Implementación del protocolo MQTT-S sobre IEEE 802.15.4e en plataformas OpenMOTE

Javier López Molinero
Máster Universitario en Ingeniería de Telecomunicación UOC-URL
Telemática

Jose López Vicario
Xavier Vilajosana Guillen

17/06/2018

Licencias alternativas (elegir alguna de las siguientes y sustituir la de la página anterior)

A) Creative Commons:

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-CompartirIgual 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-SinObraDerivada 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-CompartirIgual 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento 3.0 España de Creative Commons

B) GNU Free Documentation License (GNU FDL)

Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled "GNU Free Documentation License".

C) Copyright

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Implementación del protocolo MQTT-S sobre IEEE 802.15.4e en plataformas OpenMOTE*
Nombre del autor:	*Javier López Molinero*
Nombre del consultor/a:	*José López Vicario*
Nombre del PRA:	*Xavier Vilajosana Guillen*
Fecha de entrega (mm/aaaa):	06/2018
Titulación:	*Máster de Ingeniería de Telecomunicación (UOC, URL)*
Área del Trabajo Final:	*Telemática*
Idioma del trabajo:	*Español*
Palabras clave	*IoT, MQTT-S, IEEE 802.15.4e*

Resumen del Trabajo:

El proyecto consiste en realizar una prueba de concepto de comunicaciones IoT a través de IEEE 802.15.4e que encapsula los paquetes TCP / IP con el protocolo MQTT-S, esto se puede hacer con dispositivos OpenMote, una Raspberry Pi y un sensor específico.

Los datos del sensor se transmiten con un OpenMote hacia otro OpenMote conectado a un dispositivo Raspberry Pi con un programa instalado que es el responsable de cargar datos a la Base de datos en la nube. Estos datos se muestran con un Grafana u otra herramienta similar. Este proyecto servirá para estudiar la viabilidad del protocolo MQTT-S para comunicaciones IoT.

Todo esto tiene como principal objetivo la expansión de las bibliotecas de desarrollo del sistema operativo para plataformas IoT y OpenMote denominadas OpenWSN, agregando la capacidad de desarrollo con el protocolo MQTT-S fácilmente con este sistema operativo.

A continuación, adjunto un bosquejo explicativo del proyecto:

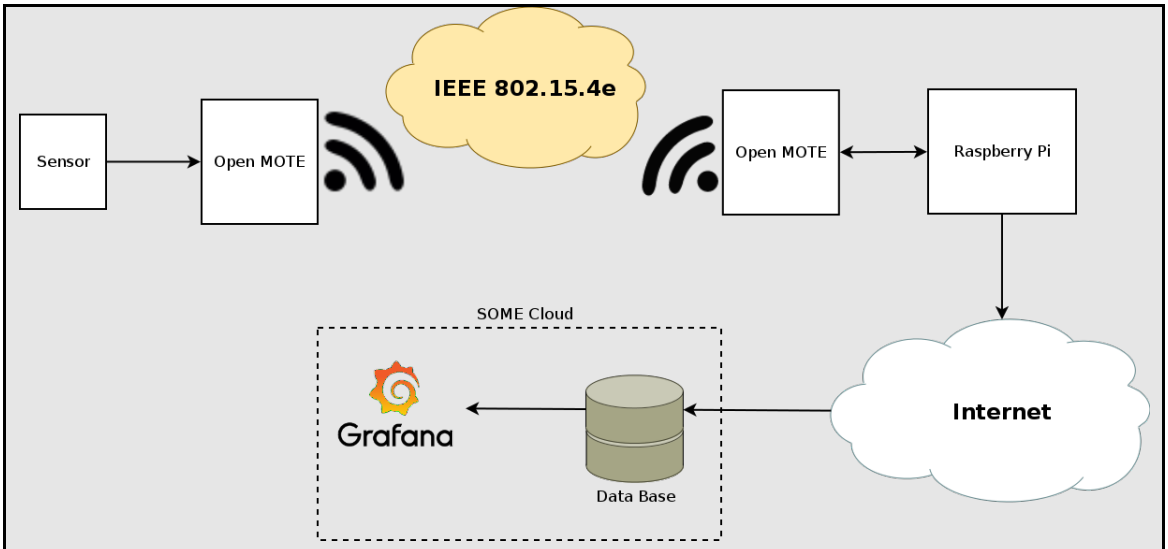


Ilustración 1: Resumen gráfico del proyecto

Abstract:

The project consists of conducting a proof of concept of IoT communications through IEEE 802.15.4e encapsulating the TCP / IP packets with the MQTT-S protocol, this can be done with OpenMote devices, a Raspberry Pi and a specific sensor.

The sensor data are transmitted with an OpenMote toward other OpenMote connected to one device Raspberry Pi with a program installed that is the responsible to upload data to the Database in the cloud. These data are shown with a Grafana or another tool simliar. This project will serve to study the feasibility of the MQTT-S protocol for IoT communications.

The main objective the expansion of the development libraries of the operating system for IoT and OpenMote platforms called OpenWSN, adding the development capacity with the MQTT-S protocol easily with this operating system.

Below, I attach an explanatory outline of the project:

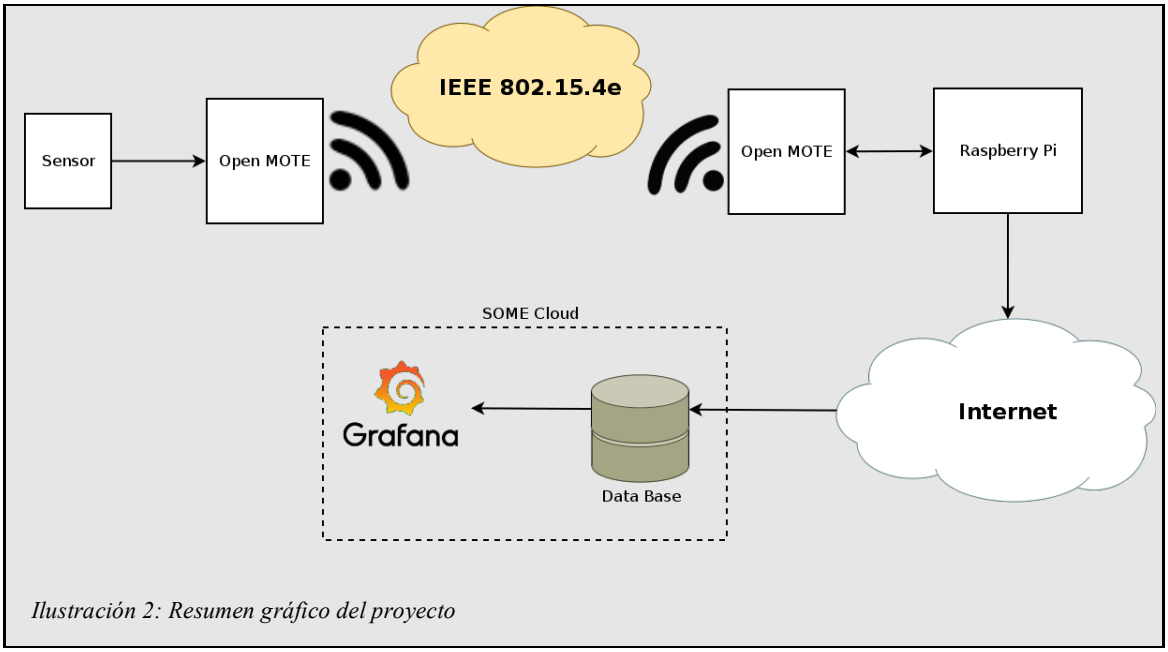


Ilustración 2: Resumen gráfico del proyecto

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

Este es un proyecto orientado al mundo de "Internet de las cosas", donde existen multitud de nuevos protocolos de comunicaciones de mensajes para dispositivos con recursos muy limitados, el proyecto va a tratar de la implementación del protocolo MQTT-S en dispositivos OpenMOTE que utilizan el estándar de comunicaciones IEEE 802.15.4e.

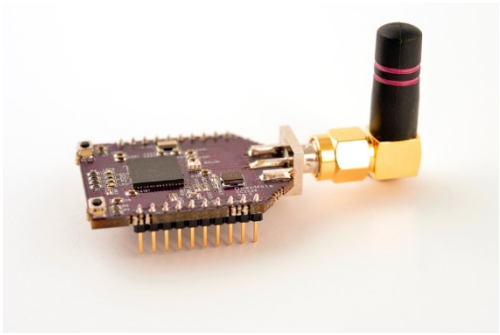


Ilustración 3: Dispositivo OpenMOTE

Este protocolo de comunicaciones tiene las siguientes capacidades:

- Gran popularidad en aplicaciones móviles.
- Protocolo simple para dispositivos con pocos recursos.
- Protocolo binario que usa Publish-Suscribe.
- Es bajo Footprint, convirtiéndole en ideal para IoT.
- Los brokers de MQTT soportan varios miles de conexiones concurrentes.
- Ofrecen 3 calidades de servicio:
    - Fire-and-forget (no confiable)
    - "at least once" que te asegura que los mensajes se envían una vez.
    - "exactly once"

Para poder integrar el protocolo, tenemos que modificar el sistema OpenWSN para que se puedan utilizar de manera sencilla todas las librerías de desarrollo del protocolo MQTT-S y poder realizar una prueba de concepto de manera sencilla. Esto es una necesidad relevante para poder ofrecer un entorno ágil a los desarrolladores de IoT con este sistema operativo muy amigable a la hora de construir despliegues de sensores.

Esta prueba de concepto consiste en enviar la información obtenida en un sensor a través de un dispositivo OpenMOTE hasta otro dispositivo OpenMOTE mediante el protocolo MQTT-S, siendo traducido por una Raspberry Pi para ser subido a una base de datos en la nube y posteriormente mostrado por un Grafana.

1.2 Objetivos del Trabajo

Los objetivos principales del proyecto son los siguientes:

- Desarrollo de Librerías de trabajo para OpenWSN
- Desarrollo de una prueba de concepto para el protocolo MQTT-S para dispositivos OpenMOTE
- Análisis de las prestaciones de MQTT-S

1.3 Enfoque y método seguido

Como ya se ha indicado es necesario hacer una prueba de concepto centrada en el protocolo MQTT-S, por lo tanto, los pasos que se van a desarrollar en este proyecto son las siguientes:

1. Realizar un estudio de los dispositivos que se van a utilizar para establecer un diseño adecuado de la prueba de concepto.
2. Construir el entorno básico para desarrollarlo.
3. Un vez establecido el entorno básico, es necesario desarrollar varias aplicaciones que realicen las siguientes funciones:
    a. Aplicación 1: Instalada en el primer OpenMote. Se ocupará de obtener los datos del sensor y enviarlos al segundo OpenMOTE o Border router.
    b. Aplicación 2: Instalada en el segundo OpenMOTE que será la receptora de los datos y los procesará para que la siguiente aplicación pueda remitirlos a la base de datos.
    c. Aplicación 3: Instalada en una Raspberry Pi, se encargará de enviar los datos procesados a una base de datos en la nube.
    d. Configuración del Grafana: Hay que configurar la herramienta Grafana para que se muestren los datos recolectados de manera amigable y entendible.
4. Completadas las tareas descritas anteriormente, es el momento de introducir el protocolo MQTT-S en el entorno.

Esta realización sucesiva de tareas sigue el concepto de desarrollo incremental de los proyectos, facilitando la consecución de objetivos de manera más sencilla, ofreciendo lo siguiente:

- Obtención de una base estable, que la inestabilidad del sistema no suponga un mal funcionamiento, con errores de implementación de protocolo o limitaciones del protocolo.
- Facilita el entendimiento de la prueba de concepto.
- Existen muy pocas probabilidades de riesgo en el sistema. Aunque se pueden encontrar dificultades en algunos avances.

