Master Universitario Ingeniería de telecomunicación

TELEMÁTICA

Implementación de un sistema de gestión “In Cloud” de redes WSN

Autor: Francisco Jesús Villaseñor García

Consultor: José López Vicario

Universitat Oberta de Catalunya

ÍNDICE

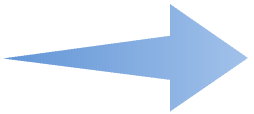
(1) Problema a resolver

- **Redes de sensores**

Cada vez más habituales

Ubicuas

Múltiples nodos



- **Problemas**

Distribución geográfica

Dificultad de acceso

- **Retos**

Reducción de consumo (autonomía)

Movilidad

- **Solución: gestión remota**

Acceso a la información

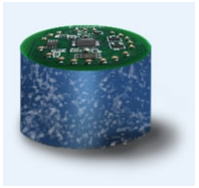
Detección de problemas

Programar mantenimientos

Cambios de funcionamiento centralizados

(1) Problema a resolver: Otras soluciones analizadas

Las más conocidas



TinyDB (Berkeley)

- OpenSource
- No tiene una interfaz sino un mecanismo para generarla

Crossbow MoteWorks (Comercial)

Inconvenientes

Ligadas a un sistema operativo TinyOS

No usan estándares

Intento de estandarización

OGC® Making location count. Sensor Web Enablement (SWE)

A desarrollar

Servicios SOAP: pesados no es la tendencia actual

(1) Problema a resolver: Solución desarrollada

Interfaz web

- Ultimas tendencias (Bootstrap, AngularJS, Single Page Application)
- Estándares HTML, HTTP-REST, AJAX, JSON

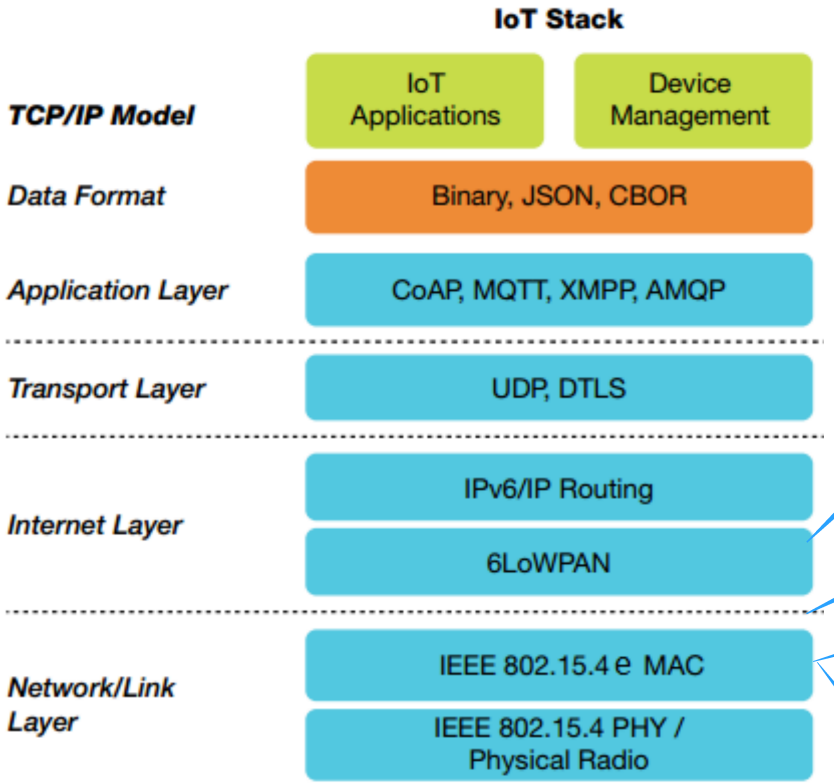
Servidor web

- Basado en Flask (PYTHON)
- Desplegable en la nube
- Preparado para desplegar sobre Raspberry PI

Mota

- Protocólos estandard IoT
- Interacción CoAP con la capa del servidor
- Desplegada en OpenMote-CC2538 y OpenWSN

(2) Estándares IoT



6LoWPAN
Adaptación a IPv6
Apilamiento/Compresión de cabeceras
Fragmentación

6top
Adaptar TSCH a 6LoWPAN

Coordinated Sampled Listening (CSL)
Low Energy (LE)
Muestreos periódicos del canal

Time Slotted Channel Hopping (TSCH)
Entornos industriales
Consumo de energía reducido
Alta robustez frente a interferencias

(2) Estándares IoT: Constrained Application Protocol

CoAP RFC7252 Junio 2014

Similar a HTTP sobre UDP: ahorro de consumo

Transacción CoAP (Token incluido en la cabecera)
- CON (confirmado) con piggybacking o sin piggybacking (respuesta en otro mensaje CON)
- NON (no confirmado)

