i una altra per escriure missatges en el fitxer. El desenvolupament d'aquesta classe és:

```
#Imports genèrics
import logging,os
#Variable global
global logger

#Funció per inicialitzar el log
def init():
   global logger
   logger = logging.getLogger('openDemo')
   logger.setLevel(logging.ERROR)
   path='/opt/openDemo/log/'
   if not os.path.exists(path):
      os.makedirs(path)
   handler = logging.FileHandler(path+'openDemo.log')
   handler.setLevel(logging.ERROR)
   formatter = logging.Formatter('%(asctime)s - %(name)s -
                                  %(levelname)s - %(message)s')
   handler.setFormatter(formatter)
   logger.addHandler(handler)
```

```
#Funció per escriure en el log
def writeError(strError):
   global logger
   logger.error(strError)
```

Taula 16: *Codi font classe Log.py*

Tot seguit es detallen les funcions de cada mètode:

- ***init():*** verifica que existeix la ruta **/opt/openDemo /log/**, en cas contrari crea els directoris necessaris. Inicialitza l'objecte *logger* i configura el format dels registres d'error.
- ***writeError(strError):*** escriu en el fitxer de registres d'error el missatge del paràmetre *strError*.

En la següent il·lustració es pot observar un exemple del contingut del *log*:

```
2015-05-23 11:26:57,267 - openDemo - ERROR - sendUdp socketUdpReal
2015-05-23 11:26:57,423 - openDemo - ERROR - No hi ha comunicacio amb la mota bbbb::12:4b00:3a5:8d52
2015-05-23 11:27:34,313 - openDemo - ERROR - No s'ha pogut enviar ordre stopAddMotes
```

Il·lustració 89: *Exemple del format del fitxer d'errors*

- Es crea l'objecte *coap* i es defineix a 60 segons el temps per realitzar les peticions de forma periòdica. Es crea un objecte Database i es verifica que la base de dades existeix, en cas contrari es crea.
- Es crea un bucle amb l'estructura *while True* que inclou la crida a la funció *collectData* i parar l'execució del codi durant un període de temps a través de la funció *time.sleep.* Aquest temps es calcula restant el temps en realitzar la captura de les dades al temps definit entre captures. Si el temps per realitzar la captura de les dades és superior al temps definit entre captures, es crida a la funció *collectData* sense parar l'execució.

```
#Funció que inicialitza diversos components i crida a la funció collectData() cada minut
def start():
    global lock
    Log.init()
    ObjectCoAP.init()
    ObjectCoAP.addResource('caddmotes') #Alta adreces IPv6
    lock = threading.Lock() #Emmagatzematge a thethings.iO
    timeCollection=60
    db=Database()
    db.init()
    while True:
        timeStart=time.time()
        collectData(db)
        tempsActual= time.time()
        diff= tempsActual-timeStart
        if (diff<timeCollection):
            temps=timeCollection-(diff%60)
            time.sleep(temps)
```

Taula 17: *Funció start de la classe cDataCollection.py*

- En la funció *collectData* es realitzen algunes tasques de les altres funcions destacades, aquestes s'expliquen en la seva funció corresponent en els següents apartats. Les accions de *collectData* per la present funció són:
    - Connexió a la base de dades per obtenir les adreces IPv6 i els *thing token* de les motes tipus *sensor* que tinguin configurat el *thing token*.
    - Per a cada mota, primer es realitza una petició CoAP amb mètode GET per obtenir el valor de temperatura i es calcula el seu valor real amb dos decimals a través del mètode *getTemperature* de la classe Convert.py.

```
def getTemperature(value):
    temp = (value[0] << 8)+value[1]
    temperatura =-46.86+175.72*temp/ 65536
    return round(temperatura,2)
```

Taula 18: *Funció getTemperature de la classe Convert.py*

Després s'envia una sol·licitud CoAP amb mètode GET per obtenir el valor d'humitat i es calcula el seu valor real amb dos decimals a través del mètode *getHumidity* de la classe Convert.py.

```
def getHumidity(value):
    hum = (value[0] << 8)+value[1]
    humidity = -6.0+125.0 * hum / 65536
    return round(humidity,2)
```

Taula 19: *Funció getHumidity de la classe Convert.py*

Una vegada obtinguts les dades de temperatura i humitat s'inicia el procés d'emmagatzematge a thethings.iO en un fil d'execució diferent per mitja de les següents línies de codi:

```
send=SendDataToServer(lock,tempValue,humValue,ipv6,str(row[1]))
#Emmagatzematge a thethings.iO
 send.start()#Emmagatzematge a thethings.iO
```

Taula 20: *Codi font de l'inci del procés d'emmagatzematge a thethings.iO*

- Finalitzada la captura de dades de temperatura i humitat de totes les motes s'inicia el procés de control del sistema de calefacció a través de la sentència *Control.checkAction(lowTemp).*

El codi font de la funció *collectData* és:

```
#Funció que captura les dades de cada mota realitzant peticions CoAP GET,
#envia les dades a l'aplicació Write.js i controla l'estat de la calefacció de la
#caldera
def collectData(db):
    global lock
    if db.connect():
        try:
            Control.getSettings (db) #Control sistema calefaccio
            records=db.select("SELECT ipv6,thingToken FROM motes where
                                type=0 and thingToken IS NOT NULL and
                                thingToken !=''")
            lowTemp=False #Control sistema calefaccio
```

```
        for row in records:
          try:
            ipv6=str(row[0])
            temp =ObjectCoAP.sendGET(ipv6,'temp' )
            if temp is not None:
              tempValue = ConvertValue.getTemperature(temp)
              if not lowTemp:  #Control sistema calefacció
                lowTemp = Control.checkTemp(tempValue) #Control
                      #sistema calefacció
            hum =ObjectCoAP.sendGET(ipv6,'hum' )
            if hum is not None:
              humValue=ConvertValue.getHumidity(hum)
            if (tempValue or humValue) is not None:

        send=SendDataToServer(lock,tempValue,humValue,ipv6,str(row[1]))
        #Emmagatzematge a thethings.iO
              send.start() #Emmagatzematge a thethings.iO
          except:
            Log.writeError( "No hi ha comunicacio amb la mota "+ ipv6)
            pass
        Control.checkAction(lowTemp) #Control sistema calefacció
      except:
        pass
    db.close()
```

Taula 21: *Funció collectData de la classe cDataCollection.py*

La captura de les dades de temperatura i humitat s'ha de realitzar cada minut. Si s'analitza l'estructura dissenyada dins del bucle *while*, es pot veure que si el temps per realitzar les captures de dades és superior a un minut això no es compleix. Si el demostrador funciona correctament aquesta situació no es produirà mai.

El temps de captura de dades depèn d'aquests dos factors:

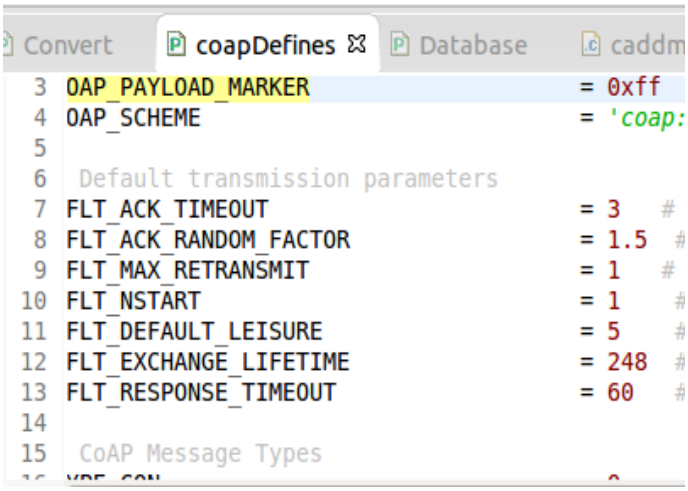
- Comunicació amb les motes: les peticions CoAP són de tipus *Confirmable*, per tant, tenen un *timeout* per rebre l'ACK i un nombre de reintents. La llibreria Python CoAP de OpenWSN defineix un *timeout* de 20 segons i el nombre màxim de reintents a 5, en realitat, es realitzen 7 tal i com està implementada la llibreria.
- Comunicació amb la plataforma thethings.iO: les peticions CoAP tenen un *timeout* de 5 segons per rebre resposta i el nombre d'intents és 2.

Un mal funcionament del demostrador pot ser provocat per dos motius que es poden produir de forma independent o concurrent, aquests són:

- No hi ha resposta de les motes: pot ser que per algun motiu una o més motes no contestin, per exemple, s'ha acabat la bateria. Com s'ha comentat anteriorment, es realitza una petició per la temperatura i una altra per la humitat. Sinó es rep l'ACK de la petició per la temperatura passat el *timeout* de tots els reintents no es realitza la petició per la humitat.
- No hi ha comunicació amb la plataforma thethings.iO: degut a què ha caigut el servidor o la connexió a Internet de la llar. La comunicació amb la plataforma thethings.iO es realitza enviant el valor de temperatura i humitat per separat per a cada mota.

Es planteja un escenari en què la WSN del demostrador està format per 8 motes de tipus *sensor* i aquestes dues suposicions per estimar el temps de captura de dades:

- No hi ha comunicació amb la plataforma thethings.iO, per cada mota el temps és 2 peticions x 2 reintents x 5 segons (*timeout*), és a dir, 20 segons. Per tota la WSN, 8 motes x 20 segons = 160 segons. En conseqüència, un únic punt de fallida provocaria no assolir amb el compliment de capturar dades cada minut. Per aquest motiu l'enviament de les dades a la plataforma es considera com una tasca de llarga duració i s'executa en un fil d'execució diferent que el del bucle *while*.
- No hi ha comunicació amb una mota, el temps és 7 reintents x 20 segons (*timeout*) = 140 segons. Per tant, no s'assoliria amb el compliment de capturar dades cada minut. Per això es modifica els valors de la llibreria Python CoAP de OpenWSN definint un *timeout* de 3 segons i el nombre màxim d'intents a 3, valors més que suficient en funcionament normal. La modificació dels valors es realitza en la classe coapDefines.py de la llibreria, configurant el *timeout* en la variable FLT_ACK_TIMEOUT i el nombre de reintents en la variable FLT_MAX_RETRANSMIT amb valor 1. S'assigna el valor 1 perquè es realitzin 3 intents, això és degut a com està implementada la llibreria. Sinó hi ha comunicació amb una mota el missatge d'error que s'obté és: *coapTimeout(reason=No ACK received after 3 tries (max 2))*.



Il·lustració 90: *Configuració dels valors de transmissió de la llibreria Python CoAP de OpenWSN*

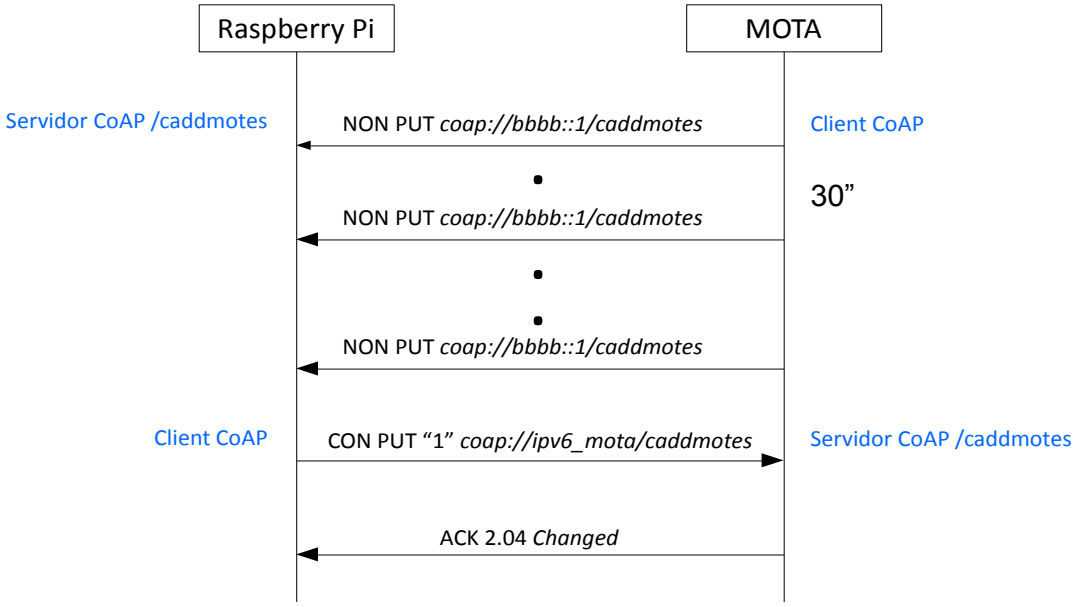
Amb aquests configuracions, per no assolir el compliment de capturar dades cada minut, no hauria d'haver comunicació amb més de 6 motes de la WSN, fet amb una probabilitat que succeeixi molt baixa.

6.2.3 Alta d'adreces IPv6 de les motes

Les adreces IPv6 de les motes s'emmagatzemen a la base de dades del demostrador de forma automatitzada. El principi de funcionament es basa en què la mota envia peticions CoAP amb mètode PUT al recurs "*caddmotes*" de la llibreria Python CoAP de OpenWSN cada 30 segons. En rebre la petició es processa i si l'adreça IPv6 no està en la base de dades s'emmagatzema. Posteriorment, s'envia una petició CoAP amb mètode PUT a la mota en el recurs "*caddmotes*" perquè deixi d'enviar peticions.

Les peticions CoAP amb mètode PUT de la mota són missatges CoAP tipus *Non-Confirmable (NON)*. S'utilitza aquest tipus de missatges perquè no és un problema greu que la petició no arribi, ja que s'envia sempre la mateixa informació de forma repetida.

En la següent il·lustració es pot veure un resum de l'explicació de funcionament anterior i com els papers de servidor i de client de CoAP s'intercanvien.



Il·lustració 91: *Esquema del procés de comunicació per donar d'alta a les motes*

En els següents apartats s'explica el desenvolupament realitzat per proporcionar la funció detallada anteriorment en el *firmware* OpenWSN i en openDemo:

6.2.3.1 *Firmware* OpenWSN

Desenvolupar una aplicació CoAP en el *firmware* de OpenWSN amb la funcionalitat d'enviar de forma periòdica cada 30 segons una petició CoAP amb mètode PUT a la llibreria Python CoAP de OpenWSN, la seva adreça IPv6 és bbbb::1. A més, ha de poder rebre peticions CoAP amb mètode PUT per deixar d'enviar peticions. Aquesta aplicació és semblant a les desenvolupades anteriorment, les diferències són:

- El nom del recurs és "*caddmotes*".

```
const uint8_t caddmotes_path0[] = "caddmotes";
```

Taula 22: *Configuració nom recurs "caddmotes" de la classe caddmotes.c del firmware de OpenWSN*

- Afegir en la funció *init* la configuració del *timer* i una nova funció que es crida quan passa el temps definit en la variable CCADDMOTESPERIOD.

```
#define CADDMOTESPERIOD  30000 ///ms
void caddmotes_init() {
…….
caddmotes_vars.timerId = opentimers_start(CCADDMOTESPERIOD,
TIMER_PERIODIC, TIME_MS, caddmotes_timer_cb);
}
void caddmotes_timer_cb(opentimer_id_t id) {
        scheduler_push_task(caddmotes_task_cb, TASKPRIO_COAP);
}
```

Taula 23: *Definició variable i funcions init i timer_cb de la classe caddmotes.c del firmware de OpenWSN*

La nova funció *caddmotes_timer_cb* afegeix al planificador de tasques una nova funció anomenada *caddmotes_task_cb*.

caddmotes_task_cb s'encarrega de generar el paquet CoAP i enviar-lo a l'adreça IP bbbb::1, definida en el *firmware* de OpenWSN en la variable *ipAddr_ringmaster*. La generació i enviament del paquet no es realitza fins que la mota estigui sincronitzada amb la xarxa. El procés de generació del paquet consisteix en crear el paquet amb la funció *openqueue_getFreePacketBuffer* i anar afegint la informació necessària. Aquesta informació conté les dades de propietari del paquet, marca d'inici de les dades (*payload*), les opcions del *content-type* i *location-path*, així com l'adreça i el port destí del paquet.

```
void caddmotes_task_cb() {
        OpenQueueEntry_t* pkt;
        owerror_t outcome;
        // don't run if not synch
        if (ieee154e_isSynch() == FALSE)
                return;

        // create a CoAP RD packet
        pkt = openqueue_getFreePacketBuffer(COMPONENT_CADDMOTES);
        if (pkt == NULL) {
                openserial_printError(COMPONENT_CADDMOTES,
                                      ERR_NO_FREE_PACKET_BUFFER,
                                (errorparameter_t) 0, (errorparameter_t) 0);
                openqueue_freePacketBuffer(pkt);
                return;
        }
```

```
        // take ownership over that packet
        pkt->creator = COMPONENT_CADDMOTES;
        pkt->owner = COMPONENT_CADDMOTES;
        packetfunctions_reserveHeaderSize(pkt, 1);
        pkt->payload[0] = COAP_PAYLOAD_MARKER;

        // content-type option
        packetfunctions_reserveHeaderSize(pkt, 2);
        pkt->payload[0] = (COAP_OPTION_NUM_CONTENTFORMAT -
                           COAP_OPTION_NUM_URIPATH) << 4 | 1;
        pkt->payload[1] = COAP_MEDTYPE_APPOCTETSTREAM;

        // location-path option
        packetfunctions_reserveHeaderSize(pkt, sizeof(caddmotes_path0) - 1);
        memcpy(&pkt->payload[0], caddmotes_path0,
                        sizeof(caddmotes_path0) -    1);
        packetfunctions_reserveHeaderSize(pkt, 1);
        pkt->payload[0] = ((COAP_OPTION_NUM_URIPATH) << 4)
                           | (sizeof(caddmotes_path0) - 1);

        // metadata
        pkt->l4_destination_port = WKP_UDP_COAP;
        pkt->l3_destinationAdd.type = ADDR_128B;
        memcpy(&pkt->l3_destinationAdd.addr_128b[0], &ipAddr_ringmaster,
                        16);

        // send
        outcome = opencoap_send(pkt, COAP_TYPE_NON,
                           COAP_CODE_REQ_PUT, 1,&caddmotes_vars.desc);

        // avoid overflowing the queue if fails
        if (outcome == E_FAIL) {
                openqueue_freePacketBuffer(pkt);
        }
        return;
}
```

Taula 24: *Funció init i task_cb de la classe caddmotes.c del firmware de OpenWSN*

- Afegir la capacitat de rebre peticions CoAP amb mètode PUT amb l'ordre de control "1" amb la finalitat de parar el *timer* i així deixar d'enviar peticions cada 30 segons. Per realitzar aquestes accions cal afegir a l'aplicació CoAP de la mota el següent codi en el mètode *receive*:

```
        case COAP_CODE_REQ_PUT:
                // Stop timer
                if (msg->payload[0] == '1') {
                        opentimers_stop(ccaddmotes_vars.timerId);
                }
                // reset packet payload
                msg->payload = &(msg->packet[127]);
                msg->length = 0;
                // set the CoAP header
                coap_header->Code = COAP_CODE_RESP_CHANGED;
                outcome = E_SUCCESS;
                break;
```

Taula 25: *Codi per rebre peticions PUT en la funció receive de la classe caddmotes.c del firmware de OpenWSN*

6.2.3.2 Aplicació openDemo

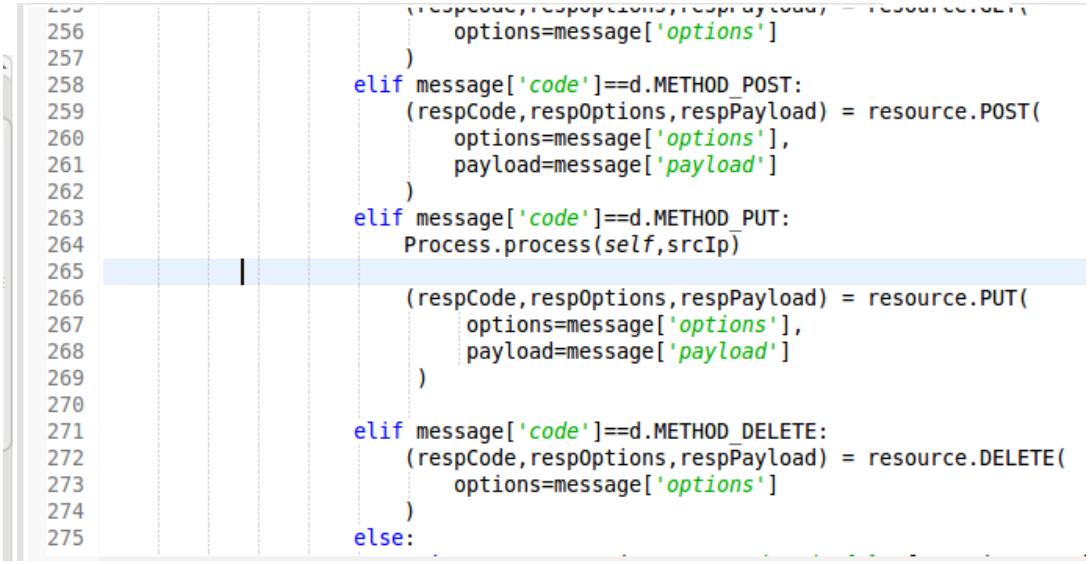
Les accions implementades en openDemo són:

- Afegir el recurs "*caddmotes*" a la llibreria Python CoAP de OpenWSN en la classe cDataCollection.py de la següent forma:

```
ObjectCoAP.addResource('caddmotes') #Alta adreces IPv6
```