1.4 Planificación del Trabajo

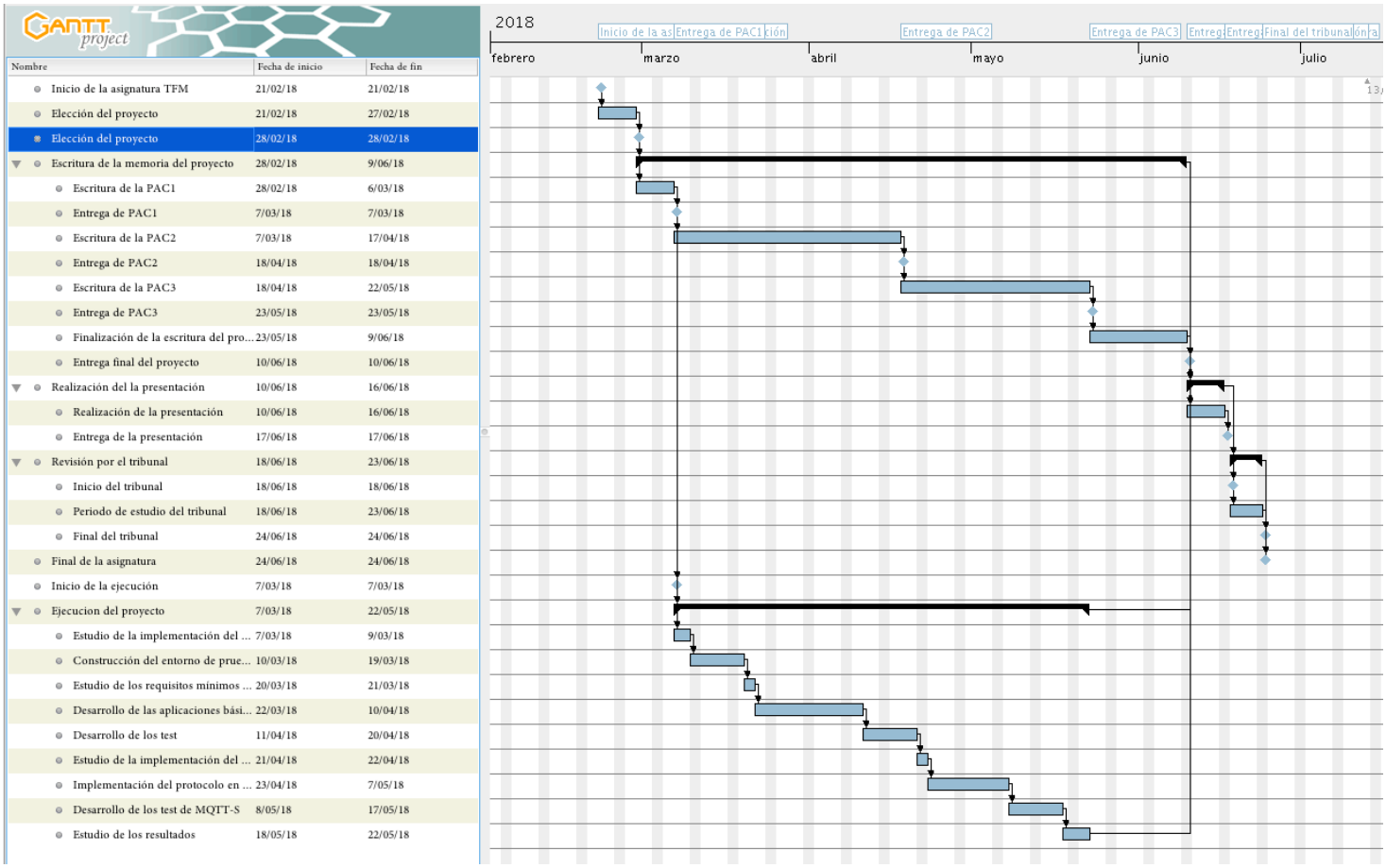
Los recursos necesarios para la realización de este proyecto son los siguientes:

- Dos dispositivos OpenMOTE.
- Una Raspberry Pi con tarjeta de memoria.
- Cables y protoboard.
- Sensor.
- Un PC personal.
- Desarrollador.
- Base de datos y un Grafana implementados en la nube.

Las tareas a realizar son las siguientes:

- Estudio de la implementación del entorno.
- Construcción del entorno de pruebas.
- Estudio de los requisitos mínimos de las aplicaciones y planificación de los test necesarios.
- Desarrollo de las aplicaciones básicas para el entorno de desarrollo.
- Desarrollo de las pruebas.
- Estudio de la implementación del protocolo MQTT-S y definición de los test funcionamiento.
- Implementación del protocolo en las aplicaciones básicas.
- Desarrollo de los test de MQTT-S.
- Estudio de los resultados.
- Escritura de la memoria del proyecto.

Con esta enumeración de tareas, se procede a definir el siguiente diagrama Gantt que ayudara en la planificación del desarrollo del proyecto:



Nombre	Fecha de inicio	Fecha de fin
Inicio de la asignatura TFM	21/02/18	21/02/18
Elección del proyecto	21/02/18	27/02/18
Elección del proyecto	28/02/18	28/02/18
Escritura de la memoria del proyecto	28/02/18	9/06/18
Escritura de la PAC1	28/02/18	6/03/18
Entrega de PAC1	7/03/18	7/03/18
Escritura de la PAC2	7/03/18	17/04/18
Entrega de PAC2	18/04/18	18/04/18
Escritura de la PAC3	18/04/18	22/05/18
Entrega de PAC3	23/05/18	23/05/18
Finalización de la escritura del pro...	23/05/18	9/06/18
Entrega final del proyecto	10/06/18	10/06/18
Realización del la presentación	10/06/18	16/06/18
Realización de la presentación	10/06/18	16/06/18
Entrega de la presentación	17/06/18	17/06/18
Revisión por el tribunal	18/06/18	23/06/18
Inicio del tribunal	18/06/18	18/06/18
Periodo de estudio del tribunal	18/06/18	23/06/18
Final del tribunal	24/06/18	24/06/18
Final de la asignatura	24/06/18	24/06/18
Inicio de la ejecución	7/03/18	7/03/18
Ejecucion del proyecto	7/03/18	22/05/18
Estudio de la implementación del ...	7/03/18	9/03/18
Construcción del entorno de prue...	10/03/18	19/03/18
Estudio de los requisitos mínimos ...	20/03/18	21/03/18
Desarrollo de las aplicaciones básí...	22/03/18	10/04/18
Desarrollo de los test	11/04/18	20/04/18
Estudio de la implementación del ...	21/04/18	22/04/18
Implementación del protocolo en ...	23/04/18	7/05/18
Desarrollo de los test de MQTT-S	8/05/18	17/05/18
Estudio de los resultados	18/05/18	22/05/18

Ilustración 4: Diagrama Gantt del proyecto

Junto con la memoria del proyecto se entregará este diagrama de Gantt en un archivo realizado por el programa gratuito GantterProyect, para un mejor estudio del mismo.

1.5 Breve sumario de los productos obtenidos

Se han obtenidos los siguientes:

- Librerías de desarrollo de MQTT-S sobre OpenWSN
- Prueba de concepto de comunicaciones del protocolo MQTT-S sobre la plataforma hardware OpenMote.
- Evaluación de los datos obtenidos en la comunicación MQTT-S

1.6 Breve descripción de los otros capítulos de la memoria

Los otros capítulos de la memoria son:

- **Estado del arte:** En este capítulo se explica el estado del mundo IoT, con especial mención a sus protocolos de comunicaciones, realizando una comparativa de los mismos.
- **Descripción de la arquitectura:** Aquí se define en detalle la arquitectura que vamos a construir.
- **Detalles de las aplicaciones:** Se describen detalladamente las aplicaciones que se van a desarrollar para esta arquitectura.
- **Detalle de las pruebas:** Explicación detallada de las pruebas que se van a llevar a cabo para verificar el perfecto funcionamiento del proyecto.
- **Conclusiones:** Estudio de las ventajas e inconvenientes de funcionamiento que ofrece este protocolo sobre la situación anterior, en base al proyecto realizado.

2. Estado del arte

El mundo de "internet de las cosas" es un conjunto de dispositivos con acceso a internet o conectados entre sí, que dan información del entorno que rodea a las personas, interactuando automáticamente con ese entorno o mediante la orden de un operador. Por ejemplo, una red de sensores de un edificio inteligente, que modifica su funcionamiento en función de los valores obtenidos de los sensores.

El concepto en sí de "internet de las cosas" fue propuesto por Kevin Ashthon en el Auto-ID Center del MIT en 1999, donde se realizaban investigaciones en el campo de la identificación por radiofrecuencia en red (RFID) y tecnologías de sensores. Este concepto evolucionó a lo que podemos entender ahora como comunicación M2M (machine to machine), cubriendo una gran variedad de protocolos, dominios y tecnologías.

Desde entonces, en el entorno de "internet de las cosas" se han ido desarrollando aplicaciones para esta tecnología en el mundo, desde termostatos inteligentes, monitorización ambiental de estancias, sistemas de seguridad, sistemas de control de consumo de energía, sistemas de control de tráfico, etc…

Para este proyecto, en concreto, el uso del sistema operativo OpenWSN porque es un sistema operativo liviano, que exige pocos recursos y con código fuente abierta, lo que facilita su uso en el ámbito privado y en el ámbito universitario, siendo muy útil para proyectos de enseñanza con el hardware OpenMote por este carácter OpenSource. Es un sistema operativo desarrollado en la universidad de Berkeley que es compatible y utilizable por el dispositivo OpenMOTE.

A este respecto, se han hecho diferentes proyectos a partir de la definición del término de "Internet de las cosas", en distintos ámbitos universitarios, definiéndose diversos proyectos de implementación este concepto IoT, con varios tipos de hardware.

A continuación se indican los proyectos que utilizan el sistema operativo OpenWSN con el hardware OpenMote:

- Demostrador de Internet of Things, con la tecnología IEEE 802.15.4e, utilizando el sistema operátivo OpenWSN, OpenSIM y things.io. de Manuel Marquez Salas: Este proyecto nos muestra la capacidad de ejecutar el sistema OpenWSN en una Raspberry pi 3, pudiendo mandar valores de sensores mediante el protocolo COAP al portal recolector de datos things.io.
- Demostrador IoT-Could en tiempo real, de Eduardo Arias Monche, que consiste en el análisis del funcionamiento del hardware OpenMote con OpenWSN insertando datos en los portales que ha elegido el autor para el análisis del funcionamiento.
- Demostrador de Internet of things con la tecnología IEEE 802.15.4e utilizando el sistema operátivo OpenWSN, OpenMOTE y things.io. de Roberto Morago Martinez. Consiste en subir los datos de un sensor de corriente con la plataforma OpenMote y el sistema operativo OpenWSN a la plataforma things.io. Tiene un propósito similar al proyecto de Manuel Marquez Salas, pero variando el hardware de ejecución.

- Estudio general de Redes de Sensores Inalámbrica e Implantación del protocolo RIP sobre una plataforma hardware y basándose en el proyecto de Software Libre OpenWSN. de Pedro J. Ruiz Yelo. El título resume el objetivo del proyecto. Este proyecto se realizó en otra titulación de máster de software libre.
- Implementació de protocol TSCH al hardware OpenMote. de Albert Creixell Antolín. Consiste en la implementación del protocolo TSCH en el hardware OpenMote como especifica el título.