Semántica REST: subconjunto de las operaciones HTTP
- GET (consulta), seguro e idempotente
- PUT (modificación/creación) y DELETE (borrado), no seguros e idempotentes
- POST (creación/modificación/borrado) no seguro y no idempotente

Códigos de respuesta: subconjunto de los códigos HTTP
2.xx correcto, 4.xx error cliente, 5.xx error servidor

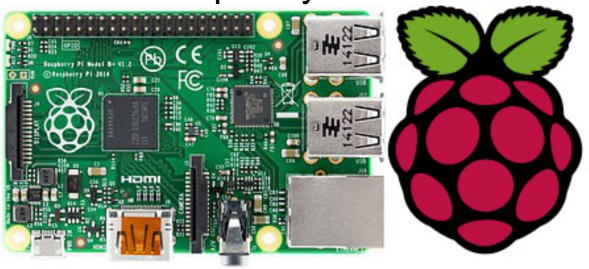
URIs CoAP: similares a las HTTP: coap[s]://host[:puerto]/path[?query].
Puertos por defecto 5683 y 5684

Compatible con HTTP: proxy CoAP-HTTP directo, proxy HTTP-CoAP sencillo

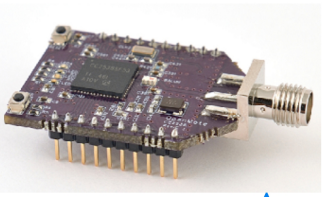
(3) Arquitectura

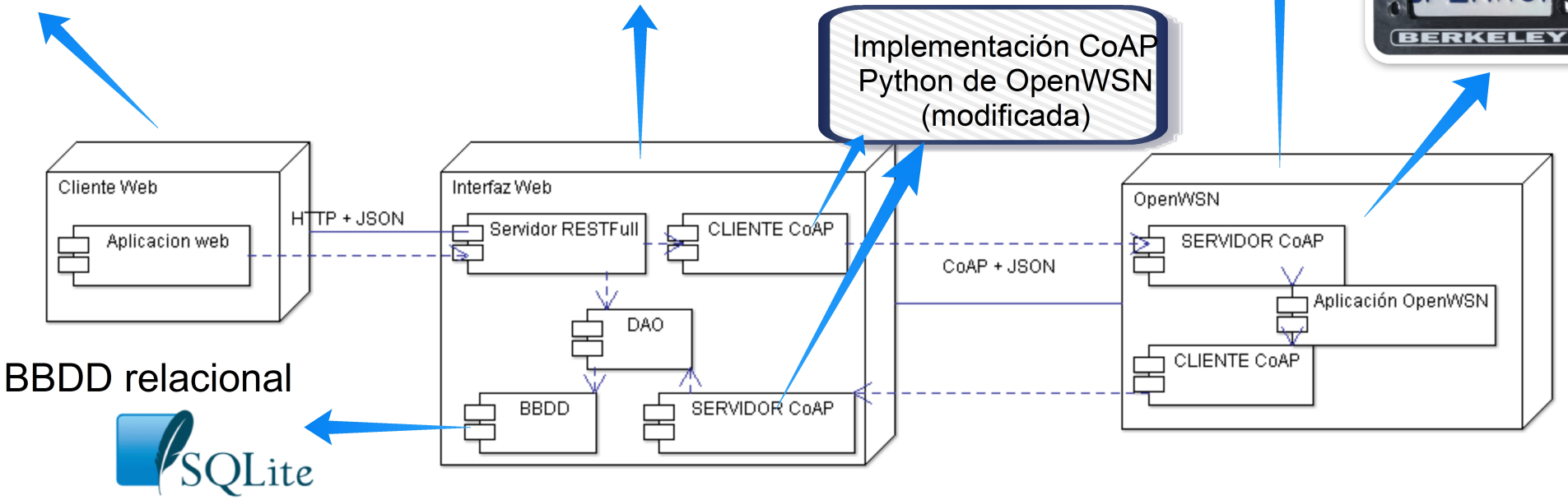
Navegador Web
(p.e. Chrome)