Taula 26: *Afegir recurs "caddmotes" a la llibreria Python CoAP de OpenWSN*

- La petició arriba a la llibreria Python CoAP de OpenWSN que no la processa. Per processar-la es crea una classe *singleton* anomenada Process, desenvolupada en un mòdul Python, i es crida en la classe coap.py a través del mètode *process* passant com a paràmetres l'objecte *coap* i l'adreça IPv6 origen de la petició realitzada per la mota.



```
256                     options=message['options']
257                 )
258             elif message['code']==d.METHOD_POST:
259                 (respCode,respOptions,respPayload) = resource.POST(
260                     options=message['options'],
261                     payload=message['payload']
262                 )
263             elif message['code']==d.METHOD_PUT:
264                 Process.process(self,srcIp)
265 
266                 (respCode,respOptions,respPayload) = resource.PUT(
267                     options=message['options'],
268                     payload=message['payload']
269                 )
270 
271             elif message['code']==d.METHOD_DELETE:
272                 (respCode,respOptions,respPayload) = resource.DELETE(
273                     options=message['options']
274                 )
275             else:
```

Il·lustració 92: *Crida al mètode process de la classe Process.py des de la llibreria Python CoAP de OpenWSN*

- S'utilitza el patró *singleton* enlloc de *multifil* perquè la concurrència és baixa i no es realitza cap tasca de duració elevada. Les accions de la funció *process* són
    - Verificar si l'adreça IPv6 de la mota està en la base de dades.
    - Sinó està, es valida que el format de l'adreça IPv6 és correcte i que comença per "bbbb::". La validació de què l'adreça IPv6 comenci per "bbbb::" es realitza perquè OpenVisualizer crea un interfície *tun0* amb l'adreça bbbb::1, per tant, totes les adreces IPv6 de les motes comencen per "bbbb::". La verificació de l'adreça es realitza:

```
#Funció que verifica el format IPv6
#@param: ipv6: string amb l'adreça IPv6 de la mota
def checkIPv6Format(ipv6):
   formatOK=True
   try:
      socket.inet_pton(socket.AF_INET6, ipv6)
   except:
      Log.writeError( "format IPv6: "+ipv6)
      formatOK=False
      pass
   return formatOK
```

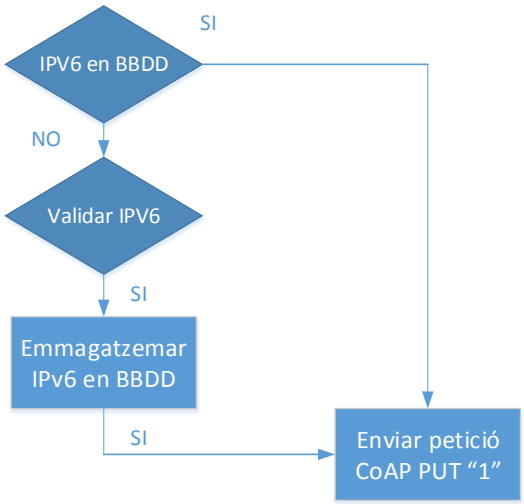
Taula 27: *Funció checkIPv6Format de la classe Process.py*

- Si la verificació és correcte s'emmagatzema l'adreça IP.
- Si l'adreça IPv6 s'ha pogut emmagatzemar en la base de dades o ja hi és, s'envia una petició CoAP PUT amb l'ordre "1" a la mota perquè deixi d'enviar peticions cada 30 segons. Aquesta petició s'envia en un altre fil d'execució per evitar problemes de concurrència en la llibreria Python CoAP de OpenWSN i perquè pot ser una tasca de duració gran (3 reintents amb *timeout* de tres segons).

```
#Funció que envia per CoAP una comanda PUT perquè la mota deixi
#d'enviar paquets
#@param: c: objecte coap
#@param: ipv6: string amb l'adreça IPv6 de la mota
def sendStopAddMote(c,ipv6):
    try:
        c.PUT('coap://[{0}]/caddmotes'.format(ipv6),payload = [ord('1')],)
    except:
        Log.writeError("No s'ha pogut enviar ordre stopAddMotes")
        pass
```

Taula 28: *Funció sendStopAddMotes de la classe Process.py*

En la següent il·lustració s'observa el diagrama de flux del funcionament del mètode *process*.



Il·lustració 93: *Diagrama de flux del funcionament del mètode process*

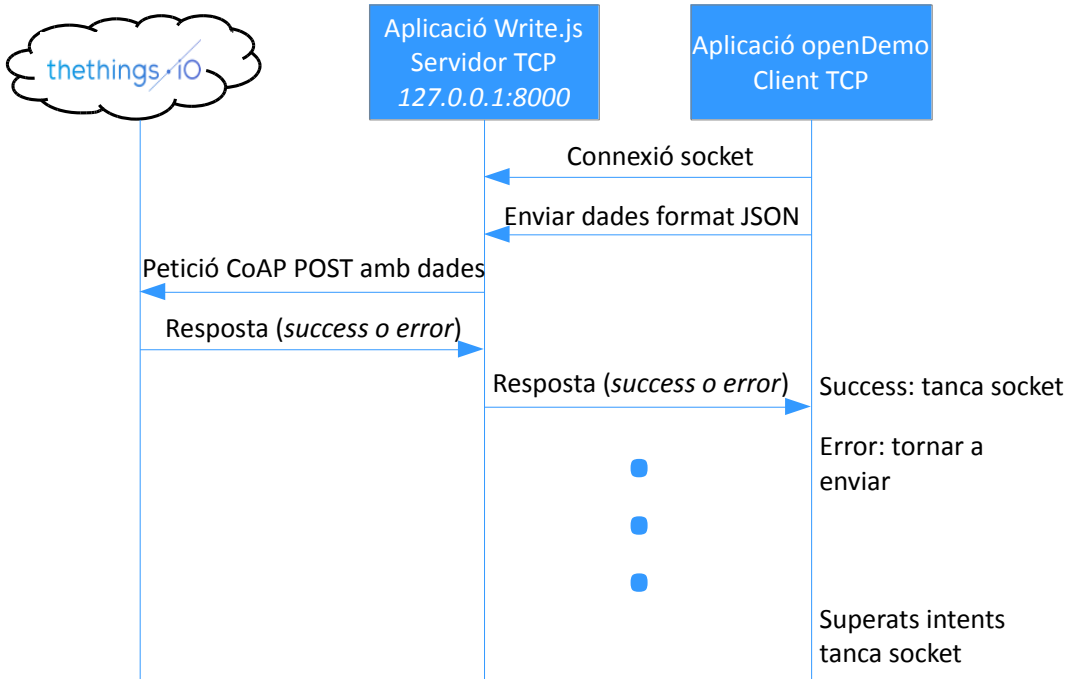
En la taula de sota es pot veure el codi font implementat en la funció *process*:

```
#Funció que processa el paquet rebut, si IPv6 no esta en el sistema s'afegeix
#Si s'afegeix o ja esta en el sistema s'envia ordre a la mota perque deixi
#d'enviar
#@param: c: objecte coap
#@param: ipv6: string amb l'adreça IPv6 de la mota
def process(c,ipv6):
   if ipv6 is not None:
      isOk=False
      db=Database()
      if db.connect():
         records=db.select("SELECT ipv6 FROM motes where ipv6='%s'" %
                  ipv6)
         if not records:
            if checkIPv6Format(ipv6) and checkIPOpenWSN(ipv6):
               if db.insert("INSERT INTO motes (ipv6) VALUES ('"+ipv6+"')"):
                  isOk=True
         else:
            isOk=True
         db.close()
      if isOk:
         thread.start_new_thread( sendStopAddMote,(c,ipv6,))
```

Taula 29: *Funció process de la classe Proces.py*

6.3 Emmagatzemar dades a thethings.iO

En l'aplicació Write.js s'implementa un servidor TCP per escoltar peticions. En l'aplicació openDemo es desenvolupen clients TCP que es connecten al servidor i envien les dades de temperatura i humitat en format JSON. Les dades s'envien a la plataforma thethings.iO a través d'una petició CoAP POST i la plataforma respon amb un estat que pot ser *success* o *error*. Aquesta resposta s'envia al client TCP, si és *success* el client TCP tanca la connexió i si és *error* es repeteix l'enviament de dades al servidor TCP tantes vegades com intents configurats. Superat el nombre d'intents es tanca la connexió i es notifica l'error. En la següent il·lustració es pot veure l'esquema de la comunicació entre les aplicacions openDemo, Write.js i la plataforma *cloud* thethings.iO.



Il·lustració 94: *Esquema del procés de comunicació entre openDemo, Write.js i thethings.iO*

En els següents apartats s'explica el desenvolupament realitzat per dur a terme aquest procés en les aplicacions Write.js i openDemo:

6.3.1 Aplicació Write.js

L'aplicació Write.js implementa per una banda un servidor TCP en l'adreça *localhost* (127.0.0.1) i el port 8000 per rebre peticions de clients TCP amb les dades de temperatura i humitat en format JSON. Per l'altra banda un client de la llibreria Node.js CoAP de thethings.iO per enviar peticions CoAP POST amb les dades a la plataforma thethings.iO pel seu emmagatzematge, així com la capacitat de capturar la resposta de la plataforma thethings.iO i enviar-la al client TCP. El codi font desenvolupat és:

```
//Capturar possibles errors
process.on('uncaughtException', function(err) {
    console.log(err);
});
//Importar mòdul net
var net = require('net')
//Número de port
var port = 8000;
```

```
//Crear servidor TCP
net.createServer(function(socket){
   socket.on('data', function(data){
      //Parse dades JSON
      var json=JSON.parse(data);
      //Importar mòdul theThingsCoAP
      var theThingsCoAP = require('../../');
      //Crear client theThingsCoAP
      var client = theThingsCoAP.createClient();
      //Format dades a enviar
      var object = {
            "values":
                [
              {
                    "key": json.key,
                    "value": json.value,
                    "units": json.units,
                     "type": "temporal"
              }
            ]
      }
      //Enviar les dades a la plataforma thethings.iO
      client.thingWrite(object, function (error, data) {
            if (typeof data!=='undefined' && data!==null){
                  //Resposta al client amb l'estat de la petició (success o error)
                  socket.write(data.status);
        }
      });

      });
//Configuració del port en el servidor TCP
}).listen(port);
```

Taula 30: *Codi font de l'aplicació Write.js*

6.3.2 Aplicació openDemo

Una vegada obtinguts els valors de temperatura i humitat relativa d'una mota en l'aplicació openDemo s'envien a l'aplicació Write.js en un fil d'execució diferent perquè pot ser una execució de llarga durada (reintents i *timeout*). L'enviament de les dades en un fil d'execució diferent es realitza a través de la classe SendDataToServer.py que es crida des de la classe cDataCollection.py.

```
send=SendDataToServer(lock,tempValue,humValue,str(row[0]),str(row[1]))
send.start()
```

Taula 31: *Codi font de l'inci del procés d'emmagatzematge a thethings.iO*

SendDataToServer.py té els següents atributs: *lock* per sincronitzar els fils, valor de temperatura, valor d'humitat, adreça IPv6 i el *thing token* associat.

Defineix una variable global amb el número d'intents de comunicació amb la plataforma thethings.iO si es produeixen errors.

```
#Import genèric
import threading
#Imports openDemo
import Log
from Configthethingsio import Configthethingsio
#Definició de la classe Thread SendDataToServer
class SendDataToServer(threading.Thread):
    #Variable global
    retries=2
    #Constructor de la classe
    #@param: lock: objecte Lock per sincronitzar fils
    #@param: tempValue: string amb el valor de temperatura
    #@param: hum: string amb el valor d'humitat
    #@param: ipv6: string amb l'adreça IPv6 de la mota
    #@param: thingtoken: string amb el Thong_Token associat a la mota
    def __init__(self, lock,tempValue,hum,ipv6,thingtoken):
        threading.Thread.__init__(self)
        self.lock = lock
        self.tempValue=tempValue
        self.hum=hum
        self.ipv6 = ipv6
        self.thingtoken=thingtoken
```

Taula 32: *Definició i constructor de la classe SendDataToServer.py*

L'execució de la classe SendDataToServer.py consisteix en:

- Adquirir el *lock* perquè només hi hagi un fil d'execució accedint als recursos de *socket* i fitxer de configuració *config.json* de la llibreria Node.js CoAP de thethings.iO.

- Crear un objecte Configthethingsio passant com a paràmetre el número de port, s'utilitza el port 8000. En la següent taula es pot veure les variables globals de la classe Configthethingsio.py, la seva definició i el seu constructor.

```
#Imports genèrics
import json, socket
#Import openDemo
import Log

#Definicio de la classe que gestiona les accions amb thethings.iO
class Configthethingsio:
   #Variables globals
   localhost='127.0.0.1'
   BUFSIZ=1024
   #Constructor de la classe
   def __init__(self,port):
      self.port = port
      self.s = None
```

Taula 33: *Definició i constructor de la classe Configthethingsio.py*

- Crear un client TCP, configurar el *timeout* a 5 segons i connectar amb el servidor TCP implementat en l'aplicació Write.js a través de l'adreça *localhost* i el port 8000. En la taula de sota es pot observar el codi font desenvolupat per realitzar les accions comentades anteriorment.

```
#Funció per connectar amb l'aplicació de la llibreria thethings.io a traves de
#socket, timeout=5 segons
   def connectLibraryTheThingsio(self):
      isConnected=True
      try:
         self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
         self.s.settimeout(5)
         self.s.connect((self.localhost, self.port))
      except:
         Log.writeError("No s'ha pogut connectar amb thethings.iO")
         isConnected=False
         pass
      finally:
         return isConnected
```

Taula 34: *Funció connectLibraryTheThingsio de la classe Configthethingsio.py*

- Modificar el fitxer *config.json* amb el *thing token* corresponent a la mota que es vol emmagatzemar les dades a la plataforma thethings.iO. En la taulasegüent es pot visualitzar el codi per modificar el fitxer.

```
#Funció per modificar el valor del Thing_Token en el fitxer config.json
#@param: path: "write" o "read"
#@param:  thingToken: string amb el Thing_Token
def modify(self,path,thingToken):
    isModify=True
    try:
        with open("/usr/local/thethingsio-coap-
         openDemo/openDemo/"+path+"/config.json", "r+") as jsonFile:
            data = json.load(jsonFile)
            data["thingToken"]=thingToken
            jsonFile.seek(0)
            jsonFile.write(json.dumps(data))
            jsonFile.truncate()
            jsonFile.close()
    except:
        isModify=False
        Log.writeError("En modificar el fitxer config.json")
        pass
    finally:
        return isModify
```

Taula 35: *Funció modify de la classe Configthethingsio.py*

- Enviar el valor de temperatura i humitat en format JSON i processar la resposta de la plataforma thethings.iO. En la taula de sota es pot veure el codi font desenvolupat per enviar les dades a través del *socket* i com es processa la resposta per si ha hagut qualsevol error.

```
#Funció per enviar dades a la llibreria thethings.iO i verifica que no hi ha error
#@param: data: dades a enviar a la plataforma thethings.iO en format JSON
def sendToTheThingsio(self,data):
    isSent=True
    try:
        if self.s is not None:
            self.s.send(data)
            data=self.s.recv(self.BUFSIZ)
            if (data=='error'):
                isSent=False
```

```
        except:
            isSent=False
            Log.writeError( "En la comunicacio amb thethings.iO")
            pass
        finally:
            return isSent
```

Taula 36: *Funció sendToTheThingsio de la classe Configthethingsio.py*

- Si hi ha errors en l'emmagatzematge de les dades en la plataforma thethings.iO es torna a enviar les dades al servidor TCP en funció del valor d'intents definit.
- Si l'error persisteix, finalitzat el nombre d'intents es notifica l'error.
- Tancar la connexió TCP amb el servidor. En la taula següent es mostra el codi font implementat per tancar la connexió.

```
    #Funció per tancar socket amb amb l'aplicació de la llibreria thethings.io
    def closeConnectionLibraryTheThingsio(self):
        try:
            if self.s is not None:
                self.s.close()
        except:
            Log.writeError( "No s'ha pogut tancar la comunicacio amb thethings.iO")
            pass
```

Taula 37: *Funció closeConnectionLibraryTheThingsio de la classe Configthethingsio.py*

- Alliberar el *lock* pel següent fil.

El codi font per realitzar l'execució de SendDataToServer és:

```
    #Funció que es crida en realitzar start() a l'objecte SendDataToServer
    #Connecta amb l'aplicació Write que inclou la llibreria de thethings.iO per
enviar            les dades a la plataforma thethings.iO
    def run (self):
        self.lock.acquire()
        thethingsio=Configthethingsio(8000)
```

```
        if thethingsio.connectLibraryTheThingsio():
            if thethingsio.modify("write",self.thingtoken):
                if self.tempValue is not None:
                    i=0
                    isSaved=False
                    while(i<self.retries):
                        if
thethingsio.sendToTheThingsio('{"key":"temperature","value":"'+str(self.tempValue
                                                    )+'","units":"Temperature"}'):
                            isSaved=True
                            break
                        i=i+1
                    if not isSaved:
                        Log.writeError( "No s'ha pogut emmagatzemar la temperatura de la
                                        mota: "+self.ipv6)
                if self.hum is not None:
                    i=0
                    isSaved=False
                    while(i<self.retries):
                        if
thethingsio.sendToTheThingsio('{"key":"humidity","value":"'+str(self.hum)+'","units
                                                    ":"Humidity"}'):
                            isSaved=True
                            break
                        i=i+1
                    if not isSaved:
                        Log.writeError( " no s'ha pogut emmagatzemar l'humitat de la mota:
                                        "+self.ipv6)
            thethingsio.closeConnectionLibraryTheThingsio()
        self.lock.release()
```

Taula 38: *Funció run de la classe SendDataToServer.py*

6.4 Control del sistema de calefacció

La funció de control del sistema de calefacció es basa en gestionar el funcionament del sistema de calefacció de la caldera amb les dades de temperatura mesurades per la WSN de forma periòdica i en l'estat de la calefacció (encesa o apagada).

A partir d'aquestes informacions, es decideix si s'ha d'actuar sobre la caldera i quina acció cal realitzar. Si és necessari actuar, s'envia una ordre d'actuació a la *mota*

actuador perquè realitzi l'acció pertinent sobre un relé a través d'un GPIO. Aquest relé controla la calefacció. Per tant, l'estat de la calefacció és el mateix del GPIO de la mota.

Es defineixen els estats de funcionament de la calefacció de la següent manera:

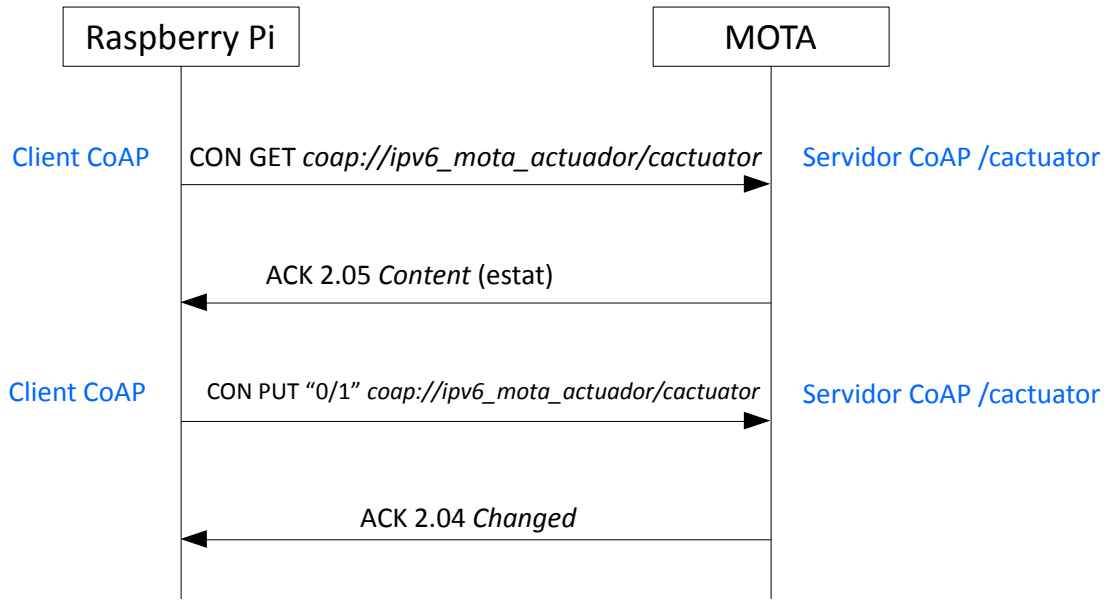
- Si la calefacció està funcionant el seu estat és "1".
- Si la calefacció està pagada el seu estat és "0".

Per conèixer l'estat de la calefacció es realitza una petició CoAP amb el mètode GET en el recurs "*cactuator*" a la *mota actuador*. La URI de la petició és:

coap://ipv6_mota_actuador/cactuator

Per encendre la calefacció s'envia una petició CoAP amb el mètode PUT en el recurs "*cactuator*" a la *mota actuador* amb un "1" en el camp de dades (*payload*). Per apagar la calefacció s'envia la mateixa petició que per encendre, però amb un "0" en el camp de dades. En tots dos casos, la URI de la sol·licitud és la mateixa que la mostrada per la petició amb mètode GET.

En la il·lustració de sota es pot visualitzar un esquema que sintetitza la comunicació amb la mota *actuador* per realitzar la funció de control del sistema de calefacció.