Haciendo una valoración de estos proyectos, no se ha cambiado el software OpenWSN desde el 2015, observando que sólo en dos proyectos se añade funcionalidad al software OpenWSN, pero no han sido validados dado que no aparecen reflejados en el repositorio del sistema operativo.

En los referidos proyectos se ha hecho uso del sistema operativo OpenWSN y los MOTEs, utilizándolos en alguna aplicación de lectura de sensores o para analizar alguna plataforma Cloud de recopilación de información. Es evidente la gran dificultad de la implementación de cualquier protocolo de comunicaciones en el sistema operativo OpenWSN, reto que este proyecto pretende conseguir.

El proyecto añadirá un protocolo de comunicaciones nuevo para este sistema operativo, el MQTT-SN, ofreciendo la puerta de inicio al desarrollo de aplicaciones de manera sencilla, para este protocolo.

El hardware OpenMote, ha sido objeto de una actualización de su versión "CC25383 Revision: E" siendo la nueva versión la OpenMote B.

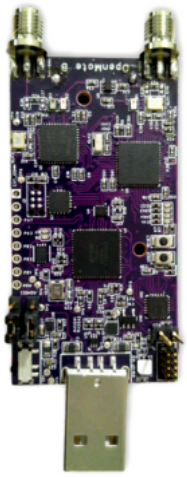


Ilustración 5: OpenMote B

Este nuevo hardware se diferencia del anterior a primera vista en el factor de forma y en las capacidades de integración, sin embargo, el OpenMote CC25383, tiene una capacidad de integración mayor y más sencilla. No se ha encontrado información detallada del nuevo dispositivo, en la web donde se comercializa.

2.1 Análisis de los protocolos de comunicación IoT

Existen varios protocolos de comunicación que están diseñados para consumir recursos mínimos y poder ser usados en las comunicaciones de IoT, entre los que se encuentran los siguientes:

- MQTT (Message Query Telemetry Transport): Creado por IBM tiene un tamaño de paquete de comunicaciones de 2 bytes, con una dimensión reducida de librería en cliente, basado en TCP, asíncrono, con una tecnología de publicador y subscriptor. Es un protocolo que se inició con otro propósito, pero actualmente está dedicado enteramente a las comunicaciones IoT. Como funciona a través de TCP, tiene una seguridad de SSL/TLS y, además, usuario y password en las comunicaciones.

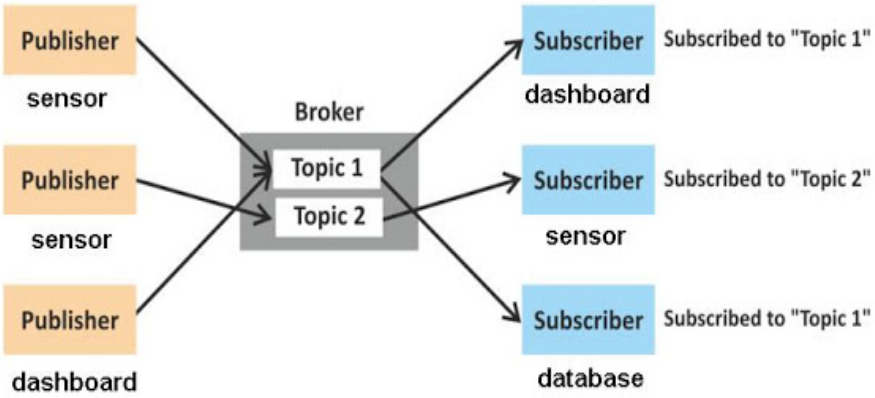


Ilustración 6: Ejemplo de arquitectura básica de MQTT

- CoAP: En julio de 2013, IETF lanzó el Protocolo de aplicación restringida (CoAP) para usarlo con nodos y redes de baja potencia y con pérdida. El CoAP, como HTTP, es un protocolo RESTful.

  El CoAP está tecnicamente alineado con HTTP e incluso tiene asignaciones uno a uno, hacia y desde HTTP. Los dispositivos de red están restringidos por microcontroladores más pequeños, con pequeñas cantidades de memoria flash y RAM, mientras que las restricciones en las redes locales, como 6LoWPAN, se deben a altas tasas de error de paquete y a un bajo rendimiento. El CoAP puede ser un protocolo apropiado para dispositivos que operan con batería o mediante extracción de energía.

  - CoAP usa UDP.
  - Debido a que el CoAP usa UDP, algunas de las funciones de TCP se reproducen directamente en el CoAP. Por ejemplo, el CoAP distingue entre mensajes que se pueden confirmar y que no se pueden confirmar.
  - Las solicitudes y las respuestas se intercambian de forma asincrónica en los mensajes de CoAP.
  - Todos los encabezados, métodos y códigos de estado se codifican de forma binaria, lo que reduce la sobrecarga del protocolo. Sin embargo, esto requiere el uso de un analizador de protocolo para solucionar problemas de red.

- A diferencia del HTTP, la capacidad de copiar en caché las respuestas de CoAP no depende del método de solicitud, sino del Código de respuesta.
- Protocolo dedicado a IoT.

- HTTP: Es la base del modelo cliente-servidor usado para la Web. El método más seguro de implementar el HTTP en un dispositivo de IoT es incluir solo un cliente, no un servidor. En otras palabras, es más seguro si el dispositivo de IoT puede iniciar conexiones a un servidor Web, pero no es capaz de recibir solicitudes de conexión.

- WebSocket: Es un protocolo que proporciona comunicación dúplex completa a través de una conexión TCP única a través de la que se pueden enviar mensajes entre el cliente y el servidor. Forma parte de la especificación HTML 5. El estándar WebSocket simplifica gran parte de la complejidad que circunda la comunicación Web bidireccional y la administración de la conexión. Usar Websockets junto con HTTP es una solución apropiada para los dispositivos de IoT si los dispositivos pueden soportar las cargas de HTTP.

- XMPP (Protocolo extensible de mensajería y presencia): Es un excelente ejemplo de una tecnología Web existente que encuentra un uso nuevo en el espacio de IoT.

  El XMPP tiene sus raíces en la mensajería instantánea y la información de presencia, y se ha ampliado a llamadas de voz y video, colaboración, middleware ligero, redifusión de contenido y enrutamiento generalizado de datos XML. Es un aspirante a administración a escala masiva de electrodomésticos de consumo, como lavadoras, secadoras, refrigeradores, etc.

  Las ventajas del XMPP son su direccionamiento, seguridad y escalabilidad, haciéndolo ideal para aplicaciones de IoT orientadas a los consumidores.

- AMQP (Advance Message Queue Protocol): Igual que MQTT, este protocolo es mantenido por OASIS y consiste en un estándar abierto para el intercambio de mensajes entre aplicaciones (M2M). En otras palabras, es un protocolo orientado a mensajes que proporciona características como el enrutamiento y la gestión de colas. Desde el punto de vista de la conexión define varias entidades:

  - Corredor de mensajes: servidor al que los clientes AMQP se conectan usando el protocolo.

  - Usuario: entidad que, mediante la presentación de credenciales, puede ser autorizada a conectarse a un corredor.

  - Conexión: conexión física usando por ejemplo TCP/IP y ligada a un usuario.

  - Canal: conexión lógica que está unida a una conexión.

- Stomp (Simple or Streaming Text Oriented Messaging Protocol): Es un protocolo orientado a mensajería de texto, muy fácil de usar. Los clientes se conectan a un

intermediario para intercambiar mensajes, por lo que su mayor utilidad se da en los middlewares. Esto además posibilita que pueda ser utilizado de forma nativa desde casi cualquier lenguaje de programación.

Su funcionamiento es muy similar al HTTP, por lo que funciona sobre TCP y sus principales comandos son: CONNECT, SEND, SUBSCRIBE, UNSUBSCRIBE, BEGIN, COMMIT, ABORT, ACK, NACK, DISCONNECT.

Estos serían, entre otros, los protocolos más usados en el mundo del IoT, siendo MQTT el más popular entre ellos porque ofrece el tamaño más pequeño de paquete además de un protocolo orientado a conexión que ofrece una seguridad SSL o TLS con usuario y password.

3. Descripción de la arquitectura

El principal problema que se quiere resolver es poder integrar el protocolo MQTT dentro del sistema operativo OpenWSN y poder tener un interfaz de comunicaciones del protocolo ya mencionado. Para lo que se ha diseñado la arquitectura que se describe en ese capítulo.

Hoy en día el protocolo MQTT es el protocolo de comunicaciones IoT más extendido debido a su tamaño de paquete que es el más pequeño, permitiendo así poder funcionar en canales de transmisión de baja velocidad, pero de larga distancia. Por lo tanto, es necesario remodelar el sistema operativo OpenWSN para poder adaptarlo al nuevo protocolo.

Para llevar a cabo la arquitectura que vamos a construir, es necesario estudiar y revisar el sistema operativo en el que vamos a integrar el protocolo MQTT. En la página web de este sistema operativo viene una descripción de su arquitectura que se muestra resumida en la siguiente imagen.

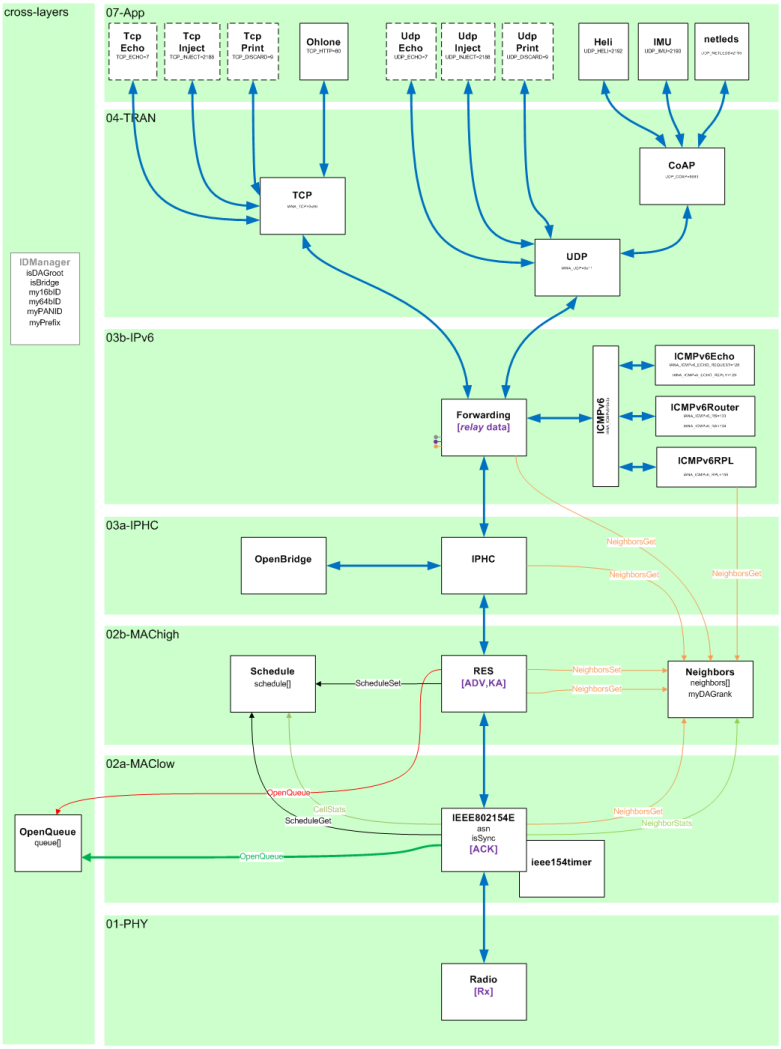


Ilustración 7: Stack del sistema operativo OpenWSN

Podemos observar que la arquitectura del sistema operativo sería compatible con el protocolo a implementar porque tiene una capa de transporte compatible con TCP, pero al parecer está siendo deprecada esa capa de transporte, por lo tanto, nos vemos obligados a estudiar otras opciones dentro del protocolo MQTT para que pueda funcionar con el protocolo de transporte UDP.