Servidor
Raspberry PI



OpenMote-CC2538



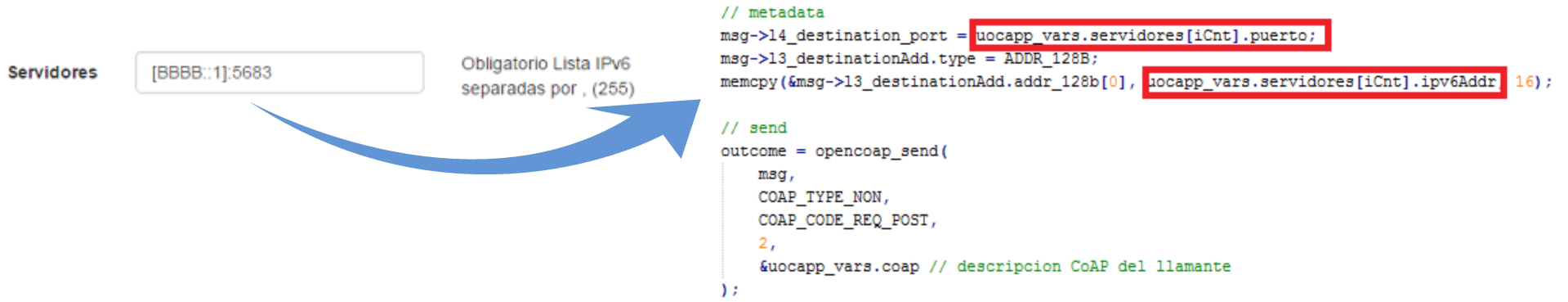


BBDD relacional

(3) Arquitectura: Parámetros en la mota

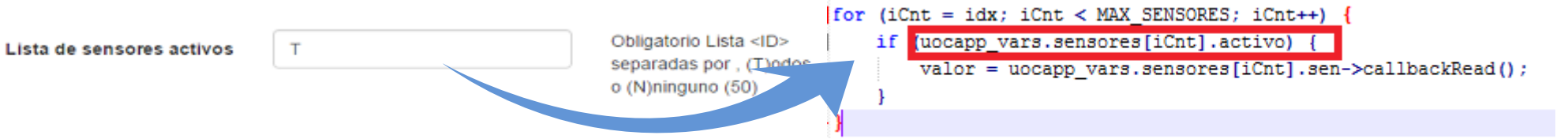
Afectan al comportamiento de la mota

1 Servidores [lista de ipv6:puerto]: servidores a los que se desea cargar la información



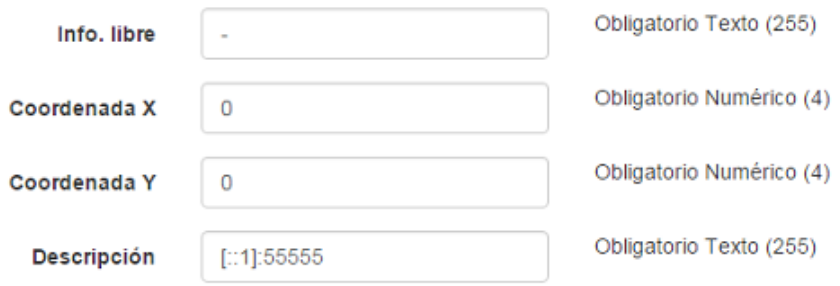
2 Tiempo entre muestras: tiempo entre envíos de información desde la mota

3 Lista de sensores activos



(3) Arquitectura: Parámetros y Descubrimiento

Parámetros que afectan a la interfaz web



- Información libre: anotaciones, etc.
- Descripción de la mota: listados
- X,Y: localización de la mota en el mapa.

Parámetros de estado

Sensores que informan del estado de la mota

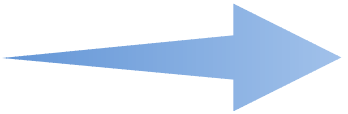
Se define un valor crítico y una comparación para generar una alerta al usuario

- Estado batería: carga de la batería de la mota en ese momento.
- Temperatura CPU: temperatura del microcontrolador.

Descubrimiento

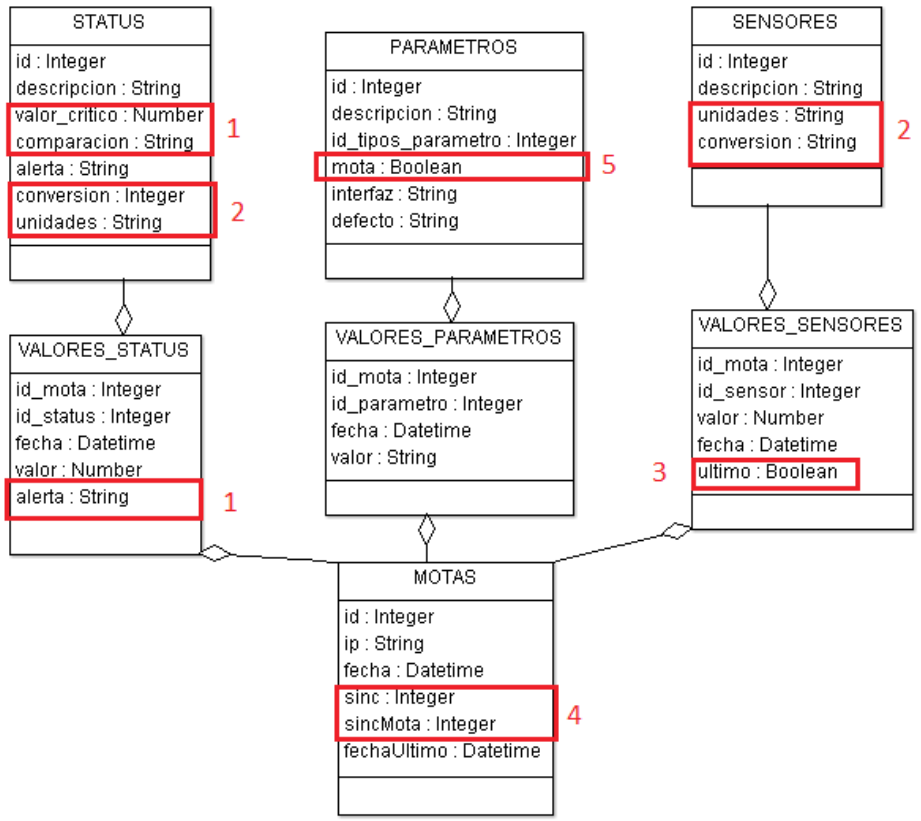
Cómo la Interfaz Web conoce las motas que forman parte del sistema.