Il·lustració 95: *Esquema del procés de comunicació del control de la calefacció*

En els següents apartats s'explica el desenvolupament realitzat per proporcionar la funció detallada anteriorment en el *firmware* OpenWSN i en openDemo:

6.4.1 *Firmware* OpenWSN

Desenvolupar una aplicació CoAP en el *firmware* de OpenWSN amb capacitat de rebre peticions pels mètodes GET i PUT. Les peticions amb mètode GET proporcionen l'estat del GPIO ("1" o "0"). Les peticions amb mètode PUT activen o desactiven el GPIO en funció de la informació rebuda. Per seleccionar el GPIO a utilitzar s'ha analitzat l'esquemàtic de OpenMote-CC2538 (Esquemàtic placa OpenMote-CC2538) i OpenBase (Esquemàtic placa OpenBase), així com els GPIOs lliures en el codi font de OpenWSN. El resultat del anàlisis és utilitzar el GPIO PB5 del microcontrolador que correspon amb la sortida DO8 de OpenBase. Aquesta aplicació és semblant a les desenvolupades anteriorment, les diferències són:

- Afegir import de la classe gpio.c.

```
#include "gpio.h"
```

Taula 39: *Importar la classe gpio.c en la classe cactuator.c del firmware de OpenWSN*

- Afegir les definicions per utilitzar el GPIO PB5.

```
#define BSP_GPIO_BASE        GPIO_B_BASE
#define BSP_GPIO_PB5         GPIO_PIN_5
```

Taula 40: *Definicions pel GPIO PB5 en la classe cactuator.c del firmware de OpenWSN*

- El nom del recurs és "*cactuator*".

```
const uint8_t cactuator_path0[] = "cactuator";
```

Taula 41: *Configuració nom recurs "cactuator" de la classe cactuator.c del firmware de OpenWSN*

- Afegir funció per inicialitzar el GPIO PB5.

```
//Funció per inicialitzar GPIO
void gpio_pb5_init() {
        GPIOPinTypeGPIOOutput(BSP_GPIO_BASE, BSP_GPIO_PB5);
        GPIOPinWrite(BSP_GPIO_BASE, BSP_GPIO_PB5, 0);
}
```

Taula 42: *Funció gpio_pb5_init de la classe cactuator.c del firmware de OpenWSN*

- Afegir funció per verificar l'estat del GPIO PB5.

```
//Funció que obté l'estat del GPIO
uint8_t gpio_pb5_isOn() {
        uint32_t ui32Toggle = GPIOPinRead(BSP_GPIO_BASE,
        BSP_GPIO_PB5);
        return (uint8_t) (ui32Toggle & BSP_GPIO_PB5) >> 5;
}
```

Taula 43: *Funció gpio_pb5_isOn de la classe cactuator.c del firmware de OpenWSN*

- Afegir funció per activar el GPIO PB5.

```
//Funció per activar el GPIO
void gpio_pb5_on() {
        GPIOPinWrite(BSP_GPIO_BASE, BSP_GPIO_PB5, BSP_GPIO_PB5);
}
```

Taula 44: *Funció gpio_pb5_on de la classe cactuator.c del firmware de OpenWSN*

- Afegir funció per desactivar el GPIO PB5.

```
//Funció per desactivar el GPIO
void gpio_pb5_off() {
        GPIOPinWrite(BSP_GPIO_BASE, BSP_GPIO_PB5, 0);
}
```

Taula 45: *Funció gpio_pb5_off de la classe cactuator.c del firmware de OpenWSN*

- Inicialitzar el GPIO PB5 en la funció *init* de l'aplicació CoAP.

```
void cactuator_init(void) {
        .......
        .......
        //Inicialitzar PB5
        gpio_pb5_init();
}
```

Taula 46: *Inicialitzar GPIO PB5 en la funció init de la classe cactuator.c del firmware de OpenWSN*

- Respondre a les peticions amb mètode GET amb l'estat del GPIO ("1" o "0") en la funció *receive*.

```
case COAP_CODE_REQ_GET:
        // reset packet payload
        msg->payload                = &(msg->packet[127]);
        msg->length                 = 0;
        // add CoAP payload
        packetfunctions_reserveHeaderSize(msg,2);
        msg->payload[0]             = COAP_PAYLOAD_MARKER;

        //Obtenir estat GPIO
        if (gpio_pb5_isOn()==1) {
          msg->payload[1]           = '1';
        } else {
          msg->payload[1]           = '0';
        }
        // set the CoAP header
        coap_header->Code           = COAP_CODE_RESP_CONTENT;
        outcome                     = E_SUCCESS;
        break;
```

Taula 47: *Codi per rebre peticions GET i respondre amb l'estat del GPIO PB5 en la funció receive de la classe cactuator.c del firmware de OpenWSN*

- Actuar sobre el GPIO a partir de les ordres rebudes en les peticions amb mètode PUT en la funció *receive*.

```
case COAP_CODE_REQ_PUT:
                    // Canviar estat GPIO
                    if (msg->payload[0] == '0') {
                            gpio_pb5_off();
                    } else if (msg->payload[0] == '1') {
                            gpio_pb5_on();
                    }
                    // reset packet payload
                    msg->payload = &(msg->packet[127]);
                    msg->length = 0;
                    // set the CoAP header
                    coap_header->Code = COAP_CODE_RESP_CHANGED;
                    outcome = E_SUCCESS;
                    break;
```

Taula 48: *Codi per rebre peticions PUT i actuar sobre el GPIO PB5 en la funció receive de la classe cactuator.c del firmware de OpenWSN*

6.4.2 Aplicació openDemo

En l'aplicació openDemo s'ha desenvolupat la classe Control.py que inclou diversos mètodes per realitzar les tasques de funcionament del control del sistema de calefacció descrites anteriorment. Control.py és una classe amb patró de disseny *singleton*, implementada com un mòdul Python, i les seves tasques són:

- Cada vegada que s'inicia el procés de captura de dades, s'accedeix a la base de dades per obtenir el valor de la temperatura de consigna a través del mètode *getSettings* cridat en la classe principal cDataCollection.py.

```
#Funció per obtenir el valor de temperatura de la base de dades
#@param: db: objecte Database
def getSettings (db):
   global tempSet,database
   database = db
   tempSet=db.getTemperaturaConsigna()
```

Taula 49: *Funció getSettings de la classe Control.py*

- Per cada valor de temperatura obtingut, es verifica si aquest és inferior al valor de la temperatura de consigna a través del mètode *checkTemp*. Si és inferior es modifica el valor de la variable *lowTemp* a "True". Aquesta acció es realitza en la classe principal cDataCollection.py.

```
#Funció que retorna un booleà per indicar si la temperatura capturada està per
#sota de la de consigna
#@param: temp: valor real de la temperatura capturat
def checkTemp(temp):
    global tempSet
    lTemp=False
    if temp < tempSet:
        lTemp=True
    return lTemp
```

Taula 50: *Funció checkTemp de la classe Control.py*

- Una vegada finalitzat el procés de captura de dades, es verifica si s'ha de realitzar alguna acció en el sistema de calefacció i quina a través del mètode *checkAction* cridat en la classe principal cDataCollection.py. El procés de verificació és:
    - Obtenir l'adreça IPV6 de la *mota actuador* de la base de dades a través del mètode *getIPActuator.*

```
#Funció per obtenir IPv6 de la mota que actua sobre la caldera
def getIPActuator ():
    global database
    ip = None
    rec=database.select("SELECT ipv6 FROM motes where type=1")
    for row in rec:
        ip = row[0]
    return ip
```

Taula 51: *Funció getIPActuator de la classe Control.py*

    - Obtenir l'estat de la calefacció per mitjà del mètode *getState* realitzant una petició CoAP amb el mètode GET en el recurs "*cactuator*" a la *mota actuador*.

```
#Funció per obtenir l'estat actual de l'actuador fent un CoAP GET al
#recurs 'cactuator'
#@param: ipActuator: string amb l'adreça IPv6 de la mota actuador
def getState(ipActuator):
    state=None
    try:
        if ipActuator is not None:
            p = ObjectCoAP.sendGET(ipActuator,'cactuator')
            state = chr(p[0])
    except:
        Log.writeError("No es pot obtenir l'estat")
        pass
    return state
```

Taula 52: *Funció getState de la classe Control.py*

- Si la variable *lowTemp* és "True" i l'estat de la calefacció és "0" s'envia petició per encendre la calefacció.
- Si la variable *lowTemp* és "False" i l'estat de la calefacció és "1" s'envia petició per apagar la calefacció.

El codi per enviar ordres de control és:

```
#Funció que envia ordre a l'actuador a través CoAP PUT en el recurs
#'cactuator'
#@param: ipActuator: string amb l'adreça IPv6 de la mota actuador
#@param: order: string amb l'ordre a realitzar per la mota
def sendOrder(ipActuator,order):
    try:
        ObjectCoAP.sendPUT(ipActuator,'cactuator',order)
    except:
        Log.writeError( "No s'ha pogut enviar l'ordre")
        pass
```

Taula 53: *Funció sendOrder de la classe Control.py*

El codi font que implementa la verificació és:

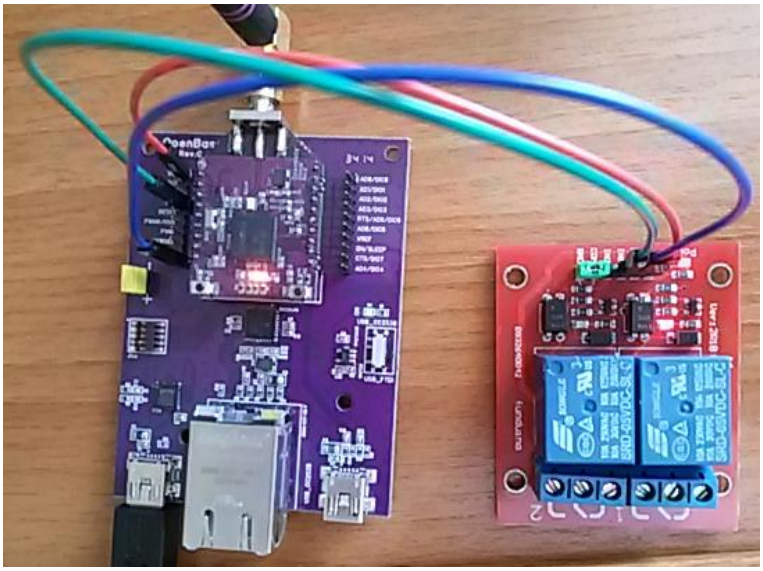
```
#Funció que verifica si s'ha d'enviar ordre de control a la mota que controla
#la caldera
#@param: lowTemp: booleà que indica si cap valor de temperatura està per
#sota del valor de consigna
def checkAction(lowTemp):
    ipActuator = getIPActuator ()
    if ipActuator is not None:
        state=getState(ipActuator)
        if state is not None:
            if (lowTemp and state == '0'):
                sendOrder(ipActuator,'1')
            elif (lowTemp == False and state=='1'):
                sendOrder(ipActuator,'0')
```

Taula 54: *Funció checkAction de la classe Control.py*

El relé utilitzat és un mòdul de dos canals de la marca Keyes Funduino. El mòdul està format per dos relés i diversos pins de connexió: VCC, GND i una entrada d'activació per a cada relé. En la següent il·lustració es pot veure el mòdul:

Il·lustració 96: *Mòdul relés dos canals*

La connexió entre les plaques OpenBase i el mòdul de relés es realitza connectant amb cables els pius de la placa OpenBase VCC,GND i DO8 amb els pius del mòdul VCC, GND i IN1 respectivament. En la següent il·lustració es pot veure com es realitza la connexió i el LED IN1 del mòdul encès simulant l'activació del sistema de calefacció.



Il·lustració 97: *Connexió entre OpenBase i el mòdul de relés*

6.5 Visualització i configuració de dades amb la GUI.

Les funcions que permet la interfície gràfica d'usuari (GUI) són:

- Configurar el valor de temperatura de consigna.
- Configurar paràmetres de cada mota de la WSN.
- Generar un gràfic amb l'històric de dades de temperatura i humitat.
- Mostrar el valor de temperatura i humitat de forma instantània de qualsevol node de la WSN.

Per proporcionar aquestes funcions en la interfície gràfica d'usuari (GUI) s'han desenvolupat dues aplicacions, la principal, openDemoGUI implementada amb el *framework* Qt en el llenguatge de programació Python. L'altra és Read.js que permet llegir dades de la plataforma thethings.iO.

6.5.1 openDemoGUI

openDemoGUI és una aplicació d'interfície gràfica d'usuari (GUI) formada per diverses finestres per proporcionar les funcions detallades anteriorment. El disseny de les diverses finestres s'ha realitat a partir d'un esborrany (fitxer amb extensió .ui) creat amb l'eina Qt Creator i transformat a un fitxer Python amb l'eina pyuic4 de la següent forma:

```
pyuic4 finestra.ui –o finestra.py
```

Tots els botons de la GUI s'identifiquen amb icones enlloc de text. Aquestes icones s'afegeixen a les propietats del botons com a recursos. Per generar els recursos es crea un fitxer amb extensió ".qrc" amb el següent format:

```
<RCC>
  <qresource prefix="/icons">
    <file alias='Up.png'>icons/Up.png</file>
    <file alias='edit.png'>icons/edit.png</file>
    <file alias='save.png'>icons/save.png</file>
    <file alias='settings.png'>icons/settings.png</file>
    <file alias='temp.png'>icons/temp.png</file>
    <file alias='update.png'>icons/update.png</file>
    <file alias='back.jpeg'>icons/back.jpeg</file>
    <file alias='down.jpeg'>icons/down.jpeg</file>
    <file alias='line_chart_icon.jpg'>icons/line_chart_icon.jpg</file>
  </qresource>
</RCC>
```

Il·lustració 98: *Fitxer qrc amb els recursos*

I s'executa la següent comanda per convertir el fitxer amb extensió ".qrc" a un fitxer Python:

```
pyrcc4 –o icon_rc.py icon.qrc
```

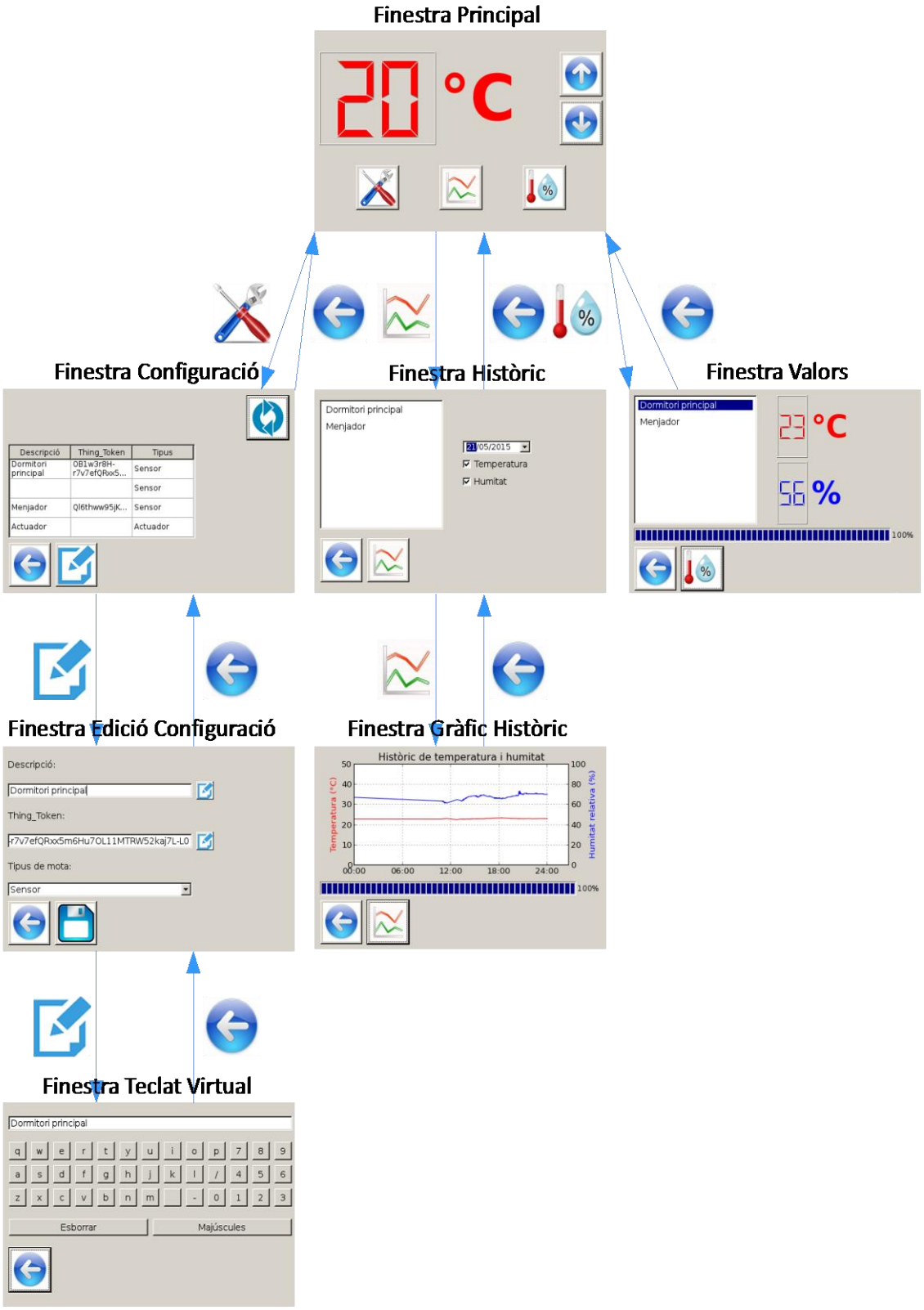
Per poder afegir el recurs en el botó només cal realitzar el import del fitxer icon_rc.py i referenciar el recurs, per exemple l'icona back.jpeg com *":/icons/back.jpeg"*.

Les finestres de openDemoGUI són:

- **Finestra Principal:** formada per un display per mostrar el valor de temperatura de consigna, per dos botons per pujar i baixar el valor de temperatura i tres botons per accedir a la resta de finestres de openDemoGUI. El primer, Configuració, per obrir la Finestra Configuració, el segon, Històric, per accedir a la Finestra Històric i el tercer, Valors, per obrir la Finestra Valors.

- **Finestra Configuració:** formada per una taula amb els paràmetres de configuració de cada mota de la WSN, el botó Actualitzar que permet actualitzar el contingut de la taula, el botó Tornar per tornar a la Finestra Principal i el botó Edició que permet accedir a la Finestra Edició Configuració.
- **Finestra Històric:** consta d'una llista de motes identificades per la seva descripció, un selector de dates, dos *checkboxes* per seleccionar les dades a mostrar, el botó Tornar per tornar a la Finestra Principal i el botó Gràfic que permet accedir a la Finestra Gràfic Històric.
- **Finestra Valors:** formada per una llista de motes identificades per la seva descripció, un display per mostrar el valor de temperatura i un altre pel valor d'humitat relativa, una barra de progrés, el botó Tornar que permet tornar a la Finestra Principal i el botó Valors per realitzar la petició de captura de dades a la mota seleccionada.
- **Finestra Edició Configuració:** consta de dos editors de text pels paràmetres descripció i *Thing_Token* amb els seus respectius botons per accedir a la Finestra Teclat Virtual, així com, una llista desplegable per seleccionar el tipus de mota, el botó Tornar per tornar a la Finestra Configuració i el botó Guardar per emmagatzemar els canvis realitzats.
- **Finestra Gràfic Històric:** formada per un gràfic, una barra de progrés, el botó Tornar per tornar a la Finestra Històric i el botó Gràfic que permet representar els valors en el gràfic.
- **Finestra Teclat Virtual:** consta d'un editor de text, d'una matriu de botons amb les lletres del teclat i botó Tornar que permet tornar a la Finestra Edició Configuració.

En la següent il·lustració es poden veure totes les finestres explicades anteriorment i el seu cicle d'ús.



Il·lustració 99: *Finestres de openDemoGUI i el seu cicle d'ús*

Les finestres Configuració, Històric i Valors s'han desenvolupat amb el patró de disseny *Model-Vista*. S'accedeix a la base de dades per obtenir les dades i es converteixen en

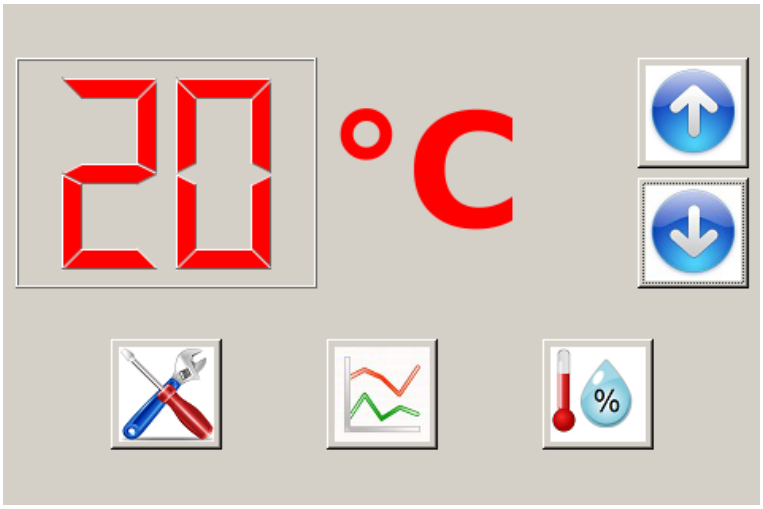
Model (*QStandardItemModel*) per incorporar-lo al component *QListView* o *QTableView* segons el cas.