En este caso existe la posibilidad de implantar una variante del protocolo que se llama MQTT-SN que utiliza una implementación de UDP supliendo las carencias de un protocolo orientado a conexión con tiempos de vida de la conexión. Debido a la necesidad de utilizar el protocolo UDP obliga a tener un Gateway que encamine todos los dispositivos que funcionan a través de UDP y pasarlo a TCP, quedando la arquitectura de esta manera.

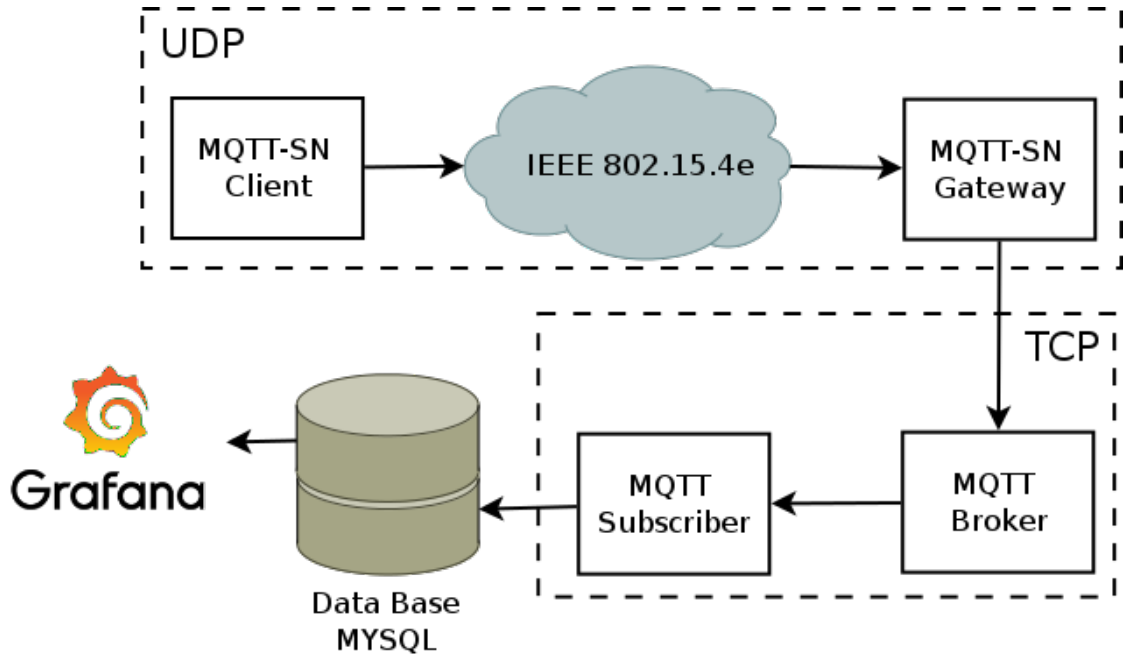


Ilustración 8: Arquitectura definitiva del proyecto

La variación que se ha producido en el planteamiento inicial del proyecto no es significativa, ya que el protocolo MQTT-SN es una variante aceptada para poder usar el protocolo MQTT sobre UDP.

Otro de los inconvenientes de esta arquitectura es que el desarrollo realizado no debe de ocupar mucho espacio para que se pueda utilizar como software embebido dentro de la placa OpenMote, por lo tanto, MQTT-SN tenía que tener características mínimas para poder realizarlo.

Para ello se ha utilizado la siguiente implementación del protocolo:
https://github.com/eclipse/paho.mqtt-sn.embedded-c

Lo que ayuda a realizar el proyecto, dado que cumple todas las características que necesitamos para llevarlo a cabo. Por lo tanto, se va a modificar el software OpenWSN para hacerlo compatible con el protocolo MQTT-SN. Eso se consigue añadiendo en el momento de compilar todas las librerías del proyecto como librerías del sistema

operativo, al igual que ocurre con OpenCoap. Lo que se consigue manipulando los archivos de SCons de la manera que se describe a continuación.

En concreto, se añaden las librerías de MQTT-SN, modificando el archivo SConscript como se describe en la ilustración 9.

```
# source files that should be included in the build other than
# those with default application name (application.c)
additionalSourceFiles = [
    os.path.join('opencoap','openoscoap.c'),
    os.path.join('opencoap','usha.c'),
    os.path.join('opencoap','hkdf.c'),
    os.path.join('opencoap','hmac.c'),
    os.path.join('opencoap','sha224-256.c'),
    os.path.join('opencoap','cborencoder.c'),
    os.path.join('openmqtt','MQTTSNConnectClient.c'),
    os.path.join('openmqtt','MQTTSNDeserializePublish.c'),
    os.path.join('openmqtt','MQTTSNPacket.c'),
    os.path.join('openmqtt','MQTTSNSearchClient.c'),
    os.path.join('openmqtt','MQTTSNSerializePublish.c'),
    os.path.join('openmqtt','MQTTSNSubscribeClient.c'),
    os.path.join('openmqtt','MQTTSNUnsubscribeClient.c'),
    os.path.join('cjoin','cbor.c'),
]
```

Ilustración 9: Ejemplo de donde se añaden los archivos con extensión "c"

Es necesario añadir todos los archivos con extensión “c” dentro de la librería openmqtt en esta sección. En la ilustración 10 se indica la sección donde se añaden los archivos de cabecera.

```
# header files that should be included in the build other than
# those with default application name (application.h)
additionalHeaderFiles = [
    os.path.join('opencoap','openoscoap.h'),
    os.path.join('opencoap','sha.h'),
    os.path.join('opencoap','cborencoder.h'),
    os.path.join('openmqtt','MQTTSNConnect.h'),
    os.path.join('openmqtt','MQTTSNPacket.h'),
    os.path.join('openmqtt','MQTTSNPublish.h'),
    os.path.join('openmqtt','MQTTSNSearch.h'),
    os.path.join('openmqtt','MQTTSNSubscribe.h'),
    os.path.join('openmqtt','MQTTSNUnsubscribe.h'),
    os.path.join('openmqtt','StackTrace.h'),
    os.path.join('cjoin','cbor.h'),
]
```

Ilustración 10: Ejemplo de donde se añaden los archivos con extensión "h"

Además, para añadir MQTT-SN como librería del sistema hay que hacer un último cambio, en la ilustración 11 se indica la zona del archivo modificado en la que hay que incluir openmqtt como librería del sistema.

```
# union of default and additional apps (without duplicates)
apps      = ['opencoap','openmqtt']
apps     += sorted(list(set(defaultApps+userApps)))
if userApps:
    appsInit  = ['opencoap','openmqtt'] # make sure opencoap is initialized first
    appsInit += sorted(list(set(userApps)))
else:
    appsInit = []
```

Ilustración 11: Añadir la librería openmqtt como userapp

A continuación de que el Stack del sistema operativo sea modificado, lo que se consigue añadiendo la subcapa de MQTT-SN sobre la capa de transporte, del mismo modo que se ha realizado con OpenCoap. Para poder programar esa subcapa de transporte debe ser manejada por una aplicación, esta aplicación la llamaremos mrandom y será la primera aplicación de MQTT-SN sobre el sistema operativo.

El Stack quedaría modificado tal y como se muestra en la ilustración 12.

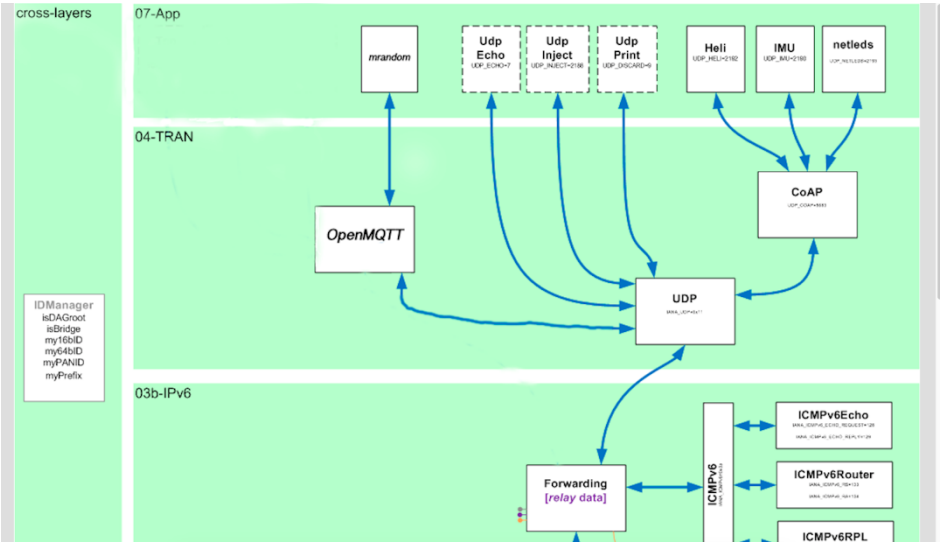


Ilustración 12: Stack modificado por la libreria MQTT-SN

En la ilustración 12 se observa que la capa TCP desaparece y se añade una subcapa de MQTT-SN, del mismo modo que se realiza en CoaP.

3.1 Criterios de validación de la arquitectura

Esta arquitectura implementa un protocolo de comunicación de IoT y para validar su buen funcionamiento, es necesario analizar el protocolo, mediante un análisis de trazas o paquetes. Para la validación del protocolo es necesario un analizador de paquetes objetivo, la herramienta que se va utilizar es el capturador de paquetes Wireshark que permitirá ver los diferentes paquetes enviados del protocolo MQTT y MQTT-SN que se ha integrado en el sistema operativo.

3.1.1 Comunicación entre OpenMote y el Gateway MQTT-SN

La comunicación MQTT-SN se caracteriza porque es un paquete UDP serializado que incluye el tipo de paquete y la cadena de datos. En este caso, hay que analizar la siguiente secuencia de envíos y respuestas entre el cliente y el Gateway:

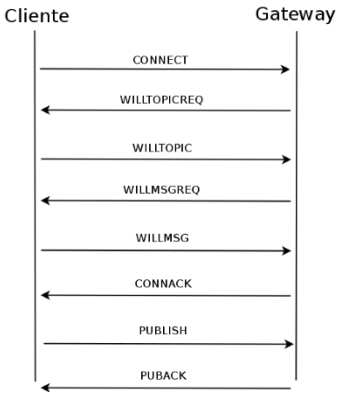


Ilustración 13: Diagrama de secuencia MQTT-SN

3.1.2 Comunicación entre el Gateway MQTT-SN y el Broker MQTT

Para validar la comunicación de esta parte del proyecto, hay que analizar la secuencia que se recoge en la ilustración 14

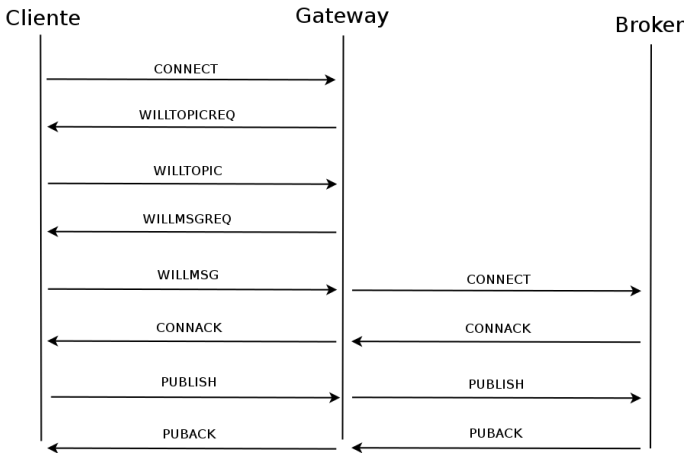


Ilustración 14 Diagrama de secuencia MQTT

Esta secuencia se produce acto seguido de recibir el mensaje WILLMSG, el Gateway automáticamente se conecta al bróker de manera no transparente, enviando un CONNECT y recibiendo una respuesta que es reproducida al Cliente MQTT-SN. También hay que analizar el momento de publicar los mensajes hacia el bróker, validando el mensaje PUBLISH y el PUBACK y retrasmitiendo el PUBACK al cliente MQTT-SN.