Responsabilidad de la mota: tiene definidos por defecto un valor para los parámetros



La mota envía información a la Interfaz Web desde el momento que se integra en la red WSN

(3) Arquitectura: Modelo de datos (resumen)



1/ Si el valor del sensor de estado cumple con la comparación establecida se genera una alerta

2/ Las unidades del valor del sensor y la función de conversión de la lectura del sensor (ADC, entero de 16 bits) al valor de trabajo (ºC, %HR, etc.).

3/ Indicador de que es el último valor del sensor recibido

4/ Indicadores de sincronización

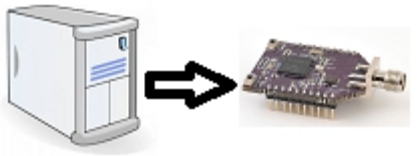
- sinc: indicador enviado a la mota
- sincMota: indicador que informa la mota que ha recibido

si son distintos es que la mota no ha recibido la actualización de los parámetros.

5/ Indicador de que el parámetro afecta al funcionamiento de la mota

(4) Mensajes CoAP

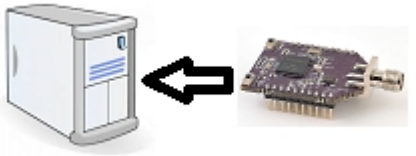
Interfaz Web a la Mota: POST /uocapp



{"s":{"1":…, "2":...}, "e":{"1":…}, "f":'...'}
- s: información de sensores
- e: información de estado
- f: indicador de sincronización

NO CONFIRMADO (NON)
- Reducir la sobrecarga de la red
- Información es periódica

Mota a la Interfaz Web: POST /uocapp



{"0":..., "1":…, "2":...}
- "1" ids parámetros configuración
- "0" corresponde a sincronización

CONFIRMADO (CON)
- Información crítica
- Esporádico (no sobrecarga la red WSN)

Limitaciones OpenWSN

OpenWSN soporta paquetes de 127 bytes sin fragmentación
División lógica de los mensajes anteriores para respetar con estos límites.

(5) Extensión de OpenWSN

Implementación CoAP Python

El recurso encargado de atender la petición recibe la información del emisor.

Así se puede identificar la mota en la red WSN por su IP:puerto

coap.py

```
    )
elif message['code']==d.METHOD_POST:
    (respCode,respOptions,respPayload) = resource.POST(
        options=message['options'],
        payload=message['payload'],
        sender=sender
    )
elif message['code']==d.METHOD_PUT:
```

coapResource.py

```
def PUT(self,options=[],payload=None,sender=None):
    raise e.coapRcMethodNotAllowed()

def POST(self,options=[],payload=None,sender=None):
    raise e.coapRcMethodNotAllowed()
```

Lectura del estado de la batería

Parámetro crítico en una red dispersa

Se utilizan las facilidades que proporciona el SoC CC2538

adc_sensors.c

```
/**
   \brief Read rough data from Battery sensor
   \param[out] ui16Dummy rough data.
*/
uint16_t adc_sens_read_vdd(void) {
   uint16_t ui16Dummy;

   SOCADCSingleStart(SOCADC_VDD);
   while(!SOCADCEndOfConversionGet());
   ui16Dummy = SOCADCDataGet() >> SOCADC_12_BIT_RSHIFT;
   return ui16Dummy;
}
```