En els casos d'ús es dóna la situació de la necessitat de passar dades des de una finestra a una altra ja creada, per donar resposta a això, s'han creat SIGNALS i les dades es transmeten amb l'emissió del SIGNAL corresponent.

Les finestres Valors i Gràfic Històric requereixen obtenir dades de processos que poden tenir una duració elevada, aquest fet pot impactar negativament en l'experiència d'usuari. Per minimitzar aquest impacte negatiu, s'ha incorporat una barra de progrés que va augmentant a mesura que avança el procés per obtenir les dades a través de l'emissió de SIGNAL i l'execució dels processos en una classe QThread.

6.5.2 Configurar valor de temperatura de consigna

Quan s'inicia la Finestra Principal, accedeix a la base de dades per obtenir el valor de temperatura de consigna i el mostra en el display, sinó pot accedir a la base de dades mostra un missatge d'error.



Il·lustració 100: *Finestra Principal*

El valor de temperatura de consigna es pot modificar a través del botons Pujar i Baixar:

Incrementa en un grau centígrad el valor de la temperatura de consigna.

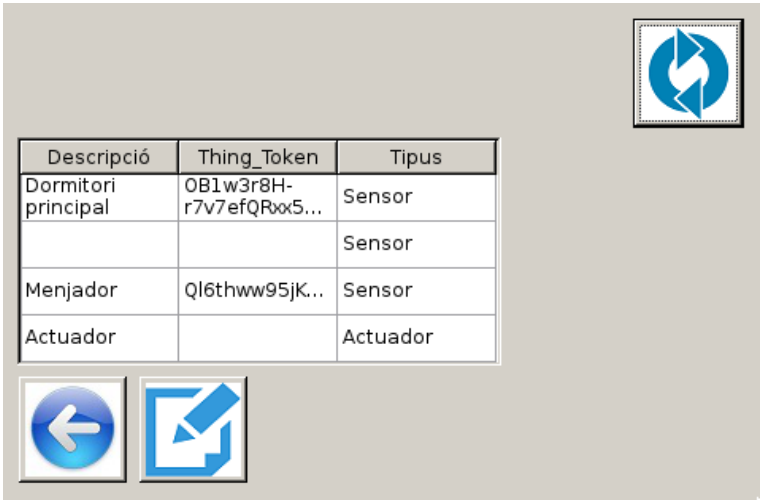
 Disminueix en un grau centígrad el valor de la temperatura de consigna.

En prémer qualsevol del dos botons es modifica el valor de la temperatura de consigna i s'actualitza aquest valor en la base de dades. Si s'ha pogut actualitzar es mostra la modificació en la GUI, en cas contrari, no es modifica la GUI i es mostra una finestra d'error.

6.5.3 Configurar paràmetres de les motes

Per configurar els paràmetres de cada mota de la WSN cal accedir a la Finestra Configuració fent clic en el botó Configuració de la Finestra Principal.

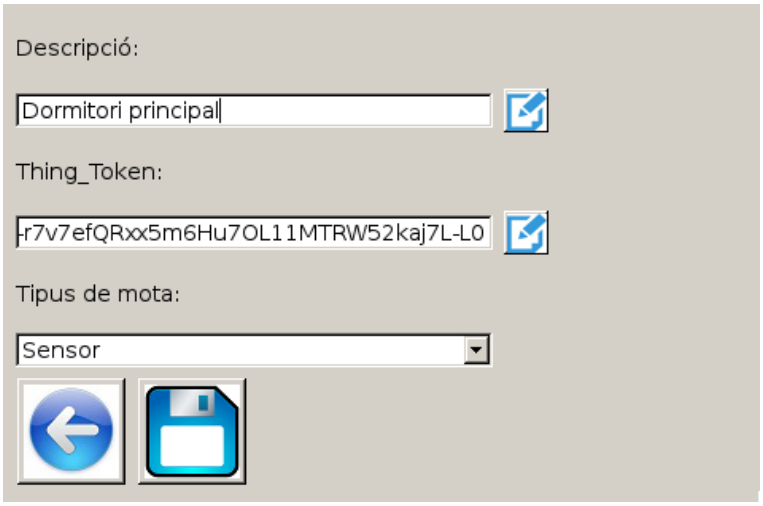
La Finestra Configuració permet visualitzar una taula (*QTableView*) que llista per a cada mota de la base de dades els paràmetres de configuració: descripció, *Thing_Token* i tipus de mota. El contingut de la taula obtingut de la base de dades es transforma en *Model* a través de *QStandardItemModel* i s'afegeix al component *QTableView*.



Il·lustració 101: *Finestra Configuració*

El botó Actualitzar permet actualitzar el contingut de la taula (actualitza el *Model*) per veure si una mota nova s'ha donat d'alta en la base de dades sense sortir i tornar a entrar en la Finestra Configuració.

Per configurar els paràmetres d'una mota, es selecciona la mota a modificar de la taula i es fa clic en el botó Edició. En prémer el botó s'accedeix a la Finestra Edició Configuració.



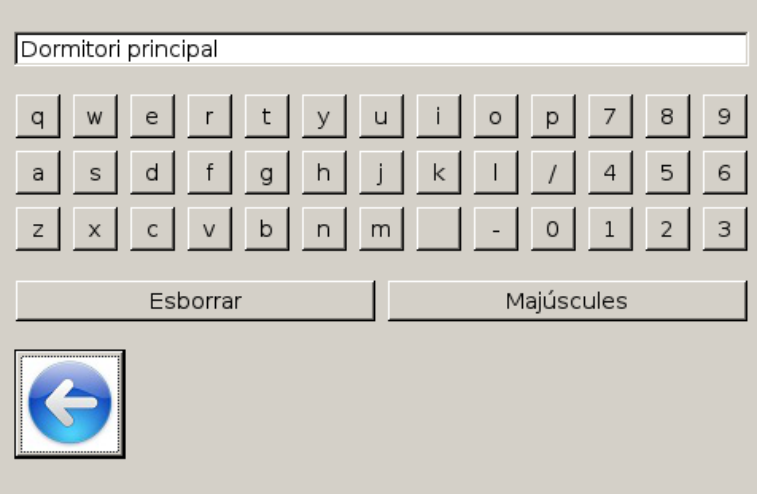
Il·lustració 102: *Finestra Edició Configuració*

La Finestra Edició Configuració càrrega de forma automàtica les dades dels paràmetres de configuració de la mota en els camps d'edició Descripció i *Thing_Token*, així com en la llista desplegable pel tipus de mota.

Per modificar el paràmetre descripció cal prémer el botó Edició situat al final del camp edició Descripció. En fer clic en el botó s'obre la Finestra Teclat Virtual amb el valor del paràmetre descripció en el camp d'edició. Es realitza la modificació desitjada i es fa clic en el botó Tornar que emet un SIGNAL amb el nou valor del paràmetre descripció a la Finestra Edició Configuració i tanca la Finestra Teclat Virtual. La Finestra Edició Configuració rep el SIGNAL i actualitza el valor del paràmetre descripció en el camp edició Descripció de forma automàtica.

La modificació del paràmetre *Thing_Token* es realitza de la mateixa manera que per la descripció explicada anteriorment.

Per canviar el tipus de mota només cal seleccionar l'opció adequada de la llista desplegable: *Sensor* o *Actuador*.



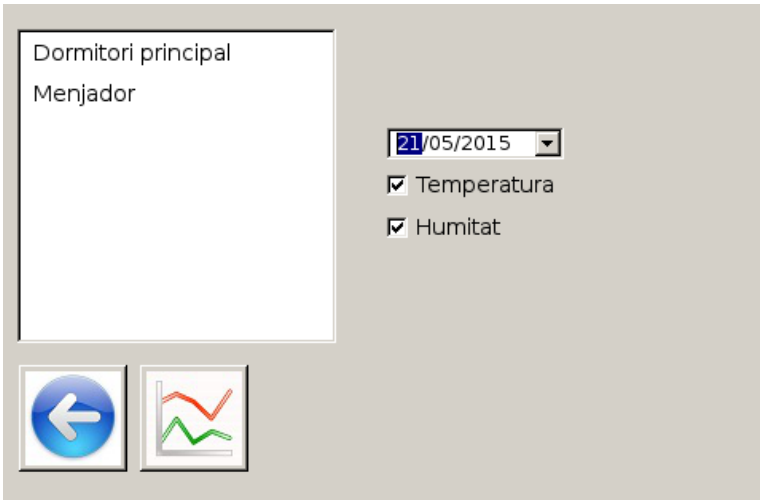
Il·lustració 103: *Finestra Teclat Virtual*

Una vegada finalitzades totes les modificacions desitjades es fa clic en el botó Guardar per emmagatzemar els canvis en la base de dades. Abans d'emmagatzemar les modificacions es verifica que el paràmetre descripció és únic a la base de dades, ja que s'utilitza com a identificador únic en altres funcions de openDemoGUI. Si la descripció ja existeix, no s'actualitzen els canvis en la base de dades i es mostra una finestra d'error amb el missatge: *"La descripció ja existeix".* Si l'actualització s'ha pogut realitzar s'envia un SIGNAL a la Finestra Configuració perquè actualitzi el contingut de la taula. En cas contrari es mostra un missatge d'error amb el text: *"No s'han pogut actualitzar els camps".*

6.5.4 Gràfic amb l'històric de dades de temperatura i humitat

El gràfic amb l'històric de dades de temperatura i humitat mostra les dades capturades per la xarxa WSN d'un dia en concret.

Per generar un gràfic amb l'històric de dades de temperatura i humitat cal accedir a la Finestra Històric fent clic en el botó Històric de la Finestra Principal.



Il·lustració 104: *Finestra Històric*

La Finestra Històric permet visualitzar una llista (*QListView*) de la descripció de les *motes sensor* que tenen configurat el *thing token* de thethings.iO. El contingut de la llista s'obté de la base de dades i es transforma en *Model* a través de *QStandardItemModel* i s'afegeix al component *QListView*.

La Finestra Històric disposa de diverses opcions del gràfic a generar com són la selecció del dia a través del selector de dates i la selecció de les dades desitjades (temperatura, humitat, temperatura i humitat) marcant els *checkboxes*. Per defecte es selecciona el dia actual i l'opció temperatura i humitat.

Per crear el gràfic es necessari seleccionar una de les motes de la llista, configurar les opcions desitjades i prémer el botó Gràfic que obre la Finestra Gràfic Històric.

Per crear el component gràfic que permet la representació de les dades en la Finestra Gràfic Històric s'ha creat la classe GraphCanvas.py. Aquesta classe utilitza els objectes *Figure* i *FigureCanvasQTAgg* de la llibreria Matplotlib.

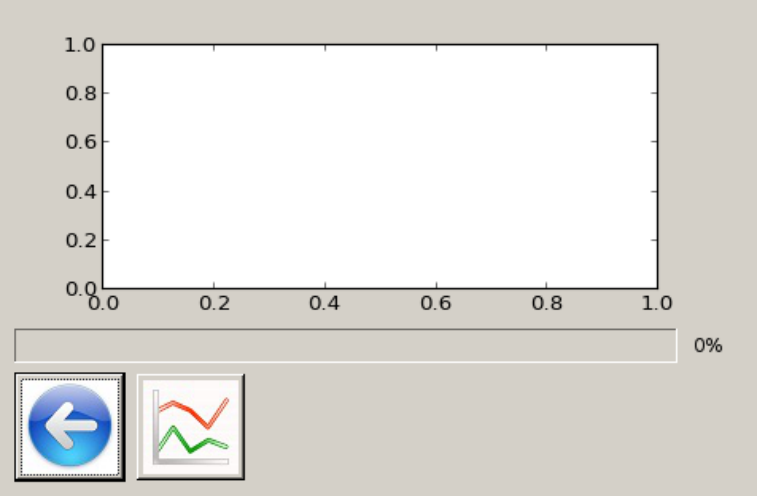
```
#Import genèric
import numpy as np

#Imports maplotlib
from matplotlib.backends.backend_qt4agg import FigureCanvasQTAgg as
FigureCanvas
from matplotlib.figure import Figure

#Classe per generar gràfic
class GraphCanvas(FigureCanvas):
   #Variable global
   data_labels=['00:00','06:00','12:00','18:00','24:00']
   #Constructor de la classe
   def __init__(self, width = 5, height = 4, dpi = 80):
      #Crear una figura
      self.figure = Figure(figsize=(width, height), dpi=dpi, facecolor = '#D4D0C8')
      #Afegir la figura en el Canvas
      FigureCanvas.__init__(self, self.figure)
      #Crear un eix
      self.ax1 = self.figure.add_subplot(111)
     self.ax1.hold(False)
```

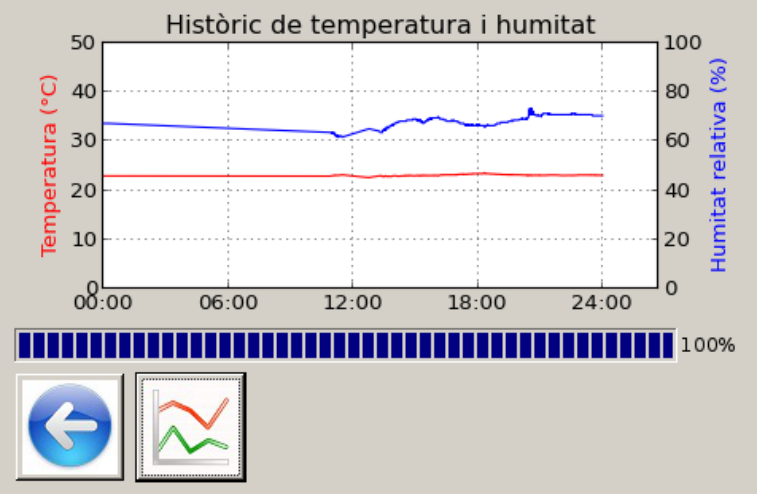
Taula 55: *Definició i constructor de la classe GraphCanvas.py*

En iniciar-se la Finestra Gràfic Històric el gràfic mostra els paràmetres per defecte.



Il·lustració 105: *Finestra Gràfic Històric sense valors*

La generació del gràfic es realitza fent clic al botó Gràfic. En prémer el botó s'inicia el procés de comunicació amb l'aplicació Read.js, s'obtenen les dades i es representen en el gràfic.

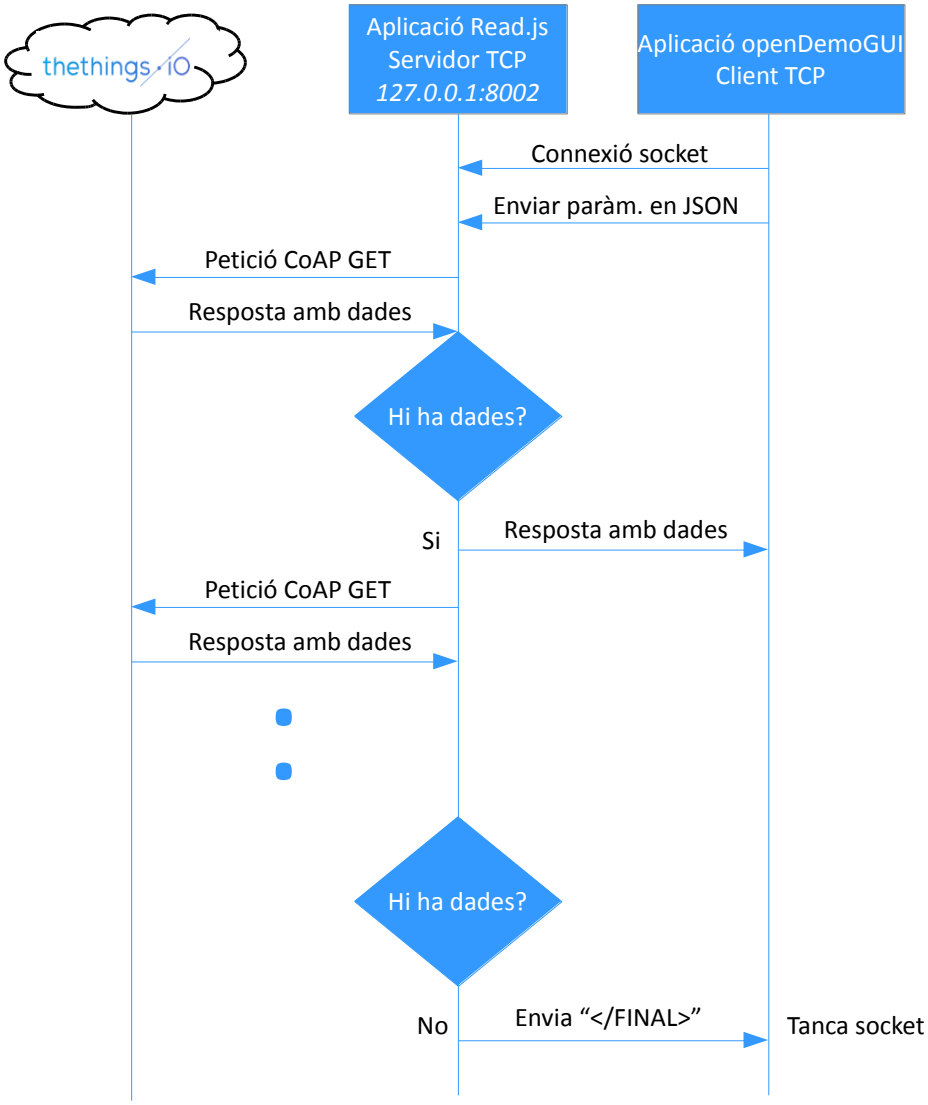


Il·lustració 106: *Finestra Gràfic Històric amb valors*

6.5.4.1 Llegir dades de thethings.iO

L'aplicació Read.js implementa un servidor TCP per escoltar peticions. En l'aplicació openDemoGUI es desenvolupen clients TCP que es connecten al servidor i envien els paràmetres de cerca en format JSON. Els paràmetres de cerca són: *key* ("temperature" o "humidity"), data inici i data final. L'aplicació Read.js "*parseja*" els paràmetres de cerca i realitza un petició CoAP GET amb aquests paràmetres sol·licitant el retorn de 100 (màxim valor suportat per la llibreria Node.js CoAP de thethings.iO) registres. openDemo emmagatzema en la plataforma thethings.iO dades cada minut i el gràfic a mostrar ha de ser les dades d'un dia sencer, per tant, els 100 registres no són suficients. Per obtenir les dades d'un dia sencer es realitza lectura per paginació de les dades de la plataforma thethings.iO a través d'una funció recursiva que es crida mentre la plataforma retorni dades. És adir, es realitzen iteracions de la consulta modificant el valor de la data final. Per a cada iteració s'envia les dades a l'aplicació openDemoGUI que les va acumulant. Quan Read.js detecta que la plataforma thethings.iO no retorna més informació envia a openDemoGUI el text </FINAL> per indicar que ha finalitzat el procés. Si durant aquest procés es produeix cap error de *timeout* en la connexió dels *sockets* es realitzen reintents.

En la següent il·lustració es pot veure l'esquema de la comunicació entre les aplicacions openDemoGUI, Read.js i la plataforma *cloud* thethings.iO.