3.1.3 Comunicación entre el Broker MQTT y el Suscriber MQTT

Para esta última secuencia del protocolo de MQTT, hay que estudiar la aplicación que se conecta al bróker para procesar los mensajes recibidos de los clientes, de la manera que indica en la ilustración 15, siguiente

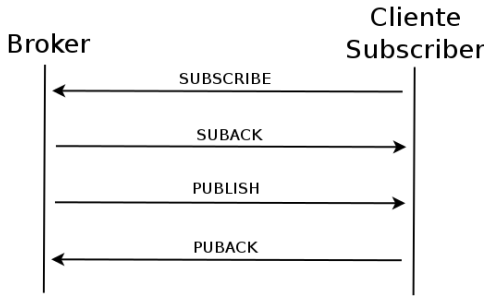


Ilustración 15: Diagrama de secuencia MQTT Subscriber

El Broker retransmite sobre el suscriptor todo lo que los publicadores introducen en el Topic correspondiente.

3.1.4 Principales métricas de análisis de resultados.

Es necesario, que además de especificar los mensajes que son necesarios, para una validación, es necesario definir unas métricas que deben cumplir los mensajes que se envían en este protocolo, estas métricas están basadas en el contenido.

Para ello nos vamos a ayudar de las diferentes especificaciones de los dos protocolos que trabajamos:

- MQTT-SN: http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf
- MQTT: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf

Y con ello definimos varias cosas a buscar dentro de los mensajes, como por ejemplo el client ID de los mensajes connect, así como los tipos de mensajes enviados dentro del paquete de datos. Los topic y mensajes también se pueden considerar unas métricas válidas para la validación. Pero el más importante de todos es que el valor enviado y la estructura del mensaje enviado se corresponda con la definición del protocolo.

3.2 Manual de desarrollo de aplicaciones en el sistema operativo OpenWSN

Este sistema operativo utiliza una herramienta llamada scons, que se considera una utilidad de construcción de software, ya que soluciona dependencias atendiendo a los niveles que se determinen, para ello utiliza unos archivos que configuran el script de compilación.

Del mismo modo que se ha explicado la forma de añadir librerías nuevas del sistema operativo OpenWSN, hay que hacer con las aplicaciones respecto de la herramienta scons.

```
#===== retrieve the list of apps to build

defaultApps   = []
if localEnv['board']=='python':
    defaultApps  += [
        'c6t',
        'cinfo',
        'cleds',
        'cwellknown',
        'mrandom',
        'uecho',
        'urandom',
        'uinject',
        'userialbridge',
        'uexpiration',
        'uexpiration_monitor',
        'rrt',
        'cexample',
        'cstorm',
        'cjoin',
    ]
```

Ilustración 16: Sección de inserción de nuevas aplicaciones en scons

Como se puede observar en la ilustración 16, si se quiere añadir una nueva aplicación que corra en el OpenMote y que sea compilada y enlazada, es necesario añadir el nombre de la misma a esta sección del archivo SConscript. A partir de esa acción, el sistema, cuando se compile el cliente OpenMote con el sistema operativo OpenWSN, tendrá disponible la aplicación.

Además, es necesario añadir las funciones de inicio de las diferentes aplicaciones dentro del archivo "openapps.c", así como sus archivos de cabecera, ilustración 17.

```
//========================== private =========================================

void openapps_init(void) {
   //-- 04-TRAN
   //opencoap_init();     // initialize before any of the CoAP applications
   openmqtt_init();     // initialize before any of the MQTT applications

   // CoAP
   //c6t_init();
   //cinfo_init();
   //cleds_init();
   //cjoin_init();
   //cwellknown_init();
   //rrt_init();

   // MQTT
   mrandom_init();

   // UDP
   //uecho_init();
   //urandom_init();
   //uinject_init();
   //userialbridge_init();
   //uexpiration_init();
```

Ilustración 17: Funciones de inicio de las diferentes aplicaciones e interfaces.

En la ilustración 18 se pueden ver los archivos de cabecera de las aplicaciones e interfaces.

```
/**
\brief Applications running on top of the OpenWSN stack.

\author Thomas Watteyne <watteyne@eecs.berkeley.edu>, September 2014.
*/

#include "opendefs.h"

// CoAP
#include "opencoap.h"
#include "c6t.h"
#include "cinfo.h"
#include "cleds.h"
#include "cjoin.h"
#include "cwellknown.h"
#include "rrt.h"

// MQTT
#include "openmqtt.h"
#include "mrandom.h"

// UDP
#include "uecho.h"
#include "urandom.h"
#include "uinject.h"
#include "userialbridge.h"
#include "uexpiration.h"
#include "uexpiration_monitor.h"
```

Ilustración 18: Includes de cabeceras

Para que el programa pueda ser compilado, hay que realizar las siguientes acciones:

- Dentro del archivo openwsnfw/bsp/boards/python/openwsnmodule_obj.h se tiene que definir tanto la variable propia de la aplicación, es decir, la variable que va a contener el registro del socket UDP, como las funciones callback de recepción y envió. Si se realiza una búsqueda de la variable de una aplicación que ya esté completa, se podrá ver dónde y cómo están definidas.
- En el siguiente archivo projects/python/SConscript.env es necesario definir todas las funciones que contiene la aplicación, en el caso de la aplicación creada, tenemos lo siguiente:

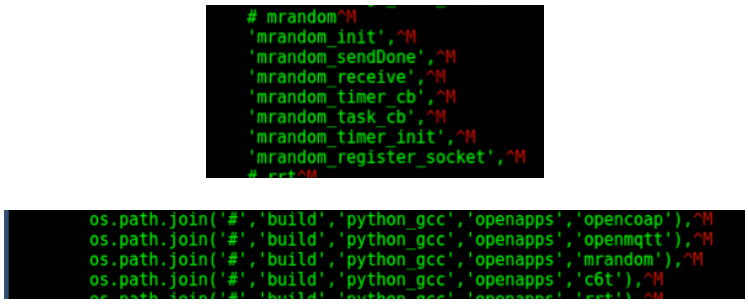


Ilustración 19: Archivo de definición de funciones

El comando para compilar el código y generar una build dentro del entorno simulado será el siguiente:

scons board=python toolchain=gcc oos_openwsn

El comando para generar la build y programar el dispositivo OpenMote es el siguiente:

sudo scons board=openmote-cc2538 toolchain=armgcc oos_openwsn bootload=/dev/ttyUSB0

El ultimo comando necesario para poder trabajar con el firmware que modificamos es el que permite simular los MOTE's, o si es necesario ser un Gateway para todos los MOTE's que se arranquen en esta red. En este caso, se va a trabajar con un entorno simulado ya que no se ha dispuesto del hardware adecuado. El comando mencionado es:

sudo scons runweb --simCount=2 --simTopology=fully-meshed

Este comando levanta la aplicación openvisualizer y su web teniendo el siguiente aspecto:

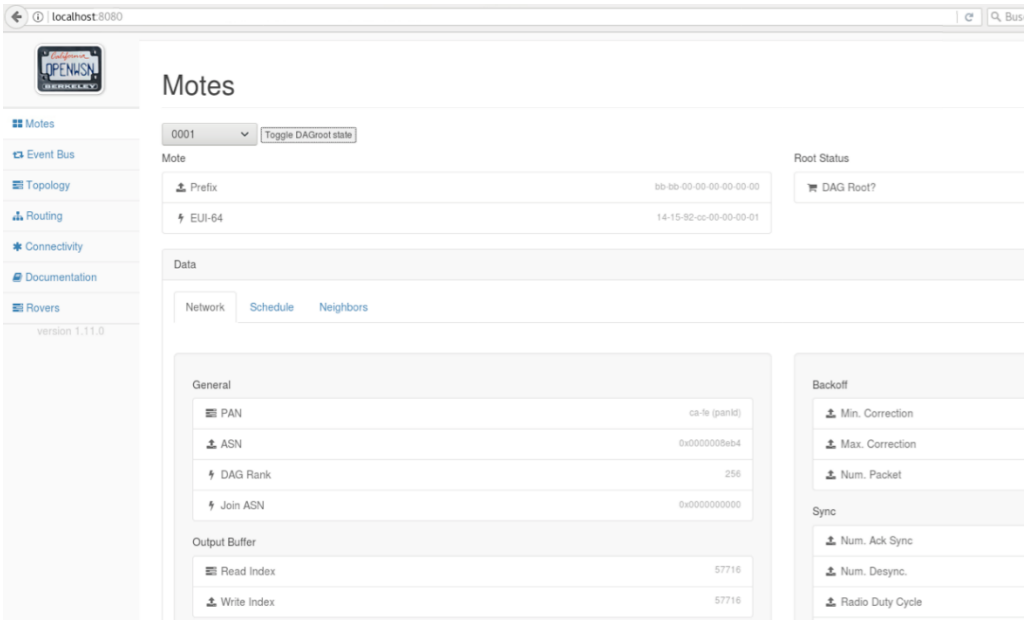


Ilustración 20: OpenVisualizer

Este interfaz web permite poder seleccionar el Mote, que es root, que es el que recibe los mensajes de todos los demás Motes. Esto se tiene que hacer cada vez que arranquemos el sistema.

Estos comandos tienen más modificadores que no es necesario especificar, porque para trabajar en la aplicación que es el propósito de este proyecto solo se necesitan estos. Además, para poder depurar el código de las aplicaciones que se creen, hay que utilizar siempre el entorno simulado, porque tenemos que depurar el código, mediante trazas que se muestran en la interfaz de OpenVisualizer en la consola donde lo ejecutamos.

3.3 Principales aspectos de diseño del software creado en la arquitectura.

La razón por la que se decide hacer este primer paso del protocolo MQTT con la arquitectura definida en el proyecto es porque crear el interfaz que se pueda utilizar desde las aplicaciones, facilita el desarrollo de las mismas abstrayéndose del funcionamiento de dicho interfaz y protocolo.

El único funcionamiento que necesitaría saber el desarrollador de la nueva aplicación MQTT, sería el workflow de trabajo con el interfaz. Que este consiste en:

1. Definir el Topic, el mensaje y lo que se necesita mandar.
2. Llamar a la secuencia de conexión cuando se detecte que hay radio y que no es DagRoot.
3. Publicar la información.

Y para el caso del suscriptor:

1. Definir el Topic donde te quieres conectar, saber qué información se quiere recibir.
2. Conectar con el Broker que nos va a enviar la información.
3. Suscribirse al Topic donde se recibe esa información.

4. Esperar a recibir información y procesarla.

Tanto, en la aplicación incluida en el sistema operativo OpenWSN, como en el interfaz que se ha creado openmqtt, se han seguido las normas de desarrollo del sistema operativo que vienen especificadas en la web del mismo.