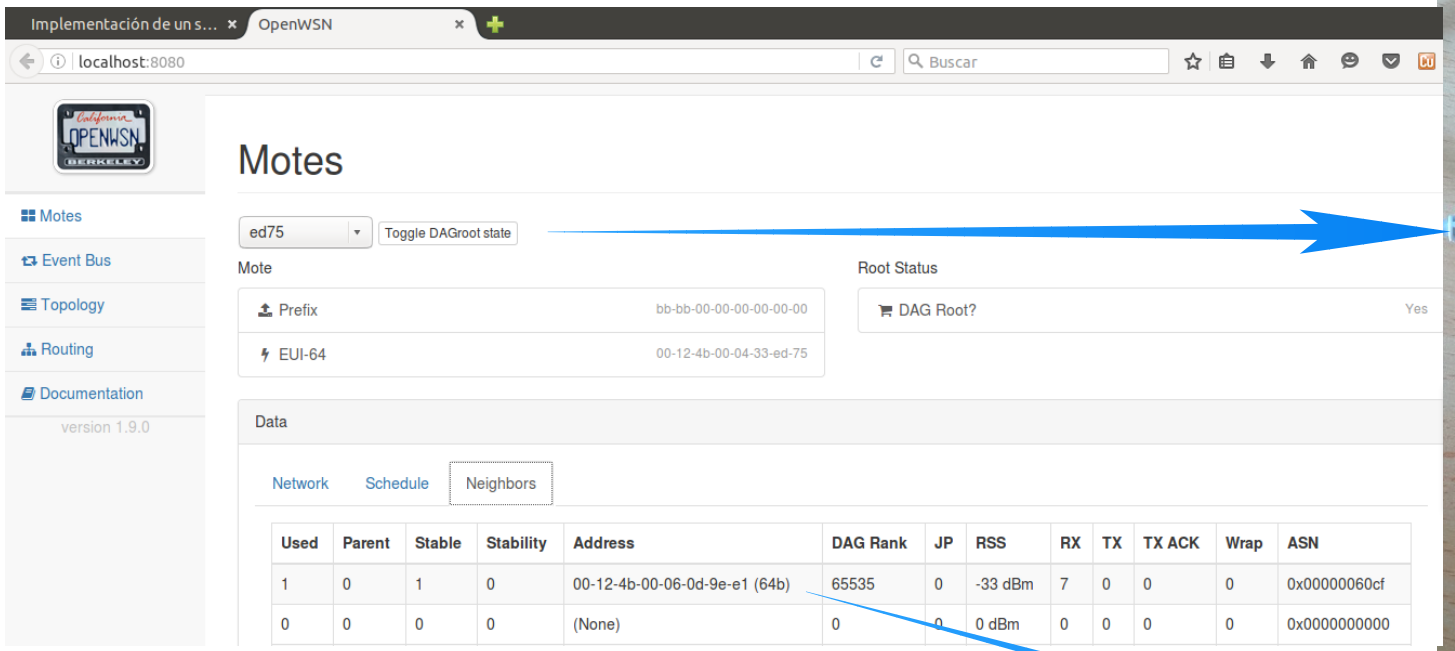
sensors.c

```
callbackRead_cbt sensors_getCallbackRead(uint8_t sensorType) {

  switch (sensorType) {
    case SENSOR_TEMPERATURE:
      return &sht21_read_temperature;
      ...
    case SENSOR_ADCTEMPERATURE:
      return &adc_sens_read_temperature;
    case SENSOR_ADCVDD:
      return &adc_sens_read_vdd;
    default:
      return NULL;
  }
```