Il·lustració 107: *Esquema del procés de comunicació entre openDemoGUI, Read.js i thethings.iO*

En els següents apartats s'explica el desenvolupament realitzat per dur a terme aquest procés en les aplicacions Read.js i openDemoGUI:

6.5.4.1.1 Aplicació Read.js

En l'aplicació Read.js s'implementa per una banda un servidor TCP en l'adreça *localhost* (127.0.0.1) i el port 8002 per rebre peticions de clients TCP amb les dades dels paràmetres de cerca en format JSON. Per l'altra banda, un client de la llibreria Node.js CoAP de thethings.iO per enviar peticions CoAP GET amb els paràmetres de cerca a la plataforma thethings.iO per obtenir les dades sol·licitades i enviar-les al client TCP. El codi font desenvolupat és:

```
//Capturar possibles errors
process.on('uncaughtException', function(err) {
        console.log(err);
});
//Importar mòdul net
var net = require('net')
//Port d'escolta del servidor
var port = 8002;
//Crear servidor TCP
net.createServer(function(socket){
   socket.on('data', function(data){
        //Parse dades JSON
        var json=JSON.parse(data);
        //Importar mòdul theThingsCoAP
        var theThingsCoAP = require('../../')
        //Crear client theThingsCoAP
        var client = theThingsCoAP.createClient()
        client.on('ready', function () {
                read(json.endDate)
        })
        //Funció per llegir dades de la plataforma thethings.iO
        function read(endDate){
        client.thingRead(json.key, {limit: 100,endDate: endDate, startDate:
                                        json.startDate}, function (error, data) {
                if (typeof data!=='undefined' && data!==null){
                        if (data.length > 0) {
                                var dataSend=""
                                var coma=","
                                for (var i=0;i<=(data.length - 1);i++){
        dataSend=dataSend+data[i].value+coma+data[i].datetime.split('T')[1]+coma
                                }
                                socket.write(dataSend);
                        read(data[data.length   -   1].datetime.split('.')[0].replace(/-/g,
                                                '').replace(/:/g, '').replace('T', ''))
                }else{
                                socket.write("</FINAL>");
                        }
                }
        })
        }

          });
//Configuració del port en el servidor TCP
}).listen(port);
```

Taula 56: *Codi font de l'aplicació Read.js*

6.5.4.1.2 Aplicació openDemoGUI

En openDemoGUI en prémer el botó Gràfic:

- Es crea un objecte Configthethingsio passant com a paràmetre el port 8002.
- Es connecta amb el servidor TCP de l'aplicació Read.js, sinó es pot connectar es mostra un missatge d'error.
- Es modifica el fitxer *config.json* de l'aplicació Read.js, sinó es pot modificar es mostra un missatge d'error.
- Es crea un objecte PlotThread (QThread), es configura dos SIGNALS i s'executa en un fil d'execució diferent.
- L'objecte PlotThread envia els paràmetres de cerca per la temperatura a l'aplicació Read.js a través de la funció *getToTheThingsio* de la classe Configthethingsio.py.

```
#Funció per llegir dades de la plataforma thethings.iO
#@param: key: "temperature" o "humidity"
#@param: startDate: data inicial
#@param: endDate: data final
def getToTheThingsio(self, key,startDate,endDate):
    dataReceived=""
    try:
      if self.s is not None:
        self.s.send('{"key":"'+key+'", "startDate":"'+startDate+'",
       "endDate":"'+endDate+'"}')
        while True:
          data=self.s.recv(self.BUFSIZ)
          if data == '</FINAL>':
            break;
          else:
            dataReceived= dataReceived+data
        dataReceived=dataReceived[:-1]
    except:
      Log.writeError("En la comunicacio amb thethings.iO")
      pass
    finally:
      return dataReceived
```

Taula 57: *Funció getToTheThingsio de la classe Configthethingsio.py*

- Una vegada obtingudes les dades de temperatura emet un SIGNAL per incrementar la barra de progrés de la Finestra Gràfic Històric.
- Processa les dades de temperatura, crea el vector de temps del l'eix X i dibuixa el gràfic en l'objecte GraphCanvas per mitjà del mètode *plotFigureTemperature*.

```
#Funció que representa els valors de temperatura
#@param: xdata: llista amb el valor dels temps en què s'ha emmagatzemat
#cada valor de temperatura
#@param: data: valors de temperatura
def plotFigureTemperature(self,xdata, data):
    try:
        self.ax1.plot(xdata,data, 'r-')
        self.ax1.set_ylim([0,50])
        self.ax1.set_ylabel(u'Temperatura (°C)',color='r')
    except:
        pass
```

Taula 58: *Funció plotFigureTemperature de la classe GraphCanvas.py*

- S'envia els paràmetres de cerca per l'humitat a l'aplicació Read.js a través de la funció *getToTheThingsio* de la classe Configthethingsio.py.
- Una vegada obtingudes les dades d'humitat emet un SIGNAL per incrementar la barra de progrés de la Finestra Gràfic Històric.
- Processa les dades d'humitat, crea el vector de temps del l'eix X i dibuixa el gràfic en l'objecte GraphCanvas per mitjà del mètode *plotFigureHumidity*.

```
#Funció que representa els valors d'humitat
#@param: xdata: llista amb el valor dels temps en què s'ha emmagatzemat
#cada valor d'humitat
#@param: data: valors d'humitat, isTemp: booleà per indicar que es vol mostrar
#la temperatura
def plotFigureHumidity(self,xdata,data, isTemp):
    try:
        if isTemp:
            self.ax2=self.ax1.twinx()
            self.ax2.plot(xdata,data, 'b-')
            self.ax2.set_ylim([0,100])
            self.ax2.set_ylabel(u'Humitat relativa (%)',color='b')
```

```
        else:
            self.ax1.plot(xdata,data, 'b-')
            self.ax1.set_ylim([0,100])
            self.ax1.set_ylabel(u'Humitat relativa (%)',color='b')
        except:
            pass
```

Taula 59: *Funció plotFigureHumidity de la classe GraphCanvas.py*

- Emet un SIGNAL per incrementar la barra de progrés de la Finestra Gràfic Històric.
- Per últim, dibuixa el títol i les etiquetes del eix X a través de la funció *setTitle* de la classe GraphCanvas.py.

```
#Funció que afegeix el títol i dibuixa el gràfic
#@param:isTemp: booleà per indicar que es vol mostrar la temperatura
#@param:isHum: booleà per indicar que es vol mostrar l'humitat
def setTitle(self,isTemp,isHum):
    text=''
    isBoth=False
    if isTemp and isHum:
        text= u'Històric de temperatura i humitat'
        isBoth=True
    elif isTemp and not isHum:
        text= u'Històric de temperatura'
    else:
        text= u'Històric d\'humitat relativa'
    self.ax1.set_title(text)
    pos = np.arange(0,1441,360)
    self.ax1.set_xticks(pos)
    if isBoth:
        try:
            self.ax2.set_xticks(pos)
        except:
            pass
    self.ax1.set_xticklabels(self.data_labels)
    self.ax1.grid(which="both")
    self.draw()
```

Taula 60: *Funció setTitle de la classe GraphCanvas.py*

El codi font de la classe PlotThread és:

```
#Import PyQT
from PyQt4 import QtCore

#Classe QThread per obtenir l'històric de valors de la plataforma thethings.iO i
representar-los gràficament
class PlotThread(QtCore.QThread):
   #Variables globals
   retries=2
   notifyProgress = QtCore.pyqtSignal()
   errorMessage = QtCore.pyqtSignal(str)

   #Constructor de la classe
   #@param: thethingsio: objecte Configthethingsio
   #@param: isTemp: booleà per indicar que es vol mostrar la temperatura
   #@params: isHum: booleà per indicar que es vol mostrar l'humitat, startDate:
         string amb la data inicial
   #@params: endDate: string amb la data final, hourUtcOffset: integer amb el
         òfset horari respecte UTC
   #@param: canvas: objecte GraphCanvas
   def __init__(self,thethingsio,isTemp,isHum,startDate,endDate,
         hourUtcOffset, canvas):
      QtCore.QThread.__init__(self)
      self.thethingsio=thethingsio
      self.isTemp=isTemp
      self.isHum=isHum
      self.startDate =startDate
      self.endDate=endDate
      self.hourUtcOffset=hourUtcOffset
      self.canvas =canvas

    #Funció que calcula el nombre de minuts d'una hora del dia
   #@param: value: hora del dia
   def getTemps(self,value):
      arrayValue=value.split(':')
      temps=int(arrayValue[0])*60+int(arrayValue[1])+60*self.hourUtcOffset
      if temps>=1440:
         temps=temps-1440
      return temps

    #Funció que es crida en realitzar el start() de la classe PlotThread
    def run(self):
       self.plotFigures()
```

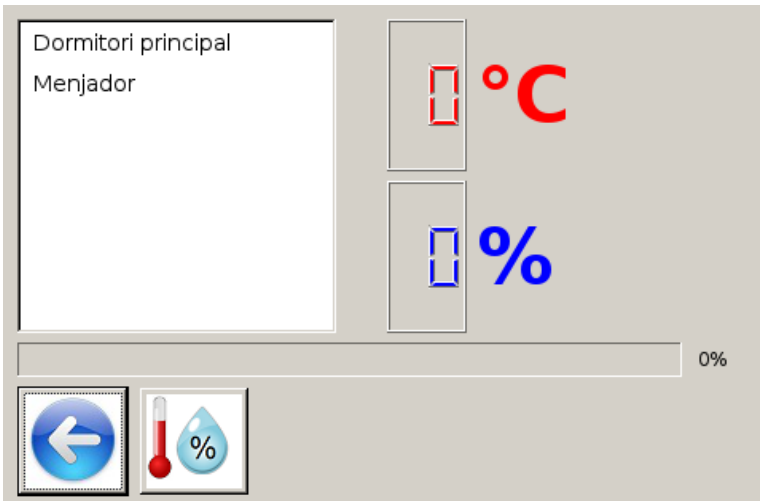
```
    #Funció que obté els valors de la plataforma thethigns.iO
    #@para: key: "temperature" o "homidity"
    def getData(self,key):
        i=0
        while(i<self.retries):
            data=self.thethingsio.getToTheThingsio(key,self.startDate,self.endDate)
            if data !="":
                break
            i=i+1
        return data
#Funció per obtenir l'històric de valors de la plataforma thethings.iO i representar-los gràficament
def plotFigures(self):
        dataTemp=''
        dataHum=''
        if self.isTemp:
            dataTemp= self.getData("temperature")
            self.notifyProgress.emit()
            if dataTemp !="":
                dataTemp=dataTemp.split(",")
                dataTemp.reverse()
                xdataTemp=list()
                for item in dataTemp[:]:
                    if 'Z' in item:
                        xdataTemp.append(self.getTemps(item))
                        dataTemp.remove(item)
                self.canvas.plotFigureTemperature(xdataTemp,dataTemp)
            else:
                self.errorMessage.emit(u"No es pot mostrar el gràfic de temperatura")
            self.notifyProgress.emit()
        if self.isHum:
            dataHum =  self.getData("humidity")
            self.notifyProgress.emit()
            if dataHum !="":
                dataHum=dataHum.split(",")
                dataHum.reverse()
                xdataHum=list()
                for item in dataHum[:]:
                    if 'Z' in item:
                        xdataHum.append(self.getTemps(item))
                        dataHum.remove(item)
                self.canvas.plotFigureHumidity(xdataHum,dataHum,self.isTemp)
            else:
                self.errorMessage.emit(u"No es pot mostrar el gràfic d'humitat")
            self.notifyProgress.emit()
```

```
        if dataTemp!='' or dataHum!='':
                self.canvas.setTitle(self.isTemp,self.isHum)
```

Taula 61: *Codi font de la classe PlotThread.py*

6.5.5 Mostrar els valors de temperatura i humitat de forma instantània

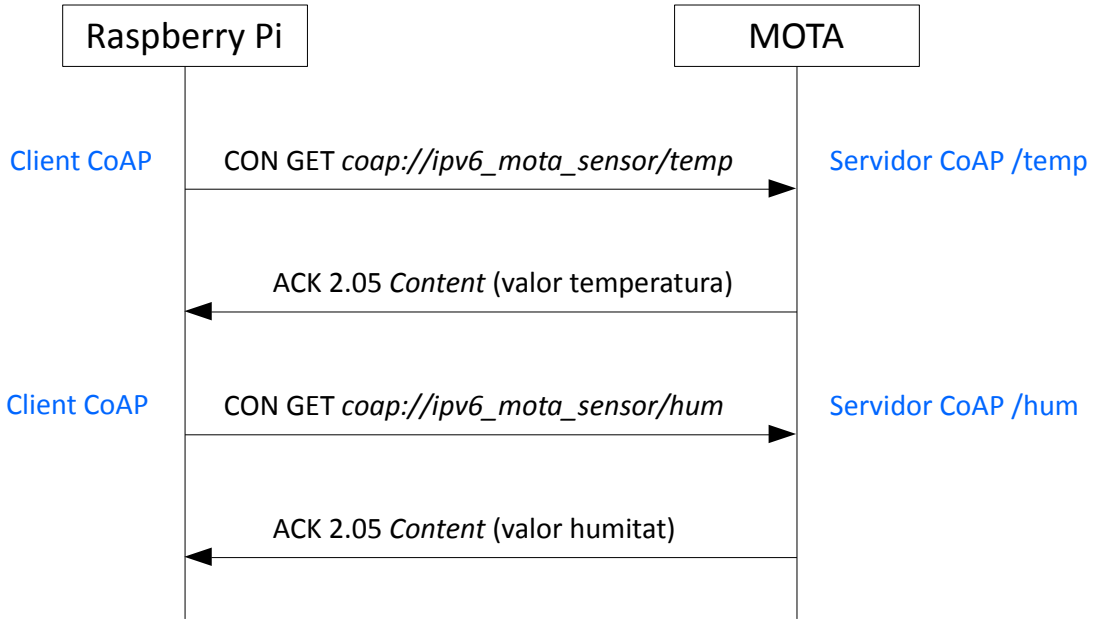
Per mostrar el valor de temperatura i humitat relativa de forma instantània per a qualsevol mota de la WSN cal accedir a la Finestra Valors fent clic en el botó Valors de la Finestra Principal.



Il·lustració 108: *Finestra Valors abans de la captura de dades*

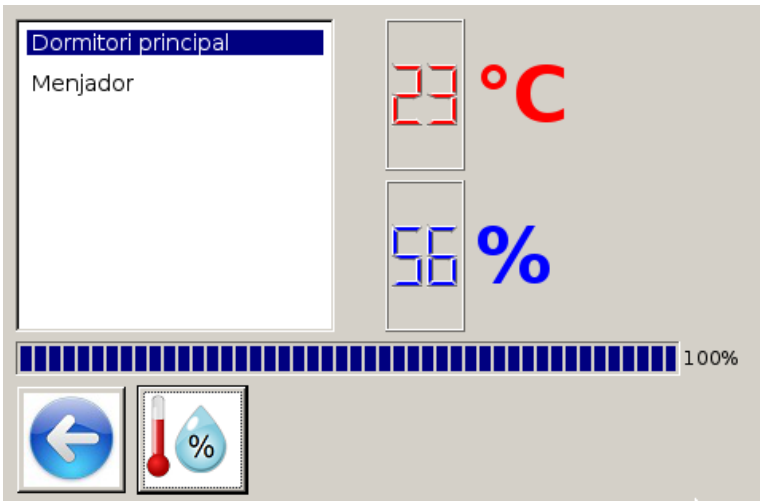
La Finestra Valors visualitza una llista (*QListView*) de la descripció de les *motes sensor* que tenen configurada la descripció de la mota. El contingut de la llista s'obté de la base de dades i es transforma en *Model* a través de *QStandardItemModel* i s'afegeix al component *QListView*.

Es selecciona una mota de la llista i es fa clic en el botó Valors per realitza una petició CoAP amb mètode GET pel recurs *"temp"* i una altra pel recurs *"hum"*. De la mateixa forma que en el procés de captura de dades de l'aplicació openDemo. En la següent il·lustració es pot veure com és la comunicació amb la mota.



Il·lustració 109: *Esquema del procés de comunicació per capturar les dades*

Una vegada obtinguts els valors de temperatura i humitat es calculen els valors reals i es mostren en els displays.



Il·lustració 110: *Finestra Valors després de la captura de dades*

El desenvolupament realitzat consisteix en:

- Crear un objecte ValuesThread (QThread), configurar-li un SIGNAL i executar-lo en un fil d'execució diferent.
- L'objecte ValuesThread envia una petició amb mètode GET al recurs "*temp*" per mitjà de la funció *sendGET* de la classe ObjectCoAP.

- Un cop obtingut el valor de temperatura de la mota, emet el seu valor amb un SIGNAL a la Finestra Valors. Aquesta calcula el valor real amb la funció *getTemperature* de la classe ConvertValue.py, el mostra en el display i incrementa la barra de progrés.
- L'objecte ValuesThread envia una petició amb mètode GET al recurs "*hum*" a través de la funció *sendGET* de la classe ObjectCoAP.
- Un cop obtingut el valor d'humitat relativa de la mota, emet el seu valor amb un SIGNAL a la Finestra Valors. Aquesta calcula el valor real amb la funció *getHumidity* de la classe ConvertValue.py, el mostra en el display i incrementa la barra de progrés.

El codi desenvolupat de la classe ValuesThread.py és:

```
#Import PyQT
from PyQt4 import QtCore
#Import openDemo
import ObjectCoAP

#Classe QThread per obtenir els valors de temperatura i humitat d'una mota
class ValuesThread(QtCore.QThread):
    #Variable global
    getValues = QtCore.pyqtSignal(int,list)
  #Constructor de la classe
  #@param: ipv6: adreça IPv6 de la mota
 def __init__(self,ipv6):
      QtCore.QThread.__init__(self)
      self.ipv6=ipv6

    #Funció per obtenir els valors de temperatura i humitat a través de la
          llibreria CoAP OpenWSN
    #retorna els valors utilitzant "SIGNALS" enviats a la Finestra Valors
    def getData(self):
      try:
         temp =ObjectCoAP.sendGET(self.ipv6,'temp')
         self.getValues.emit(0,temp)
         hum =ObjectCoAP.sendGET(self.ipv6,'hum' )
         self.getValues.emit(1,hum)
      except:
         self.getValues.emit(2,[0])
         pass
```

```
#Funció que es crida en realitzar el start() de la classe ValuesThread
def run(self):
   self.getData()
```

Taula 62: *Codi font de la classe ValuesThread.py*

6.6 Desplegament

En aquest apartat es detalla el desplegament del desenvolupament realitzat dividit en els punts: publicar codi font en GitHub, desplegament en la Raspberry i a la WSN.

6.6.1 Publicar codi font en GitHub

Abans de publicar el codi font en GitHub s'han afegit comentaris en tot el codi desenvolupat.

El procediment per publicar el codi font en GitHub consisteix en:

- Crear un compte en GitHub
- Crear els següents repositoris:
    - ***openDemo:*** format per les aplicacions openDemo, openDemoGUI i la llibreria Python CoAP de OpenWSN.

      https://github.com/rromero83/openDemo.git

    - ***openwsn-fw-openDemo:*** consta del firmware OpenWSN amb les aplicacions CoAP desenvolupades en el present projecte.

      https://github.com/rromero83/openwsn-fw-openDemo.git

    - ***thethingsio-coap-openDemo:*** formada per les aplicacions Read.js, Write.js i la llibreria Nodejs CoAP de thethings.iO.

      https://github.com/rromero83/thethingsio-coap-openDemo.git

- Publicar el codi font per a cada repositori. El procediment des de la línea de comandes consisteix en situar-se en el directori que conté el codi font i executar les següents comandes:

```
git init
git add .
git commit -m "first commit"
git remote add origin https://github.com/rromero83/openDemo.git
git push -u origin master
```

Il·lustració 111: *Comandes per publicar el codi font en el repositori GitHub*

En aquest cas es publica el codi font de openDemo, per la resta de repositoris cal situar-se en el directori que conté el codi font i substituir per la URL corresponent.

6.6.2 Desplegament en la Raspberry Pi

El desplegament de la solució del present Treball Final de Màster en la Raspberry Pi es divideix en dues parts. La primera consisteix en la descàrrega del codi font necessari des dels repositoris de GitHub, i la segona, basada en l'execució del software necessari pel funcionament del demostrador de forma automàtica en el procés d'arrencada del sistema operatiu.

6.6.2.1 Descàrrega del codi font

El desplegament del codi font dels repositoris *thethingsio-coap-openDemo* i *openDemo* s'ha de realitzar en el directori **"/usr/local/"**.

La descàrrega del codi font del repositori *thethingsio-coap-openDemo* es realitza de la següent forma:

```
sudo git clone https://github.com/rromero83/thethingsio-coap-openDemo.git
```

Il·lustració 112: *Descàrrega de thethingsio-coap-openDemo des del repositori*

La descàrrega del codi font del repositori *openDemo* es realitza de la següent forma:

```
sudo git clone https://github.com/rromero83/openDemo.git
```

Il·lustració 113: *Descàrrega de openDemo des del repositori*

6.6.2.2 Executar les aplicacions en el procés d'arrencada

Les aplicacions que s'han d'executar de forma automàtica durant el procés d'arrencada del sistema operatiu són:

- OpenVisualizer en mode web.
- Write.js
- Read.js
- openDemo
- openDemoGUI

La configuració de les tres primeres aplicacions perquè s'executin en el procés d'arrencada es basa en afegir les següents comandes en el fitxer **/etc/rc.local**, després del comentaris i abans de la línea *"exit 0"*.

```
cd /usr/local/openwsn-sw/software/openvisualizer/bin/openVisualizerApp
sudo python openVisualizerWeb.py &

cd /usr/local/thethingsio-coap-openDemo/openDemo/write
sudo node Write.js &

cd /usr/local/thethingsio-coap-openDemo/openDemo/read
sudo node Read.js &
```

Il·lustració 114: *Comandes a afegir al fitxer /etc/rc.local*

Les aplicacions openDemo i openDemoGUI estan desenvolupades en el mateix programa. openDemoGUI és una aplicació amb interfície gràfica i per executar-se en el procés d'arrencada s'ha de configurar d'una altra forma que les aplicacions anteriors. A més, no es vol veure en la pantalla PiScreen l'escriptori del sistema operatiu. Per aquest motius, la configuració de les aplicacions openDemo i openDemoGUI es realitza afegint la següent comanda en el fitxer **~/.xinitrc**.

```
sudo /usr/bin/python /usr/local/openDemo/openDemoApp.py
```

Il·lustració 115: *Comanda a afegir en el fitxer ~/.xinitrc*

6.6.3 Desplegament a la WSN

El desplegament de la WSN consisteix en carregar el *firmware* a les motes a través de l'eina Eclipse i el JTAG JLink explicat anteriorment. El *firmware* s'obté de la descàrrega del repositori *openwsn-fw-openDemo* amb la següent comanda:

```
sudo git clone https://github.com/rromero83/openwsn-fw-openDemo.git
```

Il·lustració 116: *Descàrrega de openwsn-fw-openDemo des del repositori*

Per la mota que realitza la funció de DAGroot cal activar el "*#define DAGROOT*" en la classe idmanager.c abans de compilar i carregar el codi en la mota.

idmanager.c

```
12 //========================== prototypes ======================
13
14 //========================== public ==========================
15
16 void idmanager_init() {
17
18    // reset local variables
19    memset(&idmanager_vars, 0, sizeof(idmanager_vars_t));
20 #define DAGROOT//MODIFICAT
21    // isDAGroot
22 #ifdef DAGROOT
23    idmanager_vars.isDAGroot            = TRUE;
24 #else
25    idmanager_vars.isDAGroot            = FALSE;
26 #endif
27
28    // myPANID
29    idmanager_vars.myPANID.type         = ADDR_PANID;
```

Il·lustració 117: *Configurar firmware openwsn-fw-openDemo com DAGroot*

7 Verificació

En el present capítol es presenta la verificació del correcte funcionament del demostrador desenvolupat.

El procés de verificació del demostrador consisteix en realitzar captures del trànsit de la comunicació de la WSN per les diverses funcionalitats del demostrador amb l'eina Wireshark, validar els valors capturats i realitzar una prova de funcionament de captures de valors de temperatura i humitat relativa durant un dia per la seva representació gràfica.