En cuanto a las otras aplicaciones que intervienen en la arquitectura, se ha optado por realizar dos monolitos de código con diferentes funciones, que son llamadas desde la función principal, cuyos aspectos de software serán especificados en el capítulo siguiente.

4. Detalle de las aplicaciones

En este capítulo se detallan todas las aplicaciones que se han desarrollado o utilizado en este proyecto, de las que 4 aplicaciones son nuevas y el resto de aplicaciones de terceros que se han utilizado para completar el proyecto, las aplicaciones del proyecto son:

- Interfaz openmqtt: esta aplicación es un subproceso que está integrado como librería del sistema operativo que recibe órdenes de las aplicaciones que utilizan esta librería. Esta librería utiliza a su vez el interfaz de la capa de transporte UDP.
- Aplicación mrandom: es la aplicación de ejemplo que se crea como uso del interfaz openmqtt.
- Aplicación Gateway MQTT-SN: es el Gateway que convierte el protocolo UDP a protocolo TCP.
- Aplicación Broker mosquito: es el acumulador de datos MQTT además de ser el distribuidor entre los suscriptores.
- Aplicación subscriber: extrae los datos del bróker de un topic especifico y los carga sobre una base de datos.
- Aplicación de Base de datos MYSQL: Recopilador de información extraída del bróker.
- Aplicación que ofrece un dashboard (Grafana): genera una visualización de los datos transmitidos por el cliente OpenMote.

Todas las aplicaciones desarrolladas han sido confeccionadas con el lenguaje de programación C, SQL y json. El código se añade en anexos de esta memoria del proyecto. El detalle de distribución de las carpetas del proyecto se detalla en los siguientes puntos

4.1 Estructura de carpetas

El primer nivel de archivos modificados del proyecto es el siguiente:

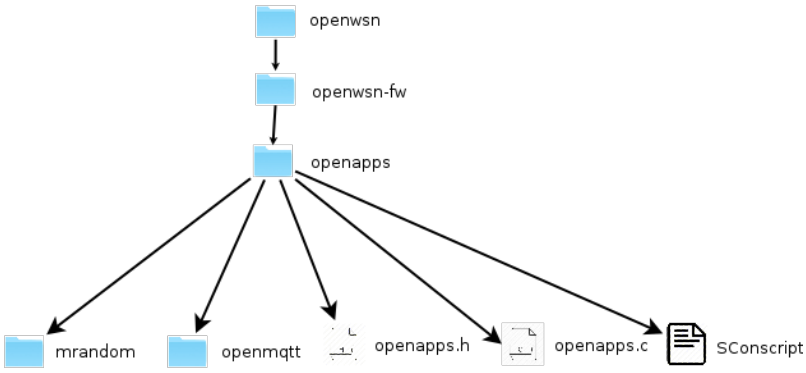


Ilustración 21: Nivel 1 de carpetas con los archivos modificados

El segundo y tercer nivel de archivos modificados lo estructuraremos a su vez en dos partes, la aplicación mrandom y la interfaz openmqtt.

El segundo nivel de la aplicación mrandom es el que se representa en la ilustración 22.

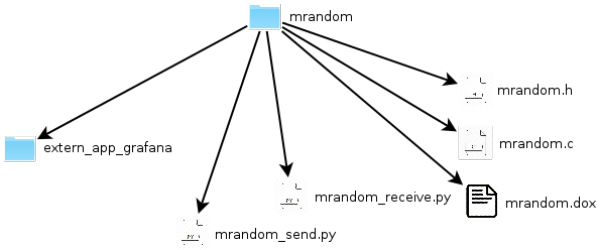


Ilustración 22: Segundo nivel por la rama de mrandom

El segundo nivel de la rama de openmqtt está representado en la ilustración 23.

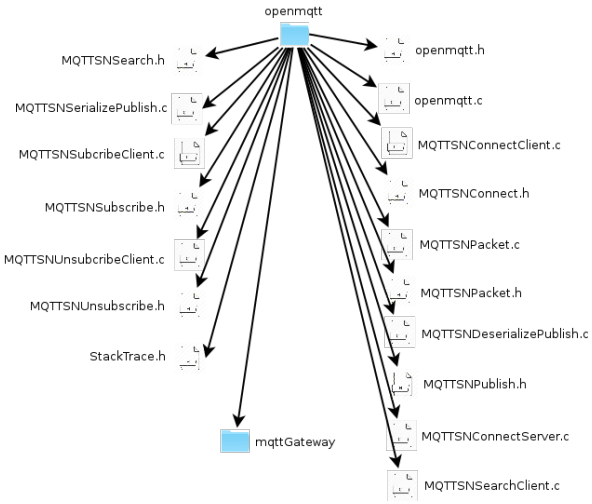


Ilustración 23: Nivel 2 Por la rama de openmqtt

La ilustración 24 representa el tercer nivel por la rama de mrandom.

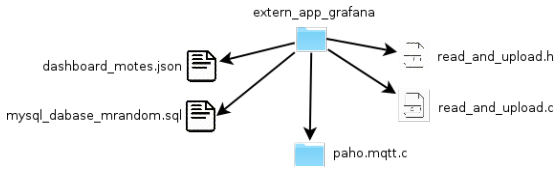


Ilustración 24: Nivel 3 de la rama mrandom

Y la ilustración 25 representa el tercer nivel de openmqtt:

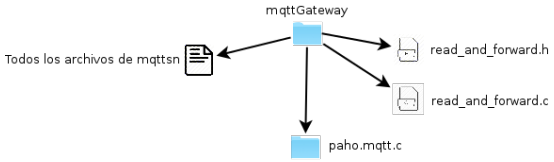


Ilustración 25: Nivel 3 de la rama openmqtt

En esta estructura de carpetas se distribuyen todos los archivos necesarios para ejecutar la arquitectura definida, a continuación, se explican en detalle las aplicaciones desarrolladas.

4.2 Interfaz OpenMQTT.

Este interfaz consiste en una serie de funciones que van a ser utilizadas por el resto de aplicaciones, la situación en la que se encuentra esta aplicación dentro del firmware es la representada resumidamente en la ilustración 26

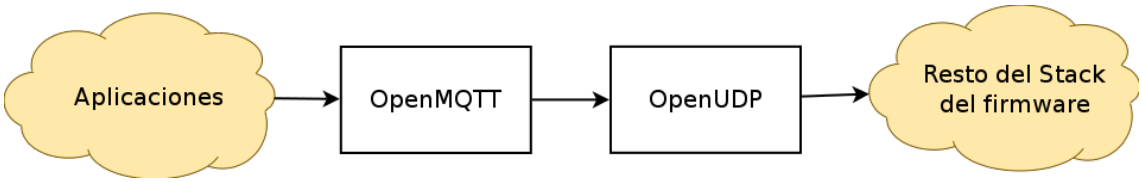


Ilustración 26: Funcionamiento de Openmqtt con respecto al stack del firmware

El esquema de las funciones que describen la forma de trabajar son las mostradas en la ilustración 26:

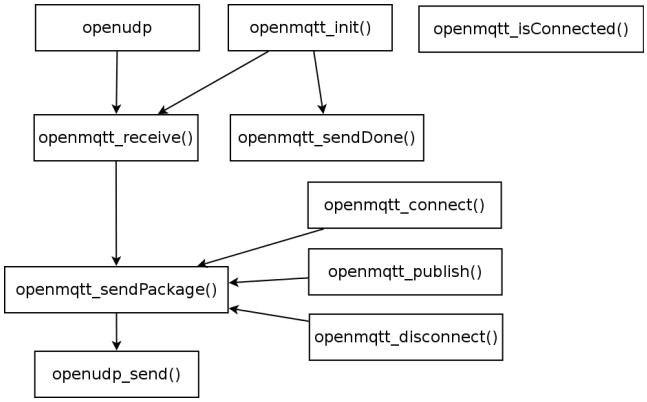


Ilustración 27: Diagrama de las funciones que componen el interfaz openmqtt

Existe una conexión del stack con la librería openudp que da soporte a este interfaz de comunicaciones. Es una capa por encima de openudp.

Para explicar el funcionamiento de este interfaz a continuación se incluye una breve descripción de cada una de las funciones que aparecen en el diagrama.

- **void openmqtt_init(void):** Esta función es la que inicia el interfaz, limpia la variable de la aplicación a la que se le asignan las direcciones de callback de recepción y envió, así como el puerto de escucha y envió de la aplicación. En el desarrollo de la aplicación el puerto se ha definido como el 61620.
- **void openmqtt_receive(OpenQueueEntry_t* msg):** Esta función es definida en openmqtt_init como callback de recepción de datos, apoyada en la librería openudp. Es la encargada de recibir los mensajes de respuesta y reaccionar en consecuencia al mensaje recibido. Esta función reacciona a los siguientes mensajes:
    - *WILLTOPICREQ*: después de enviar el mensaje connect el Gateway requiere que se envié el Topic de los mensajes publish, lo que consiste en

des-Serializar los mensajes *WILLTOPICREQ* y enviar el *WILLTOPIC* con el Topic que se necesita.
    - *WILLMSGREQ*: cuando el *WILLTOPIC* es recibido en el Gateway este responde con el requerimiento del mensaje, por lo tanto, se des-Serializa el mensaje *WILLMSGREQ* y enviamos *WILLMSG*.
    - *CONNACK*: Es el último mensaje de la secuencia de conexión, es el que dice si se ha conectado a través del Gateway, lo que hace es validar la conexión y poner el flag de isConnected en true.
    - *PUBACK*: Es el mensaje de respuesta de un *PUBLISH* en el momento que se recibe el *PUBACK* del Gateway se envía el siguiente *PUBLISH*.
    - *DISCONNECT*: Cambia la variable isConnected a falso.
- **void openmqtt_sendDone(OpenQueueEntry_t* msg, owerror_t error):** Esta función está definida como callback simplemente borra el mensaje que entra.
- **uint8_t openmqtt_sendPackage(uint8_t *buff,uint16_t lenght**): Esta función es la encargada de enviar todos los paquetes serializados a través de las funciones MQTT-SN, se apoya en la librería openudp propia del sistema operativo WSN. Como se puede observar en la ilustración 26 es la función en la que se apoya toda la interfaz para enviar paquetes, es como un canalizador de paquetes.
- **uint8_t openmqtt_connect(uint8_t *openmqtt_dst_addr, uint8_t id, MQTTSNString *Topic_ini, MQTTSNString *Msg_ini):** Esta función es la que genera el paquete de conexión inicial de la secuencia de conexión. Este paquete tiene que ser capaz de detectar si está conectado, si es DagRoot y por último, si tiene radio para poder conectarse. También guarda en la memoria todos los datos necesarios para conectarse, como son el TOPIC y el MENSAJE. El Client ID que identifica a la placa, se introduce de manera fija, pero se debería de hacer que concuerde con la MAC de la placa.
- **uint8_t openmqtt_publish(MQTTSN_topicid *topic, uint8_t *payload, uint16_t length):** Esta función es capaz de detectar si está conectado o no, además si se ha recibido un puback de respuesta de un publish. Es la función encargada de enviar las publicaciones que llegan a través del buffer de entrada de la función (payload).
- **bool openmqtt_isConnected(void):** Esta función indica a la aplicación que la llama si está conectado el interfaz o no.
- **uint8_t openmqtt_disconnect(void):** Envía un mensaje de disconnect al Gateway y espera a recibir una respuesta para poder cambiar de estado conectado a estado desconectado.