(6) Funcionamiento de la solución

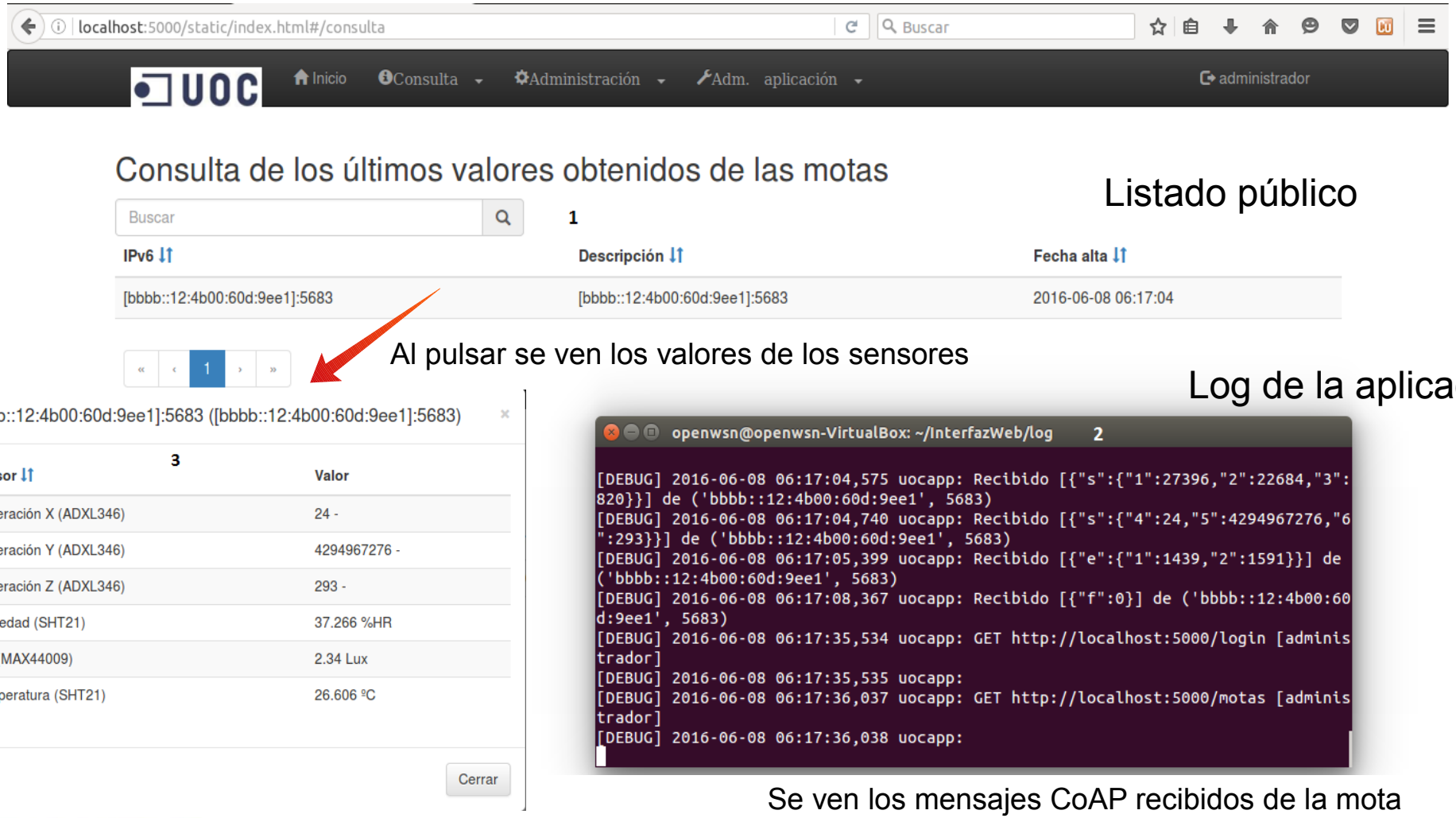
Conexión red WSN



OpenVisualizer

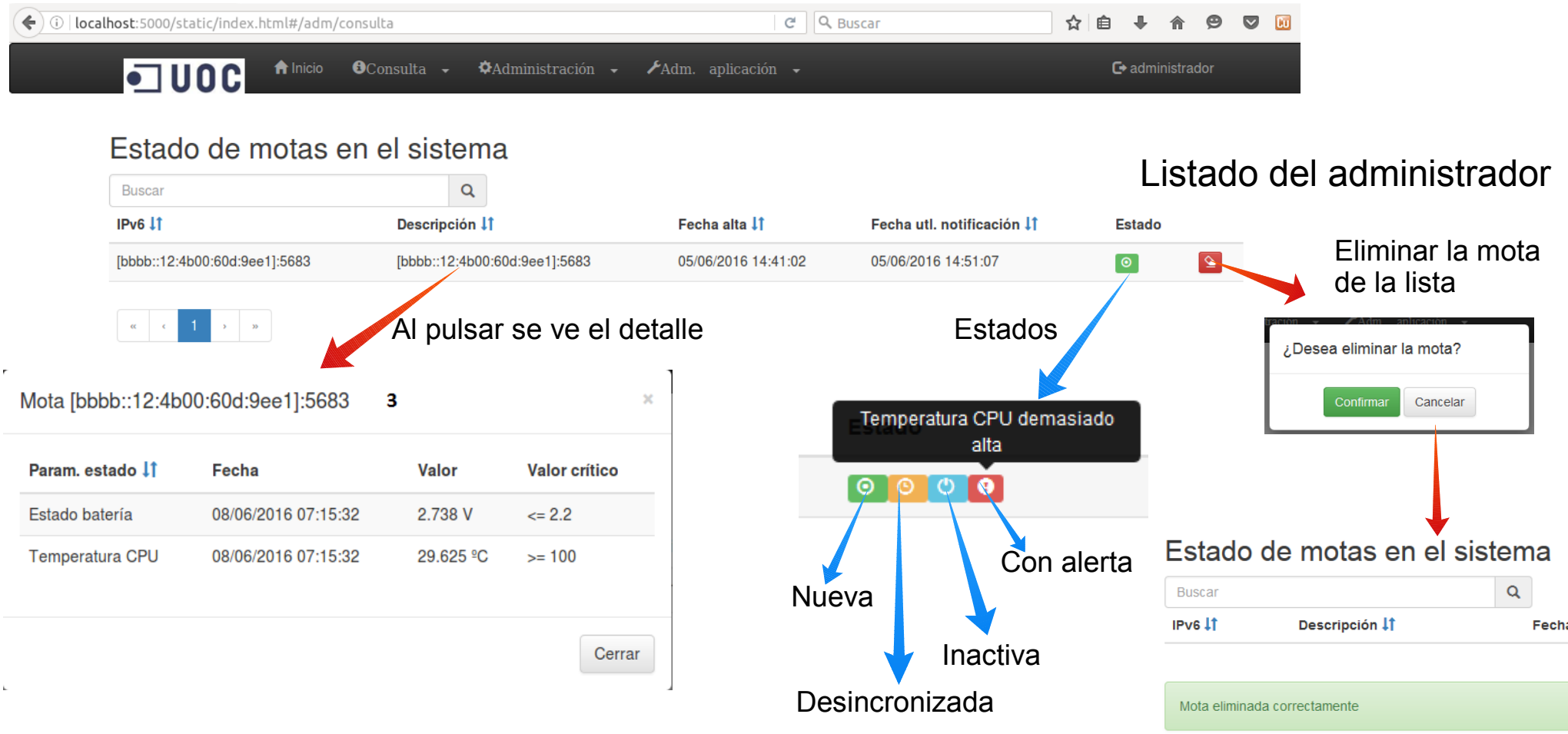