7.1 Captures de tràfic

Com s'ha comentat anteriorment, la xarxa de sensors (WSN) del demostrador està format per dues *motes sensor* i una *mota actuador* connectada a un mòdul de relés per actuar sobre el sistema de calefacció.

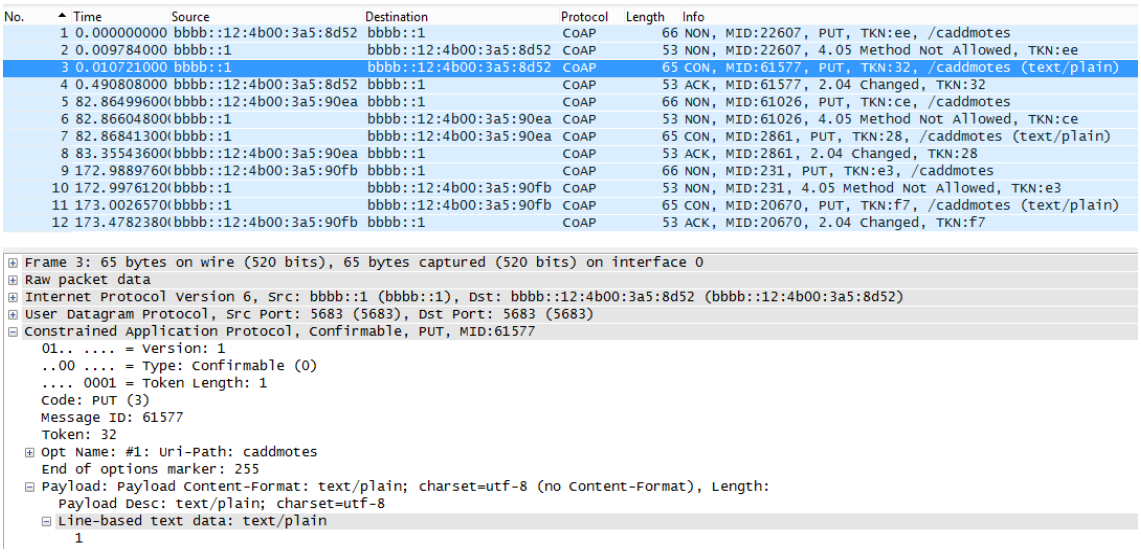
L'escenari de verificació parteix en què les *motes sensors* i *actuador* es troben apagades i es van encenen de forma progressiva espaiats en el temps per més de 30 segons.

7.1.1 Alta adreces IPv6

En el procés d'arrencada de les motes s'inicia el procés d'alta de les seves adreces IPv6 en la base de dades del demostrador.

En la següent il·lustració es pot veure com cada mota realitza l'enviament de peticions CoAP de tipus NON amb mètode PUT a la llibreria Python CoAP de OpenWSN en el recurs *caddmotes*. En rebre la petició, es processa i s'envia una petició CoAP de tipus CON amb mètode PUT i *payload* "1" a la mota en el recurs *caddmotes* perquè deixi d'enviar peticions cada 30 segons. Si la mota processa correctament la petició, envia una resposta CoAP de tipus ACK amb el codi *2.04 Changed*. Les diverses motes han processat de forma satisfactòria les peticions per deixar d'enviar i es pot observar com només hi ha una petició d'alta per mota. Per tant, en la captura del tràfic es pot verificar el correcte funcionament del procés d'alta d'adreces IPv6.

Els paquets 2, 6 i 10 són respostes per defecte que envia la llibreria Python CoAP de OpenWSN en rebre peticions CoAP amb mètode PUT, com això no interfereix en el correcte funcionament s'ha decidit no eliminar-ho.



Il·lustració 118: *Captura de tràfic de la comunicació del procés d'alta de motes*

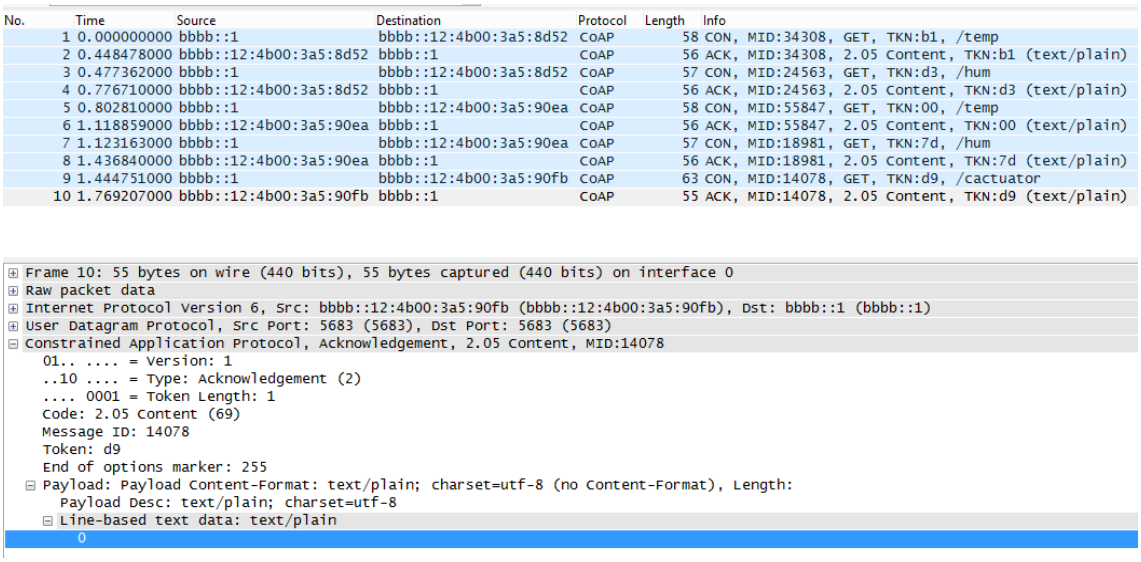
7.1.2 Captura de dades

Des de la interfície gràfica d'usuari (GUI) es configuren la descripció, el *thing token* i el tipus de mota per a cada node de la WSN. Les motes amb les adreces IPv6 bbbb::12:4b00:3a5:8d52 i bbbb::12:4b00:3a5:90ea es configuren com a *motes sensor*, mentre que la mota amb IPv6 bbbb::12:4b00:3a5:90fb es configura com a *mota actuador.* La mota actuador no és necessari configurar el *thing token*.

Una vegada configurades les motes, es realitza el procés de captura dels valors de temperatura i humitat cada minut per a cada mota sensor i el procés de control del sistema de calefacció. La situació actual de l'escenari de verificació és que el valor de temperatura de consigna és inferior als valors de temperatura capturats de la WSN i el sistema de calefacció està apagat.

En la següent il·lustració es pot observar com per a cada *mota sensor* es realitza una petició CoAP de tipus CON amb mètode GET al recurs *temp* i una altra al recurs *hum*. Així com, les motes responen amb un paquet CoAP de tipus ACK amb el codi *2.05 Content* i en el payload la dada sol·licitada anteriorment. A més, es pot veure que un cop finalitzat el procés de captura, es realitza el procés de control del sistema de

calefacció. El procés de control realitza una petició CoAP de tipus CON amb mètode GET al recurs *cactuator* de la *mota actuador* per obtenir l'estat del sistema de calefacció. La mota actuador respon amb un paquet CoAP tipus ACK amb codi *2.05 Content* i en el *payload* l'estat demanat. En aquest cas, el sistema de calefacció està apagat i el *payload* és "0". Com el valor de temperatura de consigna és inferior als valors de temperatura capturats a la WSN i l'estat del sistema de calefacció està apagat ("0"), no cal enviar cap ordre a la mota actuador per actuar sobre el sistema de calefacció.



Il·lustració 119: *Captura de tràfic de la comunicació dels processos de captura de dades i control*

7.1.3 Control sistema calefacció

El funcionament del control del sistema de calefacció es basa en encendre la calefacció quan està apagada i el valor de temperatura de consigna és superior als valors de temperatura capturats en la WSN. Així mateix, el sistema de calefacció s'apaga quan està encès i el valor de temperatura de consigna és inferior als valors de temperatura capturats en la WSN.

S'incrementa el valor de temperatura de consigna per sobre dels valors de temperatura capturats en la WSN.

En la il·lustració següent es pot veure com després d'obtenir l'estat del sistema de calefacció (apagat), s'envia una petició CoAP de tipus CON amb mètode GET al recurs

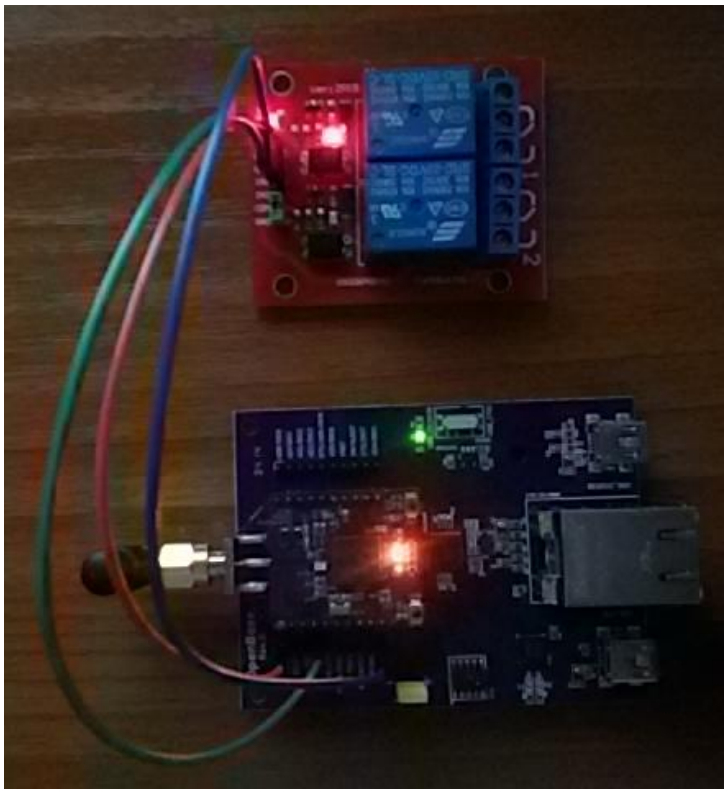
cactuator amb *payload* "1" a la *mota actuador*. La *mota actuador* encén el sistema de calefacció mitjançant l'activació del GPIO que controla l'entrada del mòdul de relés. A més, respon a la petició amb un paquet CoAP de tipus ACK amb codi *2.04 Changed*.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	bbbb::1	bbbb::12:4b00:3a5:8d52	CoAP	58	CON, MID:25371, GET, TKN:6d, /temp
2	0.322337000	bbbb::12:4b00:3a5:8d52	bbbb::1	CoAP	56	ACK, MID:25371, 2.05 Content, TKN:6d (text/plain)
3	0.372674000	bbbb::1	bbbb::12:4b00:3a5:8d52	CoAP	57	CON, MID:9809, GET, TKN:b0, /hum
4	0.653969000	bbbb::12:4b00:3a5:8d52	bbbb::1	CoAP	56	ACK, MID:9809, 2.05 Content, TKN:b0 (text/plain)
5	0.697120000	bbbb::1	bbbb::12:4b00:3a5:90ea	CoAP	58	CON, MID:33523, GET, TKN:be, /temp
6	0.981617000	bbbb::12:4b00:3a5:90ea	bbbb::1	CoAP	56	ACK, MID:33523, 2.05 Content, TKN:be (text/plain)
7	1.017269000	bbbb::1	bbbb::12:4b00:3a5:90ea	CoAP	57	CON, MID:41056, GET, TKN:4d, /hum
8	1.313705000	bbbb::12:4b00:3a5:90ea	bbbb::1	CoAP	56	ACK, MID:41056, 2.05 Content, TKN:4d (text/plain)
9	1.339759000	bbbb::1	bbbb::12:4b00:3a5:90fb	CoAP	63	CON, MID:49901, GET, TKN:e0, /cactuator
10	1.643339000	bbbb::12:4b00:3a5:90fb	bbbb::1	CoAP	55	ACK, MID:49901, 2.05 Content, TKN:e0 (text/plain)
11	1.660109000	bbbb::1	bbbb::12:4b00:3a5:90fb	CoAP	65	CON, MID:52422, PUT, TKN:f9, /cactuator (text/plain)
12	1.976176000	bbbb::12:4b00:3a5:90fb	bbbb::1	CoAP	53	ACK, MID:52422, 2.04 Changed, TKN:f9

```
⊞ Frame 11: 65 bytes on wire (520 bits), 65 bytes captured (520 bits) on interface 0
⊞ Raw packet data
⊞ Internet Protocol Version 6, Src: bbbb::1 (bbbb::1), Dst: bbbb::12:4b00:3a5:90fb (bbbb::12:4b00:3a5:90fb)
⊞ User Datagram Protocol, Src Port: 5683 (5683), Dst Port: 5683 (5683)
⊟ Constrained Application Protocol, Confirmable, PUT, MID:52422
    01.. .... = Version: 1
    ..00 .... = Type: Confirmable (0)
    .... 0001 = Token Length: 1
    Code: PUT (3)
    Message ID: 52422
    Token: f9
  ⊞ Opt Name: #1: Uri-Path: cactuator
    End of options marker: 255
  ⊟ Payload: Payload Content-Format: text/plain; charset=utf-8 (no Content-Format), Length:
      Payload Desc: text/plain; charset=utf-8
    ⊟ Line-based text data: text/plain
        1
```

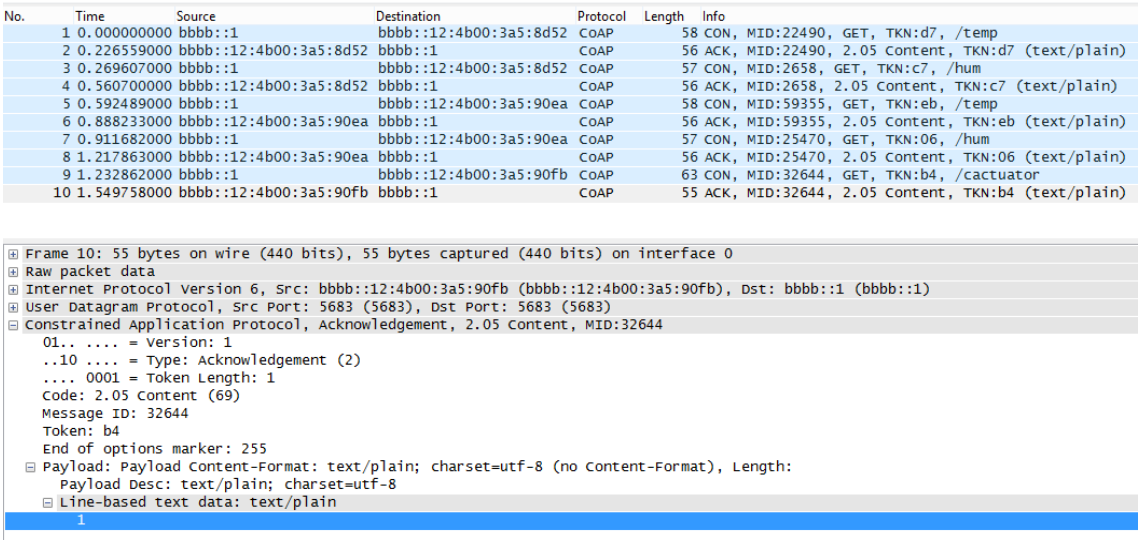
Il·lustració 120: *Captura de tràfic de la comunicació del procés de control del sistema de calefacció, encendre calefacció*

En la il·lustració de sota es pot veure com s'ha activat l'entrada del mòdul de relés.



Il·lustració 121: *Imatge de l'entrada del mòdul de relés activada*

En la captura de tràfic de sota es pot observar com l'estat actual del sistema de calefacció és encès ("1") i com no s'envia cap ordre a la *mota actuador* perquè el valor de temperatura de consigna és superior als valors de temperatura capturats en la WSN.



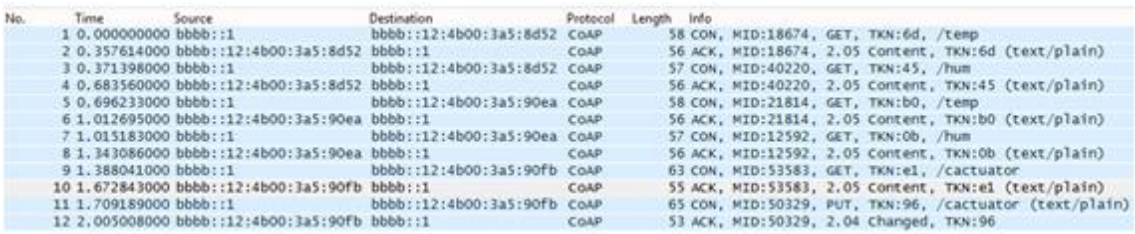
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	bbbb::1	bbbb::12:4b00:3a5:8d52	CoAP	58	CON, MID:22490, GET, TKN:d7, /temp
2	0.226559000	bbbb::12:4b00:3a5:8d52	bbbb::1	CoAP	56	ACK, MID:22490, 2.05 Content, TKN:d7 (text/plain)
3	0.269607000	bbbb::1	bbbb::12:4b00:3a5:8d52	CoAP	57	CON, MID:2658, GET, TKN:c7, /hum
4	0.560700000	bbbb::12:4b00:3a5:8d52	bbbb::1	CoAP	56	ACK, MID:2658, 2.05 Content, TKN:c7 (text/plain)
5	0.592489000	bbbb::1	bbbb::12:4b00:3a5:90ea	CoAP	58	CON, MID:59355, GET, TKN:eb, /temp
6	0.888233000	bbbb::12:4b00:3a5:90ea	bbbb::1	CoAP	56	ACK, MID:59355, 2.05 Content, TKN:eb (text/plain)
7	0.911682000	bbbb::1	bbbb::12:4b00:3a5:90ea	CoAP	57	CON, MID:25470, GET, TKN:06, /hum
8	1.217863000	bbbb::12:4b00:3a5:90ea	bbbb::1	CoAP	56	ACK, MID:25470, 2.05 Content, TKN:06 (text/plain)
9	1.232862000	bbbb::1	bbbb::12:4b00:3a5:90fb	CoAP	63	CON, MID:32644, GET, TKN:b4, /cactuator
10	1.549758000	bbbb::12:4b00:3a5:90fb	bbbb::1	CoAP	55	ACK, MID:32644, 2.05 Content, TKN:b4 (text/plain)

```
⊞ Frame 10: 55 bytes on wire (440 bits), 55 bytes captured (440 bits) on interface 0
⊞ Raw packet data
⊞ Internet Protocol Version 6, Src: bbbb::12:4b00:3a5:90fb (bbbb::12:4b00:3a5:90fb), Dst: bbbb::1 (bbbb::1)
⊞ User Datagram Protocol, Src Port: 5683 (5683), Dst Port: 5683 (5683)
⊟ Constrained Application Protocol, Acknowledgement, 2.05 Content, MID:32644
    01.. .... = Version: 1
    ..10 .... = Type: Acknowledgement (2)
    .... 0001 = Token Length: 1
    Code: 2.05 Content (69)
    Message ID: 32644
    Token: b4
    End of options marker: 255
  ⊟ Payload: Payload Content-Format: text/plain; charset=utf-8 (no Content-Format), Length:
      Payload Desc: text/plain; charset=utf-8
    ⊟ Line-based text data: text/plain
        1
```

Il·lustració 122: *Captura de tràfic de la comunicació del procés de control del sistema de calefacció, no s'envia ordre d'actuació*

Es baixa el valor de la temperatura de consigna per sota dels valors de temperatura capturats per la WSN.