Este interfaz todavía tiene trabajo por completar, debido a que a día de hoy esta implementado, pero solo es capaz de atender a una sola aplicación simultáneamente. Pero como integración inicial cumple con los requisitos establecidos. La forma de compilarse, linkarse y ejecutarse ha quedado reflejado en el tema anterior.

Los aspectos más relevantes del software que se ha desarrollado para esta aplicación son los siguientes:

- El interfaz MQTT-SN funciona como una aplicación apoyándose sobre openudp como hace opencoap. En la siguiente secuencia de código se describe como se inicializa el socket de openudp.

```
1.  void openmqtt_init(void) {
2.
3.        memset(&openmqtt_vars,0,sizeof(openmqtt_vars_t));
4.
5.        // register at UDP stack
6.        openmqtt_vars.desc.port              = WKP_UDP_MQTT;
7.        openmqtt_vars.desc.callbackReceive   = &openmqtt_receive;
8.        openmqtt_vars.desc.callbackSendDone  = &openmqtt_sendDone;
9.        openudp_register(&openmqtt_vars.desc);
10. }
```

- Dentro del interfaz, en la función callback de recepción de paquetes esta la sentencia switch que es clave para el funcionamiento de dicho interfaz MQTT-SN, como parte de la sentencia se incluyen unos ejemplos de funcionamiento que se explican más adelante:

```
1.  memcpy(rcvbuff,&msg->payload[0],msg->length);
2.  len = MQTTSNPacket_decode(rcvbuff, msg->length, &datalen);
3.  rc=rcvbuff[len];
4.  switch(rc){
5.  case MQTTSN_CONNACK:
6.       if (MQTTSNDeserialize_connack(&connack_rc, rcvbuff, MAX_PACKET_SIZE) == 1){
7.          if(SUCCESS == connack_rc){
8.              isConnected = TRUE;
9.              puback_recv = TRUE;
10.         }
11.         return;
12.      }
13.      break;
14. case MQTTSN_WILLTOPICREQ:
15.     if (MQTTSNDeserialize_willtopicreq(rcvbuff, MAX_PACKET_SIZE) == 1){
16.         len = MQTTSNSerialize_willtopic(sendbuf, MAX_PACKET_SIZE, 0, 0, Topic);
17.         openmqtt_sendPackage(&sendbuf[0], len);
18.         return;
19.     }
20.     break;
21. case MQTTSN_WILLMSGREQ:
22.     if (MQTTSNDeserialize_willmsgreq(rcvbuff, MAX_PACKET_SIZE) == 1){
23.         len = MQTTSNSerialize_willmsg(sendbuf, MAX_PACKET_SIZE, Msg);
24.         openmqtt_sendPackage(&sendbuf[0], len);
25.         return;
26.     }
27.     break;
```

En la anterior sección de código, se observa que se estructura cada paso de esta sentencia de la misma manera, primero se identifica el tipo de paquete que esel que viene en el segundo valor del buffer. Después de identificar el paquete, se desserializa con la función del tipo de paquete que ha llegado y dependiendo de la respuesta llegada se toma una acción u otra.

- La parte más importante de la función openmqtt_connect, muestra cómo funcionan los envíos al Gateway desde este cliente

```
1.  memcpy(&openmqtt_dst_addr_connect[0],openmqtt_dst_addr,16);
2.  memcpy(&Topic,Topic_ini,sizeof(MQTTSNString));
3.  memcpy(&Msg,Msg_ini,sizeof(MQTTSNString));
4.  MQTTSNPacket_connectData options = MQTTSNPacket_connectData_initializer;
5.  char cadena[50];
6.  sprintf(cadena,"d:mqttsn:openwsn:%d", id);
7.  options.clientID.cstring = cadena;
8.  len = MQTTSNSerialize_connect(sendbuf, MAX_PACKET_SIZE, &options);
9.  if(openmqtt_sendPackage(&sendbuf[0], len) == FAILURE)
10.     return FAILURE;
11. return SUCCESS;
```

La sección de código anterior muestra que openmqtt utiliza una función genérica que envía el paquete por el protocolo de transporte openudp, simplificando el desarrollo del interfaz. Además, se puede observar que en esta zona se genera el ClientID que será enviado al Gateway.

El resto de funciones siguen los mismos principios, pero con la salvedad de que la función openmqtt_publish utiliza su función específica para serializar la información.

4.3 Aplicación mrandom.

La aplicación mrandom ha sido diseñada para usar la librería de openmqtt envía un número aleatorio como si se tratase de un valor de un sensor, para que sea recogido en el otro extremo de la comunicación. El esquema de funciones es el siguiente:

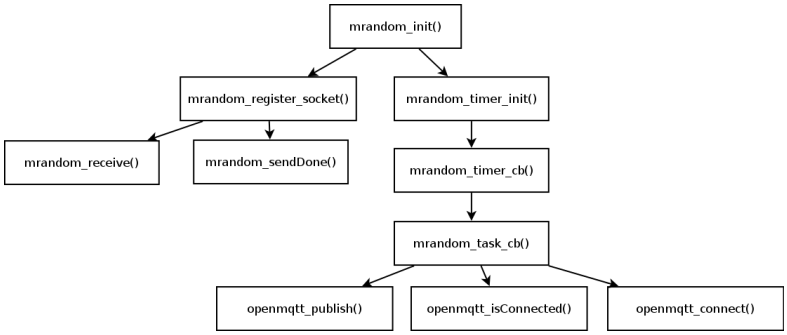


Ilustración 28: Diagrama de funciones de la aplicación mrandom

A continuación, una breve explicación del funcionamiento del diagrama:

- **void mrandom_init(void):** es la encargada de inicializar las callback del socket además de la callback del timer que crea. Se ejecuta la primera y es el detonante de toda la aplicación.
- **void mrandom_register_socket(mrandom_vars_t *mrandom, bool send_receive):** asigna a las funciones de receive y sendDone, las callback del socket asignado a la aplicación.
- **void mrandom_sendDone(OpenQueueEntry_t* msg, owerror_t error):** es la función asignada como callback al socket udp de la aplicación, pero en este caso no se utiliza para nada, ya que cuando se ejecuta desecha el paquete que le llega por la llamada de la función.
- **void mrandom_receive(OpenQueueEntry_t* msg):** lo mismo ocurre con esta función, el paquete que recibe por la llamada de la función es rechazado y eliminado.
- **void mrandom_timer_init(mrandom_vars_t *send, uint16_t period_aux):** es la encargada de registrar la callback del temporizador que se crea, para que la aplicación este enviando datos cada cierto tiempo.
- **void mrandom_timer_cb(opentimers_id_t id):** esta función es llamada por la mrandom_init para realizar la función de asignar callback y definir temporizador.
- **void mrandom_task_cb(void):** es la función ejecutora de la aplicación, la que envía y conecta con el Gateway a través del interfaz MQTT-SN. Se apoya en la

librería para realizar esta función. Es llamada cada cierto periodo de tiempo, como consecuencia del timer que se crea.

Los principales aspectos de diseño de la aplicación mrandom, que simplifican el desarrollo de las aplicaciones basadas en el protocolo MQTT-SN son:

- La aplicación mrandom no usaría la estructura básica que se podría usar con las aplicaciones basadas en la utilización de la librería openudp directamente, quedando en el siguiente segmento de código demostrado:

```
1.  void mrandom_init(void) {
2.     mrandom_register_socket(&mrandom_vars_send, TRUE);
3.     mrandom_timer_init(&mrandom_vars_send, MRANDOM_PERIOD_MS);
4.  }
5.
6.  void mrandom_sendDone(OpenQueueEntry_t* msg, owerror_t error) {
7.     openqueue_freePacketBuffer(msg);
8.  }
9.
10. void mrandom_receive(OpenQueueEntry_t* msg) {
11.    openqueue_freePacketBuffer(msg);
12. }
```

  En este segmento de código de la aplicación mrandom se puede observar que las estructuras básicas de desarrollo con las aplicaciones de MQTT no se cumplen, porque cuando aparece algún paquete por el canal ordinario de comunicaciones es desechado.

- Como novedad en esta aplicación el registro del socket y del timer quedan relegadas a dos funciones, con lo que se consigue la simplificación de la acción, que son las siguientes:

```
1.  void mrandom_register_socket(mrandom_vars_t *mrandom, bool send_receive){
2.     memset(mrandom,0,sizeof(mrandom_vars_t));
3.     mrandom->desc.port              = WKP_UDP_RANDOM_SEND;
4.     mrandom->desc.callbackSendDone  = NULL;
5.     mrandom->desc.callbackReceive   = NULL;
6.     openudp_register(&mrandom->desc);
7.  }
8.
9.  void mrandom_timer_init(mrandom_vars_t *send, uint16_t period_aux){
10.   send->period = period_aux;
11.   // start periodic timer
12.   send->timerId = opentimers_create();
13.   opentimers_scheduleIn(
14.       send->timerId,
15.       period_aux,
16.       TIME_MS,
17.       TIMER_PERIODIC,
18.       mrandom_timer_cb
19.   );
20. }
```

- Como se ha realizado en otras aplicaciones, por ejemplo uinject, se utilizan los timers internos del sistema operativo OpenWSN para cada cierto tiempo lanzar una ejecución, que envía un numero aleatorio simulando los valores de un sensor. La función expuesta y que concentra las llamadas al interfaz openmqtt es la siguiente:

```
1.  void mrandom_task_cb(void) {
2.
3.        Topic.cstring = "Pruebas";
4.        Msg.cstring = "MQTTSNClient";
5.
6.        if(ieee154e_isSynch() == FALSE)
7.              return;
8.
9.        if (idmanager_getIsDAGroot()){
10.             opentimers_destroy(mrandom_vars_send.timerId);
11.             return;
12.       }
13.
14.       if(!openmqtt_isConnected()){
15.             openmqtt_connect(mrandom_dst_addr,1,&Topic, &Msg);
16.             return;
17.       }
18.
19.       uint8_t low_num = 0, hi_num = 101, rc = 0;
20.       srand(time(NULL));
21.       uint16_t random_number= (uint16_t)(rand() % (hi_num - low_num)) + low_num;
22.
23.       uint8_t payload[2];
24.       payload[1] = (uint8_t)((random_number & 0xff00)>>8);
25.       payload[0] = (uint8_t)(random_number & 0x00ff);
26.
27.       topic_publish.type = MQTTSN_TOPIC_TYPE_NORMAL;
28.       topic_publish.data.long_.name = Topic.cstring;
29.       topic_publish.data.long_.len = strlen(Topic.cstring);
30.
31.       rc = openmqtt_publish(&topic_publish, &payload[0], sizeof(payload));
32.       if(rc == SUCCESS){
33.             printf("Exito!!! El valor enviado es %d\n", random_number);
34.       } else {
35.             openmqtt_disconnect();
36.       }
37. }
```

Esta función es la clave de la aplicación mrandom y se repite cada 3 segundos, según la definición del timer que se genera, con la que se pueden ver las principales funciones del interfaz, además del Topic y el Msg.

- Es importante destacar, que para generar el numero aleatorio se utilizan las siguientes librerías del sistema:

```
1.  #include "time.h"
2.  #include "stdlib.h"
```

Estas librerías se usan principalmente para estas líneas de código:

```
1.  uint8_t low_num = 0, hi_num = 101, rc = 0;
2.  srand(time(NULL));
3.  uint16_t random_number= (uint16_t)(rand() % (hi_num - low_num)) + low_num;
```

4.4 Aplicación Gateway de MQTT-SN.

Esta aplicación es una pasarela entre el protocolo MQTT-SN y el protocolo MQTT, por lo tanto pasa del protocolo de transporte UDP al protocolo TCP. Esto se consigue utilizando la librería de funciones paho.mqtt.c para el protocolo TCP que sería la parte encargada de conectar con el bróker mosquitto.