(6) Funcionamiento de la solución

Consultas



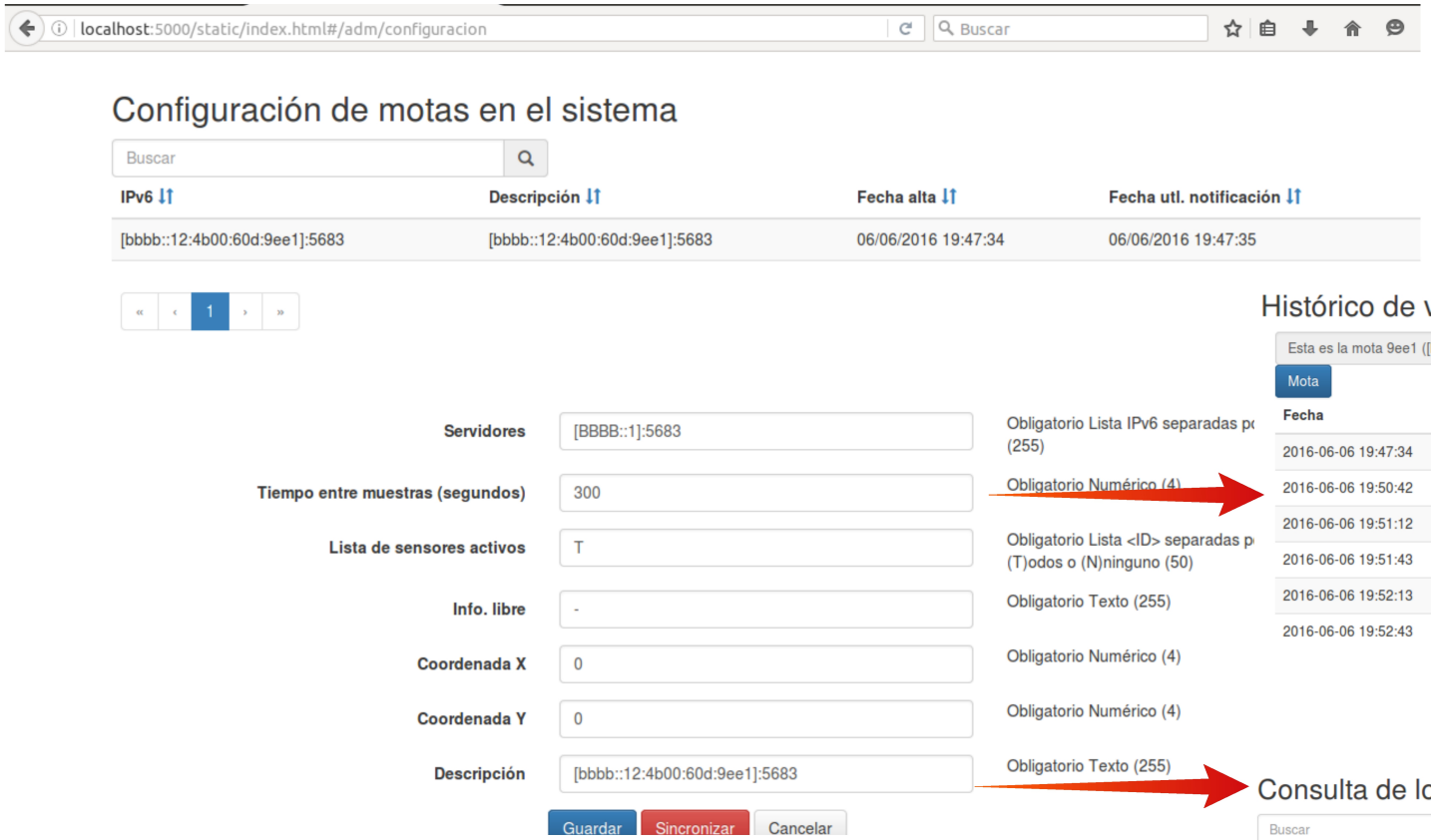
(6) Funcionamiento de la solución

Administrador (I)