En les captures següents es pot observar que l'estat del sistema de calefacció és encès ("1") i s'envia una petició CoAP de tipus CON amb mètode GET al recurs *cactuator* amb *payload* "0" a la *mota actuador*. La *mota actuador* apaga el sistema de calefacció mitjançant la desactivació del GPIO que controla l'entrada del mòdul de relés. A més, respon a la petició amb un paquet CoAP de tipus ACK amb codi *2.04 Changed*.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	bbbb::1	bbbb::12:4b00:3a5:8d52	CoAP	58	CON, MID:18674, GET, TKN:6d, /temp
2	0.357614000	bbbb::12:4b00:3a5:8d52	bbbb::1	CoAP	56	ACK, MID:18674, 2.05 Content, TKN:6d (text/plain)
3	0.371398000	bbbb::1	bbbb::12:4b00:3a5:8d52	CoAP	57	CON, MID:40220, GET, TKN:45, /hum
4	0.683560000	bbbb::12:4b00:3a5:8d52	bbbb::1	CoAP	56	ACK, MID:40220, 2.05 Content, TKN:45 (text/plain)
5	0.696233000	bbbb::1	bbbb::12:4b00:3a5:90ea	CoAP	58	CON, MID:21814, GET, TKN:b0, /temp
6	1.012695000	bbbb::12:4b00:3a5:90ea	bbbb::1	CoAP	56	ACK, MID:21814, 2.05 Content, TKN:b0 (text/plain)
7	1.015183000	bbbb::1	bbbb::12:4b00:3a5:90ea	CoAP	57	CON, MID:12592, GET, TKN:0b, /hum
8	1.343086000	bbbb::12:4b00:3a5:90ea	bbbb::1	CoAP	56	ACK, MID:12592, 2.05 Content, TKN:0b (text/plain)
9	1.388041000	bbbb::1	bbbb::12:4b00:3a5:90fb	CoAP	63	CON, MID:53583, GET, TKN:e1, /cactuator
10	1.672843000	bbbb::12:4b00:3a5:90fb	bbbb::1	CoAP	55	ACK, MID:53583, 2.05 Content, TKN:e1 (text/plain)
11	1.709189000	bbbb::1	bbbb::12:4b00:3a5:90fb	CoAP	65	CON, MID:50329, PUT, TKN:96, /cactuator (text/plain)
12	2.005008000	bbbb::12:4b00:3a5:90fb	bbbb::1	CoAP	53	ACK, MID:50329, 2.04 Changed, TKN:96

Il·lustració 123: *Captura de tràfic de la comunicació del procés de control del sistema de calefacció, apagar calefacció*

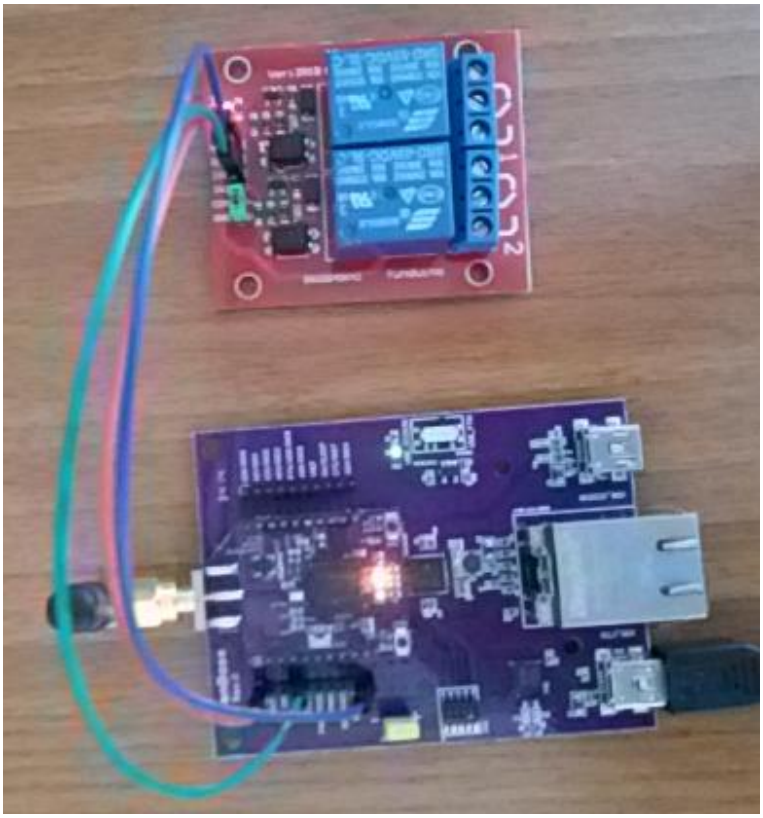
```
⊞ Frame 10: 55 bytes on wire (440 bits), 55 bytes captured (440 bits) on interface 0
⊞ Raw packet data
⊞ Internet Protocol Version 6, Src: bbbb::12:4b00:3a5:90fb (bbbb::12:4b00:3a5:90fb), Dst: bbbb::1 (bbbb::1)
⊞ User Datagram Protocol, Src Port: 5683 (5683), Dst Port: 5683 (5683)
⊟ Constrained Application Protocol, Acknowledgement, 2.05 Content, MID:53583
    01.. .... = Version: 1
    ..10 .... = Type: Acknowledgement (2)
    .... 0001 = Token Length: 1
    Code: 2.05 Content (69)
    Message ID: 53583
    Token: e1
    End of options marker: 255
  ⊟ Payload: Payload Content-Format: text/plain; charset=utf-8 (no Content-Format), Length:
      Payload Desc: text/plain; charset=utf-8
    ⊟ Line-based text data: text/plain
        1
```

Il·lustració 124: *Capçaleres CoAP del paquet de l'estat del sistema de calefacció*

```
⊞ Frame 11: 65 bytes on wire (520 bits), 65 bytes captured (520 bits) on interface 0
⊞ Raw packet data
⊞ Internet Protocol Version 6, Src: bbbb::1 (bbbb::1), Dst: bbbb::12:4b00:3a5:90fb (bbbb::12:4b00:3a5:90fb)
⊞ User Datagram Protocol, Src Port: 5683 (5683), Dst Port: 5683 (5683)
⊟ Constrained Application Protocol, Confirmable, PUT, MID:50329
    01.. .... = Version: 1
    ..00 .... = Type: Confirmable (0)
    .... 0001 = Token Length: 1
    Code: PUT (3)
    Message ID: 50329
    Token: 96
  ⊞ Opt Name: #1: Uri-Path: cactuator
    End of options marker: 255
  ⊟ Payload: Payload Content-Format: text/plain; charset=utf-8 (no Content-Format), Length:
      Payload Desc: text/plain; charset=utf-8
    ⊟ Line-based text data: text/plain
        0
```

Il·lustració 125: *Capçaleres CoAP del paquet per apagar el sistema de calefacció*

En la il·lustració de sota es pot veure com s'ha desactivat l'entrada del mòdul de relés.



Il·lustració 126: *Imatge de l'entrada del mòdul de relés desactivada*

7.2 Validar valors capturats

El sensor Sensirion SHT21 integrat en OpenBattery està calibrat de fàbrica.

La validació dels valors de temperatura i humitat relativa capturats en la WSN es basa en la comparació de valors per un mateix període de temps i es realitza de dues formes. La primera, es compara els valors capturats per les dues *motes sensors*. La segona, es compara els valors capturats de temperatura amb el valor de temperatura proporcionat per un termòstat comercial.

7.2.1 Validar valors entre dues motes

La validació es realitza mitjaçant la comparació dels valors capturats de temperatura i humitat relativa de les dues *motes sensors*.

7.2.1.1 Temperatura

En la següent il·lustració es poden veure els valors capturats de temperatura (°C) per les dues *motes sensors*. Aquests valors s'han obtingut de la plataforma thethings.iO.

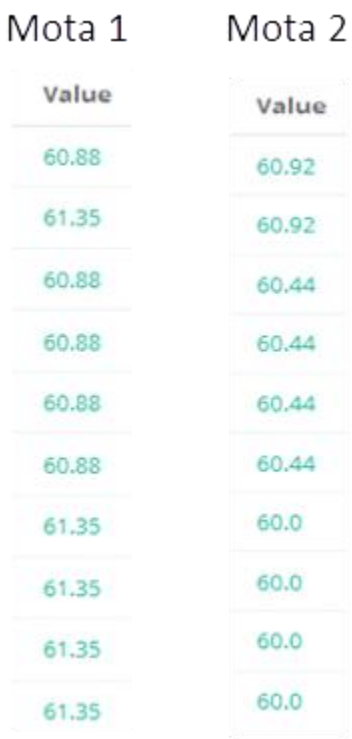
Mota 1	Mota 2
Value	Value
27.22	27.09
27.3	27.09
27.26	27.18
27.3	27.3
27.3	27.26
27.3	27.3
27.26	27.43
27.26	27.48
27.26	27.48
27.26	27.43

Il·lustració 127: *Valors de temperatura (°C) de les dues motes sensors*

Es pot apreciar que els valors de temperatura de les dues motes són molt semblants. En els pitjors casos, la variació és d'aproximadament 0.2°C. Els valors de la mota 1 presenten unes mesures més precises que la mota 2.

7.2.1.2 Humitat relativa

En la il·lustració de sota es poden observar els valors capturats d'humitat relativa (%) per les dues *motes sensors*. Aquests valors s'han obtingut de la plataforma thethings.iO.



Mota 1	Mota 2
Value	Value
60.88	60.92
61.35	60.92
60.88	60.44
60.88	60.44
60.88	60.44
60.88	60.44
61.35	60.0
61.35	60.0
61.35	60.0
61.35	60.0

Il·lustració 128: *Valors d'humitat relativa (%) de les dues motes sensors*

Es pot veure que els valors d'humitat relativa de les dues motes són semblants. En els pitjors casos, la variació és d'aproximadament 1.35 %.

7.2.2 Validar valors de temperatura amb termòstat comercial

La validació es du a terme a través de la comparació dels valors capturats de temperatura de les dues *motes sensors* i el termòstat comercial Honeywell DT92A.

El termòstat Honeywell DT92A disposa d'una resolució de 0.5°C.

En la següent il·lustració es pot observar com el valor de temperatura (27°C) del termòstat és molt semblant als obtinguts en la Il·lustració 127. Així mateix, es poden veure els valors de temperatura i humitat instantanis per la mota amb la descripció *Dormitori principal*. S'observa com el valor instantani de temperatura (27°C) coincideix amb el valor del termòstat.

Il·lustració 129: *Validar valor de temperatura entre el termòstat i la mota "Dormitori principal"*

En la següent il·lustració es pot veure els valors de temperatura i humitat instantanis per la mota amb la descripció *Menjador*. S'observa com el valor instantani de temperatura (27°C) coincideix amb el valor del termòstat.

Il·lustració 130: *Validar valor de temperatura entre el termòstat i la mota "Menjador"*

Per tant, es pot afirmar que els valors de temperatura capturats per les motes sensors són exactes, és a dir, valors molt propers als valors verdaders.

7.3 Prova de funcionament

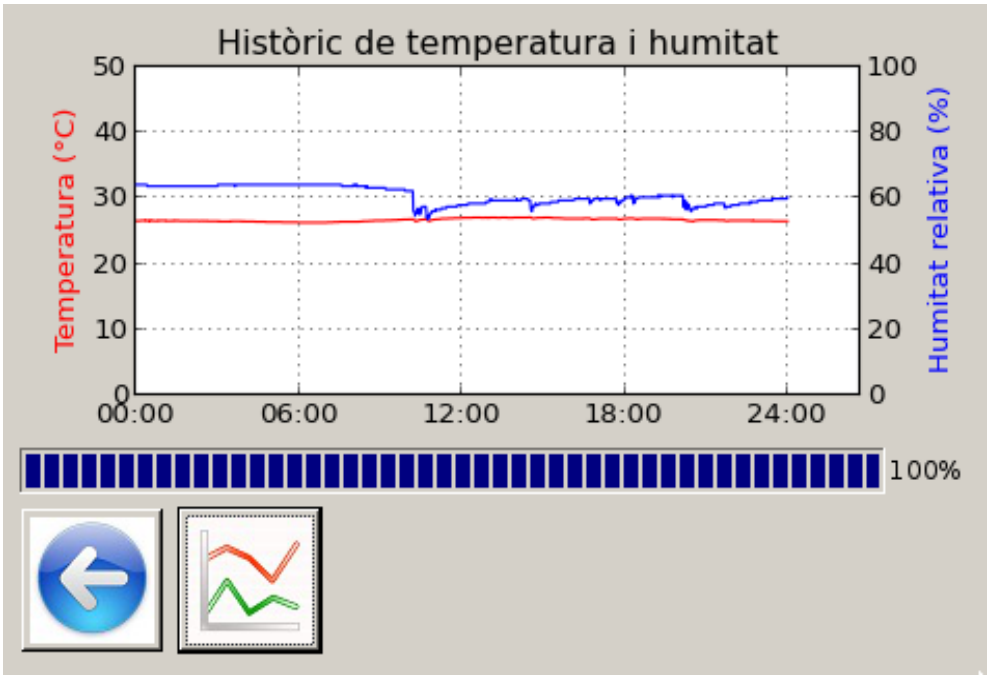
La prova de funcionament consisteix en capturar els valors de temperatura i humitat relativa durant un dia i la representació gràfica dels mateixos. La representació gràfica es realitza de dues maneres, una obtinguda a través de l'aplicació *openDemoGUI* i l'altra mitjançant la plataforma thethings.iO.

7.3.1 Aplicació *openDemoGUI*

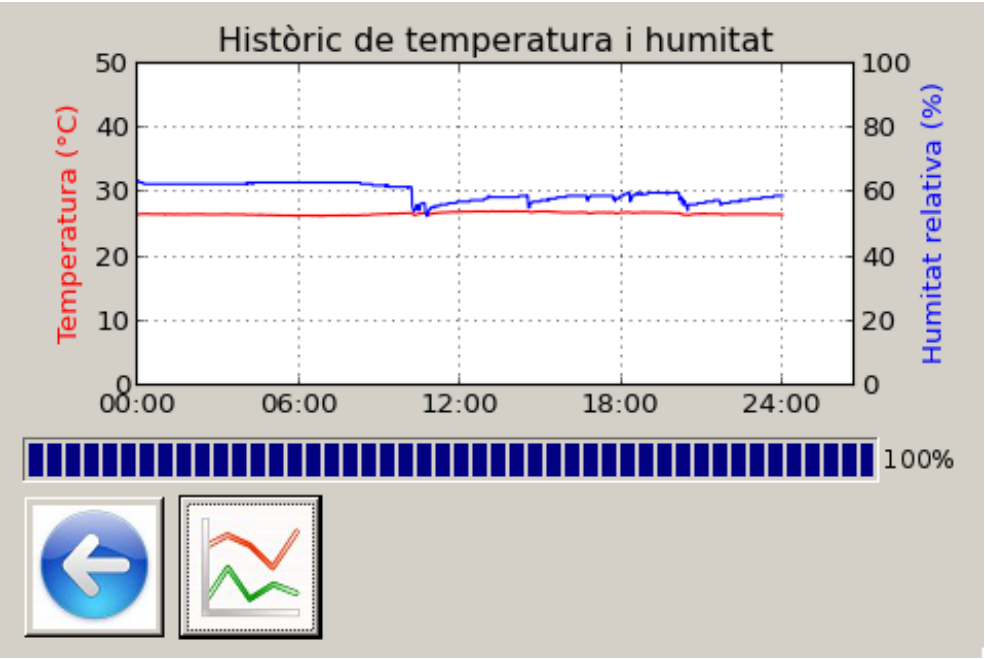
En les següents il·lustracions es poden veure les representacions gràfiques dels valors de temperatura i humitat relativa capturats durant un dia per les dues *motes sensors*. Primer la mota amb descripció *Dormitori principal* i després la mota amb descripció *Menjador*. En les dues gràfiques es poden observar com els valors obtinguts de temperatura i humitat relativa de les dues *motes sensors* són molt semblants, confirmant els resultats del punt *7.2.1 Validar valors entre dues motes*.

Els valors de temperatura són gairebé constants durant tot el dia. Uns 26°C per la matinada i 27°C durant el dia. Les variacions són gairebé inapreciables. En la representació gràfica mitjançant la plataforma thethings.iO es poden veure millor les variacions. En els valors d'humitat relativa es poden veure petites variacions sofertes. Durant la matinada s'ha obtingut una humitat d'aproximadament del 63% i durant el dia per sota del 60%.

Els pics de baixada corresponen a obertures de la finestra de l'estança on estaven situades les *motes sensors*.



Il·lustració 131: *Representació gràfica de les mesures capturades de la mota "Dormitori principal" amb openDemoGUI*



Il·lustració 132: *Representació gràfica de les mesures capturades de la mota "Menjador" amb openDemoGUI*

7.3.2 Plataforma thethings.iO

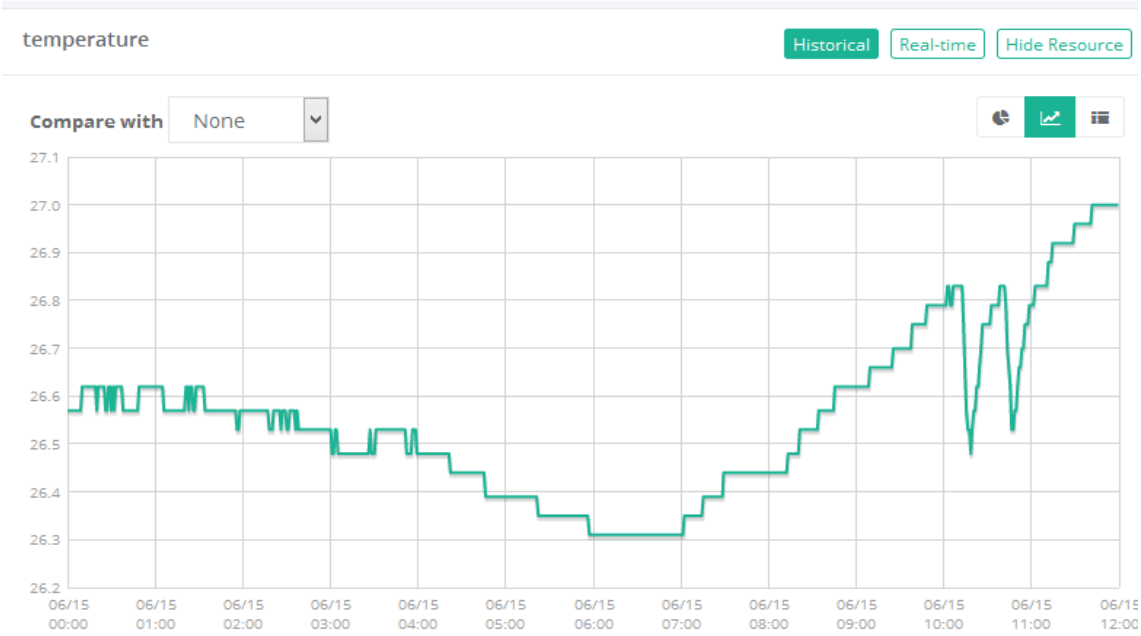
La plataforma thethings.iO no permet la visualització de les dades d'un dia sencer. Com s'ha pogut verificar anteriorment, els valors de les dues motes sensors són gairebé els mateixos. Per aquests motius es mostra la representació gràfica de les 12 primeres hores de captures de la mota amb descripció *Dormitori Principal* i les altres 12 hores corresponen a la mota amb descripció *Menjador*.

En tots els gràfics, els pics de baixada corresponen a obertures de la finestra de l'estança on estaven situades les *motes sensors*.

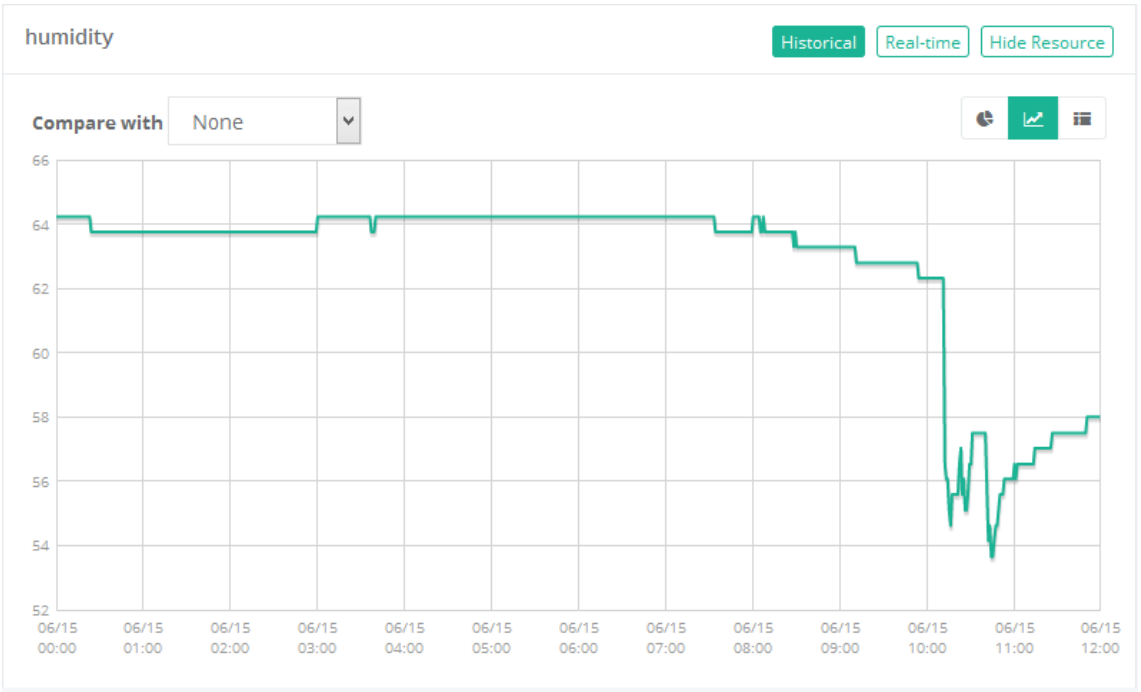
En les següents il·lustracions es poden visualitzar els gràfics de temperatura i humitat relativa de les 12 primeres hores de captura de dades de la mota amb descripció *Dormitori Principal*.

En el gràfic dels valors de temperatura es pot observar com la temperatura de la nit i matinada està una mica per sobre dels 26°C. A mida que va avançant el dia la temperatura puja fins els 27°C.

La representació gràfica de la humitat relativa mostra valors constants, al voltant del 64 %, durant la nit i la matinada. A mesura que avança el dia, la humitat relativa es situa al 58%.