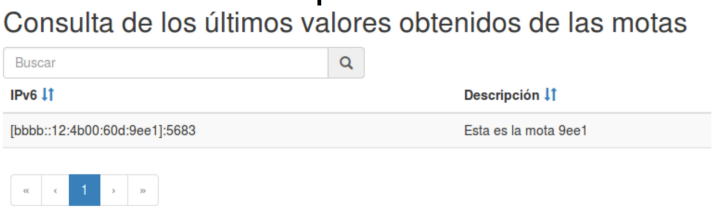
(5) Funcionamiento de la solución

Administración (II)



Ejemplo modificación tiempo entre muestras a 30 segundos

Ejemplo modificación descripción



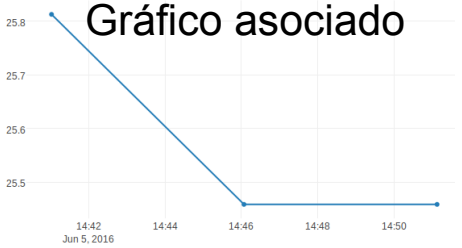
Cambios en la mota seleccionada

Cambios en todas las motas; sólo los 3 primeros parámetros

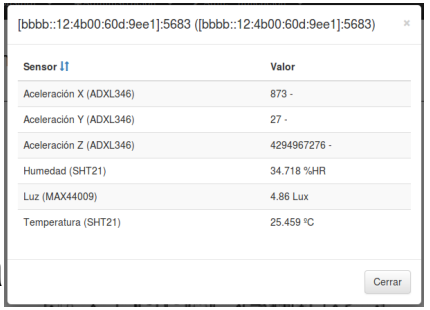
(6) Funcionamiento de la solución

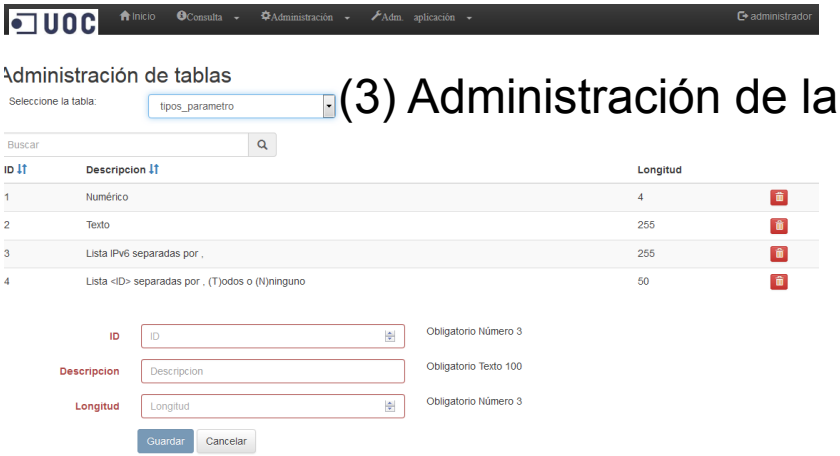
Otras funcionalidades

(1) Histórico de valores



(2) Posición sobre un mapa





(3) Administración de la aplicación

(7) Conclusiones y líneas futuras

Origen:

- Las redes inalámbricas de sensores WSN son cada vez más habituales.
- Deseable una aplicación de gestión remota de la red.
- IEEE 802.15.4 estándar inalámbrico de baja tasa y bajo consumo para aplicaciones IoT.
- Soluciones tradicionales limitadas a un S.O. sin utilizar estándares.

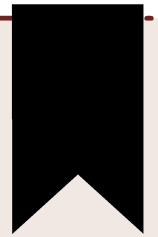
Objetivo:

- Desarrollar solución basada en estándares IoT como CoAP.
- Uso de configuración a nivel de servidor: cualquier mota sobre cualquier S.O. puede integrarse en la herramienta de gestión
- Seguir tendencias actuales de aplicaciones web (SPA, comunicaciones REST)

Líneas futuras:

- Modificación dinámica del software (programación over-the-air OTA)
- Extensión a otras motas y otros S.O. (Contiki, etc.).

Implementación de un sistema de gestión “In Cloud” de redes WSN



Agradecimientos

A José López Vicario, consultor del proyecto, por su colaboración y ayuda a lo largo de todo el desarrollo del mismo.

A Pere Tuset-Peiró, por su ayuda en lo referente a OpenMote

Francisco Jesús Villaseñor García <fvillasenor@uoc.edu>

UOC
Universitat Oberta de Catalunya