Il·lustració 133: *Representació gràfica de la temperatura de la mota "Dormitori principal" amb thethings.iO*



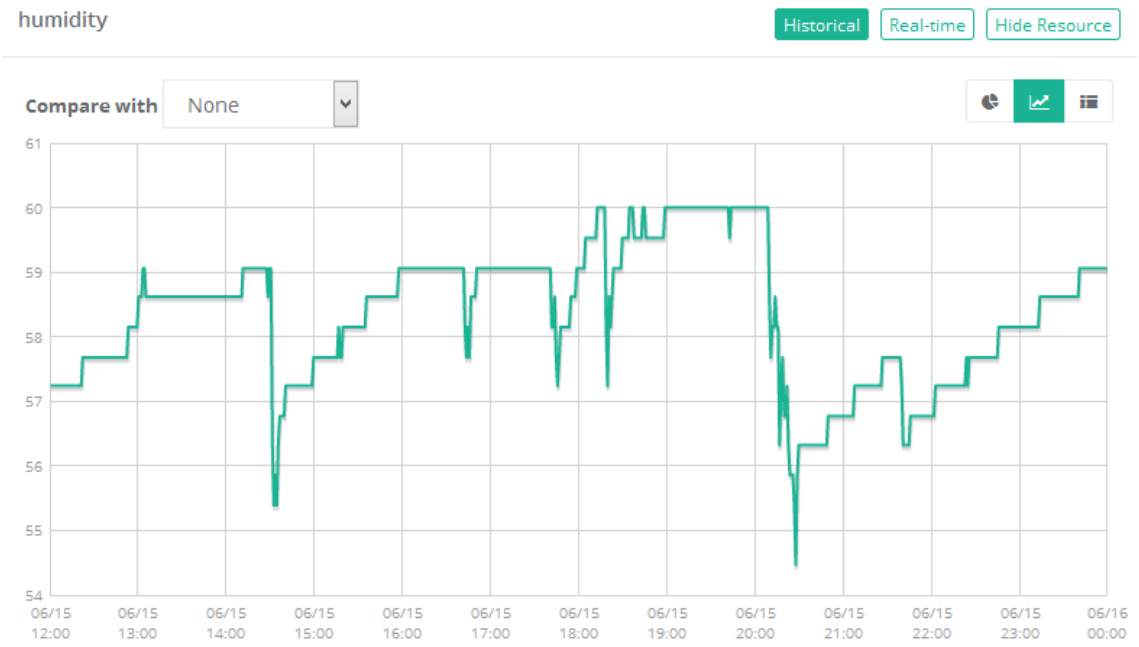
Il·lustració 134: *Representació gràfica de la humitat relativa de la mota "Dormitori principal" amb thethings.iO*

En les següents il·lustracions es poden visualitzar els gràfics de temperatura i humitat relativa de les 12 últimes hores de captura de dades de la mota amb descripció *Menjador*.

Il·lustració 135: *Representació gràfica de la temperatura de la mota "Menjador" amb thethings.iO*

En el gràfic dels valors de temperatura es pot observar com la temperatura a la part central del dia és una mica superior a 27°C i va disminuït quan cau la nit a 26.6°C.

La representació gràfica de la humitat relativa mostra com la humitat relativa s'estabilitza a un valor proper al 59% quan transcorre un període de temps sense ventilació en l'estança.



Il·lustració 136: *Representació gràfica de la humitat relativa de la mota "Menjador" amb thethings.iO*

8 Conclusions

S'han explicat i après el diversos protocols estàndards per IoT distribuïts per capes.

S'han presentat els diversos components de la plataforma de hardware OpenMote: OpenMote-CC2538, OpenBase i OpenBattery. A més, s'han especificat les característiques tècniques més importants per cadascun d'ells.

S'ha après a desenvolupar *firmware* sobre el sistema encastat OpenMote.

S'ha explicat i entès el funcionament de la pila de protocols i OpenOS del projecte OpenWSN, així com l'aplicació OpenVisualizer i la llibreria Python CoAP.

S'ha especificat i comprès el mode de funcionament de la plataforma thethings.iO i de la seva llibreria Node.js CoAP.

S'ha configurat la Raspberry Pi per executar OpenVisualizer de OpenWSN en mode web. A més, s'ha creat una aplicació en llenguatge Python anomenada *openDemo*, una aplicació basada en el *framework* Qt en Python com interfície gràfica d'usuari (GUI) anomenada *openDemoGUI* i dues aplicacions en llenguatge Node.js per interactuar amb la plataforma thethings.iO. Una per emmagatzemar i una altra per llegir les dades anomenades Write.js i Read.js respectivament. Així mateix, disposa d'una base de dades SQLite on s'emmagatzemen els paràmetres del demostrador. La comunicació amb la WSN de OpenWSN es realitza a través de la llibreria Python CoAP de OpenWSN.

Les funcions de l'aplicació *openDemo* són:

- Capturar cada minut les dades de temperatura i humitat.
- Enviar les dades capturades l'aplicació Write.js a través d'un *socket*.
- Controlar el sistema de calefacció a partir de les dades de temperatura i de l'estat del sistema de calefacció.
- Afegir de forma automàtica en la base de dades l'adreça IPv6 de les motes.

L'aplicació Write.js rep les dades de l'aplicació *openDemo* i les envia a la plataforma thethings.iO pel seu emmagatzematge a través de la llibreria Node.js CoAP de thethings.iO.

openDemoGUI permet visualitzar, configurar i emmagatzemar en la base de dades els paràmetres del demostrador. A més, permet visualitzar l'històric de temperatura i humitat relativa de cada node obtenint les dades de la plataforma thethings.iO a través de la comunicació amb un *socket* amb l'aplicació Read.js. Així com, capturar dades de temperatura i humitat de forma instantània de qualsevol mota.

En les motes de la WSN, s'han desenvolupat quatre aplicacions CoAP sobre el firmware de OpenWSN per donar resposta a les funcions del demostrador.

S'ha verificat i validat el correcte funcionament del demostrador desenvolupat.

Per tant, es pot afirmar que **s'ha assolit els objectius** marcats en l'inici del projecte.

Referent a la gestió del projecte, s'han executat les diverses tasques de la planificació temporal del projecte més o menys en el temps marcat. La metodologia emprada ha contribuït a aquest fet i ha permès obtenir una bona qualitat. L'ús de la metodologia àgil per realitzar gairebé tot el desenvolupament de software ha permès estalvi de temps en reprogramar parts del codi. Fet que s'hagués produït en la metodologia en cascada, ja que la falta d'experiència de desenvolupament en l'entorn utilitzat, hauria portat a un disseny erroni de la solució.

8.1 Línies futures

Les possibles línies futures que es deriven del present Treball Final de Màster són:

- El present projecte pot servir com a base per a la construcció de noves aplicacions de IoT amb OpenMote, OpenWSN i thethings.iO.
- Incorporar un sistema expert (SE) pel control del sistema de calefacció amb l'objectiu de millor el confort i reduir els costos de calefacció.
- Accés remot a la interfície gràfica d'usuari (GUI), ja sigui des d'un PC o un dispositiu mòbil com *smartphones* o *tablets*.
- Permetre la possibilitat d'assignar temperatures de consigna per estances de la llar.
- Detecció de nivell baix de bateria en les *motes sensor* i notificar-ho a l'usuari.
- Indicador en la GUI per identificar quan el sistema de calefacció està en funcionament.

- Indicador en la GUI si s'ha produït un error en el procés de monitoratge i control del sistema de calefacció.
- Millores de la usabilitat de la GUI a partir del feedback d'usuaris.

9 Glossari

6LoWPAN: *IPv6 over Low power Wireless Personal Area Networks*

ACK: *Acknowledgement*

ARM: *Avanced RISC Machine*

ASN: *Absolute Slot Number*

BD: Base de dades

BSP: *Board Support Package*

CAP: *Contention Access Period*

CFP: *Contention Free Period*

CoAP: *Constrained Application Procotol*

CON: *Confirmable*

CSMA/CA: *Carrier Sense Multiple Access with Collision Avoidance*

DSSS: *Direct Sequence Spread Spectrum*

EB: *Enhanced Beacon*

EBR: *Enhanced Beacon Requests*

FFD: *Full Function Device*

FSM: *Finite-state machine*

GPIO: *General-purpose input/output*

GTS: *Guaranteed Time Slot*

GUI: *Graphical user interface*

HTTP: *Hypertext Transfer Protocol*

I2C: *Inter-Integrated Circuit*

IDE: *Integrated Development environment*

IE: *Information Element*

IEEE: *Institue of Electrical and Electronics Engineers*

IETF: *Internet Engineering Task Force*

IoT: *Internet of Things*

IP: Internet Protocol

IPC: *inter-process communication*

ISR: *Interrupt Service Routine*

JSON: *JavaScript Object Notation*

JTAG: *Join Test Action Group*

LBR: *Low power and lossy network Border Router*

LE: *Low Energy*

LED: *Light-emitting diode*

M2M: *Machine-to-machine*

MAC: *Medium Access Control*

MTU: *Maximum Transfer Unit*

NON: *Non- Confirmable*

PAN: *Personal Area Network*

RAM: *Random Access Memory*

RFD: *Reduced Function Device*

RPL: *IPv6 Routing Protocol for Low power and Lossy Networks*

SoC: *System on Chip*

SQL: *Structured Query Language*

TCP: *Transmission Control Protocol*

TI: *Texas Instruments*

TSCH: *Time Slotted Channel Hopping*

UDP: *User Datagram Protocol*

URI: *Uniform Resource Identifier*

USB: *Universal Serial Bus*

WPAN: *Wireless Personal Area Networks*

WSN: *Wireless Sensor Network*

10 Bibliografia

[1] *J. Gubbi i altres*. **Internet of Things (IoT): A vision, architectural elements and future directions.** 2013, Future Generation Computer Systems, Vol.29, pp: 1645-1660.

[2] Ugaitz Amozarrain. **Low Power WiFi: A study on power consumption for Internet of Things.** 2015 Tesis Màster, Facultat d'Informàtica de Barcelona (FIB) - UPC. Disponible: http://upcommons.upc.edu/pfc/bitstream/2099.1/25583/1/104901.pdf

[3] **Predicció de la consultora Gartner**. Novembre de 2014. Disponible: http://www.gartner.com/newsroom/id/2905717

[4] Gartner. **Gràfic *Hype Cycle for Emerging Technologies***. Juliol 2014. Disponible: http://www.gartner.com/newsroom/id/2819918

[5] Maria Rita Palattella, Nicola Accettura, Xavier Vilajosana i altres. **Standardized Protocol Stack for the Internet of (Important) Things.** IEEE Communications Surveys and Tutorials**,** IEEE, PP(99):1 –18, 2012. ISSN 1553-877X. doi: 10.1109/SURV.2012.111412.00158, December *2012*.

[6] Thomas Watteyne, Xavier Vilajosana, Branko Kerkez i altres. **OpenWSN: A Standards-Based Low-Power Wireless Development Environment**. Wiley Transactions on Emerging Telecommunications Technologies. Volume: 23: Issue 5. 480–493. Agost 2012.

[7] Lloc web OpenWSN. **Resum estàndard IEEE802.15.4-2006**. Disponible en:

https://openwsn.atlassian.net/wiki/display/OW/IEEE802.15.4-2006

[8] Lloc web Wikipedia de l'estàndard **IEEE802.15.4**. Disponible en línea:

http://en.wikipedia.org/wiki/IEEE_802.15.4

[9] Víctor Alcázar. **Estudio de las prestaciones de IEEE 802.15.4e**. 2013 Treball Final de Grau. Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels – UPC. Disponible en:

http://upcommons.upc.edu/pfc/handle/2099.1/19708

[10] Sergio Gonzalo. **Implementación del Mecanismo de Acceso al Medio (MAC) IEEE 802.15.4e CSL (Coordinated Sampled Listening) sobre OpenWSN y Plataforma OpenMote**. Gener 2015 Treball Final de Màster. Màster Universitari d'Enginyeria de Telecomunicació – UOC. Disponible en:

http://hdl.handle.net/10609/40043

[11] Lloc web OpenWSN. **Resum estàndard IEEE802.15.4e**. Disponible en:

https://openwsn.atlassian.net/wiki/display/OW/IEEE802.15.4e

[12] Lloc web OpenMote Technologes. **Resum estàndard IEEE802.15.4e**. Disponible en: http://www.openmote.com/standards/ieee-802154e.html

[13] Xavier Vilajosana, Pere Tuset-Peiro i altres. **Standardized Low-Power Wireless Communication Technologies for Distributed Sensing Applications**. Sensors 2014, Vol.14, pp.2663-2682, Febrer 2014.

[14] Lloc web OpenMote Technologes. **Resum estàndard IETF 6LoWPAN**. Disponible en: http://www.openmote.com/standards/ietf-6lowpan.html

[15] Lloc web OpenWSN. **Resum estàndard IETF 6LoWPAN**. Disponible en:

https://openwsn.atlassian.net/wiki/display/OW/6LoWPAN

[16] Zach Shelby i Carsten Bormann. **6LoWPAN: The Wireless Embedded Internet**. John Willey & Sons, Inc., Novembre 2009, ISBN: 978-0-470-74799-5, Part 4: Addressing, etc. & a network example, Figure 1.10. Disponible:

http://www.embedded.com/print/4217261

[17] René Hummen, Jens Hiller i altres. **6LoWPAN Fragmentation Attacks and Mitigation Mechanisms.** Proceedings of the sixth ACM Conference on Security and privacy in Wireless and Mobile Networks (WiSec), pp:55-66, ISBN: 978-1-4503-1998-0 doi: 10.1145/2462096.2462107, 2013.

[18] Lloc web OpenMote Technologes. **Resum estàndard IETF RPL**. Disponible en: http://www.openmote.com/standards/ietf-rpl.html

[19] Lloc web OpenWSN. **Resum estàndard IETF RPL**. Disponible en:

https://openwsn.atlassian.net/wiki/display/OW/RPL

[20] Mario Pareja. **L'Internet de les Coses: arquitectures, protocols i aplicacions**. Gener 2015 Treball Final de Màster. Màster Universitari d'Enginyeria de Telecomunicació – UOC. Disponible en: http://hdl.handle.net/10609/40041

[21] Lloc web CoAP Technlogy. **Definició protocol CoAP**. Disponible en: http://coap.technology/

[22] Lloc web OpenMote Technologes. **Resum estàndard IETF CoAP**. Disponible en: http://www.openmote.com/standards/ietf-coap.html

[23] Lloc web **RFC 7252: The Constrained Application Protocol (CoAP). Disponible en:**

http://tools.ietf.org/html/rfc7252

[24] FARNOOSH FAROKHMANESH. **ANALYZING AND EVALUATING NETWORK PROTOCOLS IN IoT.** 2014 Tesis Màster, Facultat d'Informàtica de Barcelona (FIB) - UPC. Disponible: http://upcommons.upc.edu/pfc/bitstream/2099.1/24235/1/101097.pdf

[25] Lloc web TinyOS. **Definició TinyOS**. Disponible: http://www.tinyos.net/

[26] Lloc web TinyOS. **Hardware compatible amb TinyOS**. Disponible: http://tinyos.stanford.edu/tinyos-wiki/index.php/Platform_Hardware

[27] Lloc web Contiki. **Hardware compatible amb Contiki**. Disponible: http://www.contiki-os.org/hardware.htm

[28] Lloc web RIOT. **Definició i hardware compatible amb RIOT**. Disponible: http://www.riot-os.org/#home

[29] Lloc web OpenWSN. **Presentació projecte OpenWSN**. Disponible: https://openwsn.atlassian.net/wiki/display/OW/Home

[30] Lloc web OpenWSN. **Hardware compatible amb OpenWSN**. Disponible: https://openwsn.atlassian.net/wiki/display/OW/Hardware

[31] Lloc web OpenWSN. **Pila de protocols de OpenWSN**. Disponible: https://openwsn.atlassian.net/wiki/display/OW/Protocol+Stack

[32] Lloc web OpenWSN. **OpenVisualizer**. Disponible: https://openwsn.atlassian.net/wiki/display/OW/OpenVisualizer

[33] Watteyne Thomas y Adjih Cedric. **OpenWSN: Implementing the Internet Of Important Things**. Workshop Internet of Things / Equipex FIT IoT-LAB. Montbonnot, France. 2014. Disponible en: https://www.iot-lab.info/wp-content/uploads/2014/11/141106_openwsn_iotlab_public.pdf

[34] Lloc web OpenWSN. **Màquina d'estats de la llibreria Python de CoAP**. Disponible: https://openwsn.atlassian.net/wiki/display/OW/CoAP

[35] Lloc web OpenMote Technologes. **Hardware plataforma OpenMote**. Disponible en: http://www.openmote.com/hardware.html

[36] Lloc web OpenMote Technologes. **Especificacions OpenMote-CC2538**. Disponible en: http://www.openmote.com/hardware/openmote-cc2538-en.html

[37] Lloc web OpenMote Technologes. **Especificacions OpenBase**. Disponible en: http://www.openmote.com/hardware/openbase.html

[38] Lloc web OpenMote Technologes. **Especificacions OpenBattery**. Disponible en: http://www.openmote.com/hardware/openbattery.html

[39] Lloc web Wikipedia Raspberry Pi. **Definició Raspberry Pi**. Disponible en: http://en.wikipedia.org/wiki/Raspberry_Pi

[40] Lloc web Raspberry Pi. **Productes Raspberry Pi**. Disponible en: http://www.raspberrypi.org/products/

[41] Lloc web botiga RS Components Ltd. **Imatge Raspberry Pi 1 Model A+.** Disponible: http://uk.rs-online.com/web/p/processor-microcontroller-development-kits/8332699/

[42] Lloc web botiga RS Components Ltd. **Imatge Raspberry Pi 1 Model B+.** Disponible: http://uk.rs-online.com/web/p/processor-microcontroller-development-kits/8111284/

[43] Lloc web botiga RS Components Ltd. **Imatge Raspberry Pi 2 Model B.** Disponible: http://uk.rs-online.com/web/p/processor-microcontroller-development-kits/832-6274/

[44] Lloc web thethings.iO. **Plataforma *cloud* IoT thethings.iO**. Disponible en: https://thethings.iO/

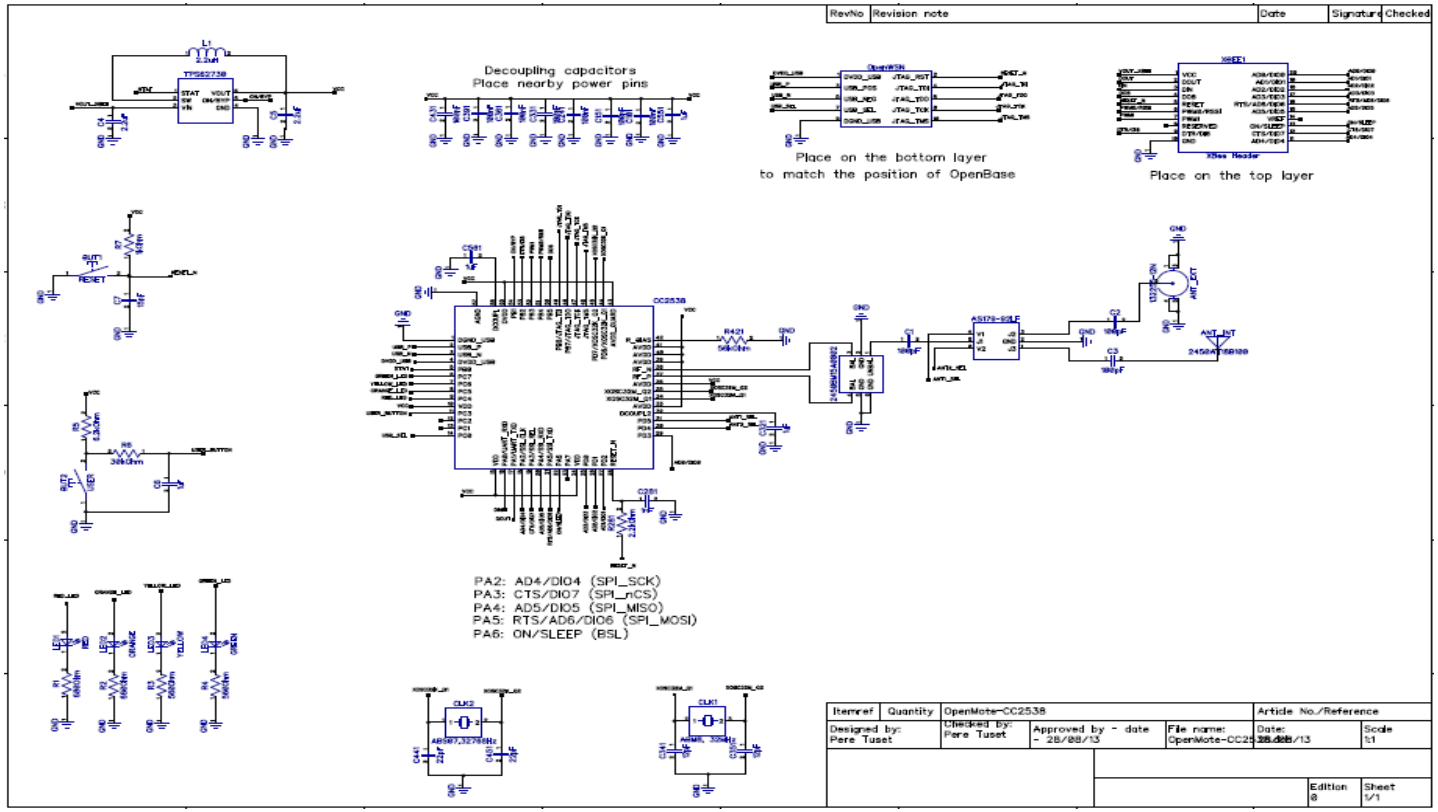
[45] Lloc web thethings.iO. **thethings.iO CoAP**. Disponible en: https://developers.thethings.iO/makers-coap.html

[46] Lloc web thethings.iO. ***Quickstart* thethings.iO**. Disponible en: https://developers.thethings.iO/

[47] Lloc web GitHub. Llibreria Node.js CoAP de thethings.iO. Disponible en: https://github.com/theThings/thethingsio-coap-node

11 Annexos

11.1 Esquemàtic placa OpenMote-CC2538



11.2 Esquemàtic placa OpenBase