Y la otra librería clave seria paho.mqtt-sn.embeded-c que sería la encargada de conectarse con los clientes OpenMote. Para poder conectarse con los clientes OpenMote es necesario utilizar un socket ipv6 debido a que el openvisualizer abre un puerto tun simulando un interfaz Ipv6 que conecta con los Motes simulados.

Esta aplicación ejecuta un bucle infinito, que es que todo mensaje recibido de un tipo determinado lo procesa según el tipo de mensaje que llega, siendo necesario unas veces completar un dialogo en la zona de mqtt-sn y otras veces conectar con el broket en la zona de mqtt por tcp.

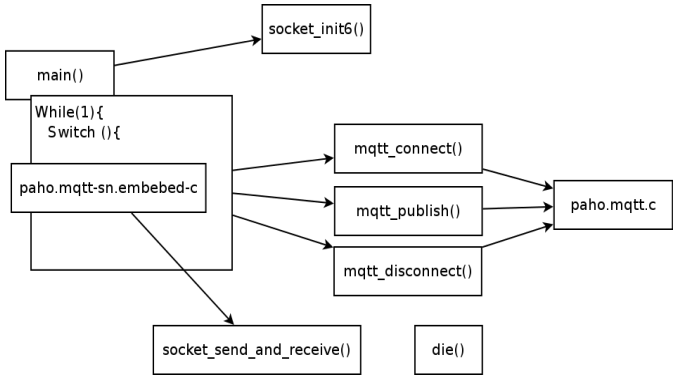


Ilustración 29: Diagrama de funciones de la aplicación gateway

La ilustración 28 representa un diagrama que muestra una aplicación que solo es capaz de gestionar de manera transparente la conexión de los clientes de MQTT-SN a MQTT, esto se consigue abriendo la escucha de un socket ipv6 en el puerto 61620 UDP.

Como estructura principal, esta aplicación tiene una zona de inicio de socket, variables y una zona con un bucle infinito que mediante la función switch, que funciona como filtro, es capaz de seleccionar la acción a realizar según el tipo de paquete de datos que llegue a través del socket.

El bucle infinito y la función switch que se han descrito anteriormente, hace uso de las librerías de paho.mqtt-sn.embebed-c para gestionar su funcionamiento, que consiste en serializar los datos en el cliente con la misma librería y des-serializado en el Gateway.

Según el tipo de mensaje que se recibe desde el cliente se realiza una acción en concreto, las acciones que es capaz de gestionar el Gateway con el protocolo de MQTT-SN son las siguientes:

- *CONNECT*: esta acción lo que recibe es el mensaje de conexión desde el cliente aportando el clienteID para que se pueda identificar inequívocamente tanto en el Gateway como en el bróker de MQTT. Automáticamente este mensaje envía el mensaje *WILLTOPICREQ* de requerimiento de TOPIC al cliente.
- *WILLTOPIC*: mensaje respuesta al *WILLTOPICREQ* cuyo contenido principal es el Topic que se necesita registrar en el bróker para que los Subscribers puedan recibir las publicaciones. Cuando la recepción es correcta el Gateway responde al cliente con el requerimiento del mensaje *WILLMSGREQ*.

- *WILLMSG*: dicho mensaje es recibido con el contenido necesario para la identificación en el bróker y en el Gateway. Cuando la recepción es correcta el Gateway realiza la conexión con el brocker utilizando la función mqtt_connect(), a través del protocolo MQTT. Según la respuesta de la función especificada se envía el mensaje *CONNACK* con el ack de respuesta del bróker.
- *DISCONNECT*: mensaje que envía automáticamente una desconexión al bróker a través de la función mqtt_disconnect(), la respuesta es reenviada al cliente según el código de respuesta con otro mensaje *DISCONNECT*.
- *PUBLISH*: el mensaje que lleva el contenido funcional de las aplicaciones y que deben ser enviadas al brocker para ser expuestas a los subscritores, cuando este mensaje llega es necesario que se des-serialice y mediante la función mqtt_publish() sea enviado al bróker cuya respuesta será enviada al cliente en el mensaje *PUBACK*.

Todos estos mensajes que se envían al bróker se gestionan con la librería paho.mqtt.c que funciona a través del protocolo TCP. Esta conexión hacia el bróker se configura por el puerto 1883 a la dirección de localhost, dado que se encuentra en la situación siguiente:

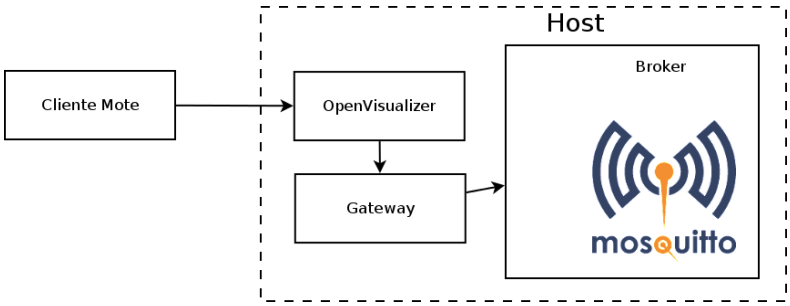


Ilustración 30: Diagrama de funcionamiento del gateway

Existen aplicaciones ya creadas que hacen la función de Gateway, pero se intentó probar su funcionamiento con el cliente dentro de la openmote y a través del openvisualizer pero no se consiguió hacerlo funcionar. Por lo tanto, decidí hacer una aplicación Gateway para poder llevar a cabo el proyecto. Esto supuso un aumento de las horas de trabajo técnico.

A continuación, se resume una breve descripción de las funciones que aparecen en el esquema de la ilustración 28:

- **int socket_init6(struct sockaddr_in6 *si_me):** inicializa un socket de ipv6 en el puerto 61620 del host.
- **void socket_send_and_receive(int *s, char *buf, struct sockaddr_in6 *si_other, int *slen, int *recv_len, bool send_receive):** se pone a la escucha y envía a través del socket ipv6 creado anteriormente un buffer.
- **int mqtt_connect(MQTTClient *client, MQTTClient_connectOptions *conn_opts, char * clientID):** realiza la conexión con el bróker con la ayuda de la librería paho.mqtt.c.
- **int mqtt_publish(MQTTClient *client, char *Topic, unsigned char *payload, int *payloadlen):** es la encargada de publicar al bróker toda la información que proviene del cliente OpenMote.
- **void die(char *s):** en caso de error de la aplicación, esta función se encarga de mostrar el error y salir de la ejecución.

- **int main(void)**: es la función principal de la aplicación, que se ejecuta nada más arrancar.

Los aspectos más destacados de la aplicación descrita en este apartado, son los siguientes:

- La aplicación se estructura como una función principal que mediante un bucle infinito, lee de un socket ipv6, con un procesado posterior, es capaz de realizar acciones en función del mensaje que le llega, lo que se describe en la sección de código siguiente:

```
1.   while(1){
2.         printf("Waiting for data...\n");
3.         fflush(stdout);
4.         //try to receive some data, this is a blocking call
5.         socket_send_and_receive(&s,buf,&si_other,&slen,&recv_len,true);
6.
7.         //print details of the client/peer and the data received
8.         printf("Received packet from %s:%d\n",
9.                inet_ntop(AF_INET6, &si_other.sin6_addr, straddr, sizeof(straddr)),
10.               ntohs(si_other.sin6_port));
11.
12.        len = MQTTSNPacket_decode(buf, recv_len, &datalen);
13.        rc=buf[len];
14.
15.        switch(rc){
16.        case MQTTSN_CONNECT:
17.             printf("Connect llego\n");
18.             len = MQTTSNDeserialize_connect(&data,buf,recv_len);
19.             len = MQTTSNSerialize_willtopicreq(buf,BUFLEN);
20.             socket_send_and_receive(&s,buf,&si_other,&slen,&len,false);
21.             break;
```

Tambien, muestra un caso dentro del switch que es el MQTTSN_CONNECT que des-serializa el mensaje, serializa y envía la respuesta.

- En el segmento de código anterior, existe una función que sirve tanto para recibir, como para enviar a través del socket ipv6 y que conecta con los motes, que es la siguiente:

```
1.  void socket_send_and_receive(int *s,
2.                               char *buf,
3.                               struct sockaddr_in6 *si_other,
4.                               int *slen,
5.                               int *recv_len,
6.                               bool send_receive){
7.     if(send_receive){
8.       if ((*recv_len = recvfrom(*s, buf, BUFLEN, 0, (struct sockaddr *)si_other, slen)) == -1){
9.          die("recvfrom()");
10.      }
11.    } else {
12.      if (sendto(*s, buf, *recv_len, 0, (struct sockaddr*)si_other, *slen) == -1){
13.         die("sendto()");
14.      }
15.    }
16. }
```

Esta función sirve para simplificar y facilitar el uso del socket en la aplicación, no obstante, es necesario mejorarla para que se pueda usar una transmisión asíncrona, porque son sockets bloqueantes y sincronos. No hace uso de funciones callback de recepción para dar carácter asíncrono al sistema.

- La inicialización del socket ipv6, ha sido separada a una función independiente, para que la función principal sea legible y sencilla, abstrayendo al desarrollador de ello, esta función es la siguiente:

```
1.  int socket_init6(struct sockaddr_in6 *si_me){
2.    int s = 0;
3.    //create a UDP socket
4.    if ((s=socket(PF_INET6, SOCK_DGRAM, 0)) == -1){
5.       die("socket");
6.    }
7.    // zero out the structure
8.    memset((char *)si_me, 0, sizeof(si_me));
9.    si_me->sin6_family = AF_INET6;
10.   si_me->sin6_port = htons(PORT);
11.   si_me->sin6_addr = in6addr_any;
12.   //bind socket to port
13.   if( bind(s,(struct sockaddr *)si_me, sizeof(*si_me) ) == -1){
14.      die("bind");
15.   }
16.   return s;
17. }
```

- Las funciones MQTT que se usan en el bucle infinito y en el switch de actuaciones, son las siguientes:

    - La primera es CONNECT, esta función realiza la conexión con el bróker en un socket TCP diferente del que ya se había creado.

```
1.  int mqtt_connect(MQTTClient *client,
2.                   MQTTClient_connectOptions *conn_opts,
3.                   char * clientID){
4.
5.       int rc = 0;
6.       MQTTClient_create(client, ADDRESS, CLIENTID,
7.                         MQTTCLIENT_PERSISTENCE_NONE, NULL);
8.       conn_opts->keepAliveInterval = 20;
9.       conn_opts->cleansession = 1;
10.
11.      if ((rc = MQTTClient_connect(*client, conn_opts)) != MQTTCLIENT_SUCCESS)
12.      {
13.            printf("Erro al conectar, codigo de retorno %d\n", rc);
14.            return rc;
15.      }
16.      return rc;
17. }
```

La conexión con broker se crea de manera transparente al usuario a través de la librería de MQTT.

    - La segunda función es PUBLISH, encargada de publicar el valor recibido del cliente OpenWSN al Broker correspondiente;

```
1.  int mqtt_publish(MQTTClient *client,
2.                   char *Topic,
3.                   unsigned char *payload,
4.                   int *payloadlen){
5.        int rc = 0;
6.        MQTTClient_message pubmsg = MQTTClient_message_initializer;
7.        MQTTClient_deliveryToken token;
8.
9.        pubmsg.payload = payload;
10.       pubmsg.payloadlen = *payloadlen;
11.       pubmsg.qos = QOS;
12.       pubmsg.retained = 0;
13.       MQTTClient_publishMessage(*client, Topic, &pubmsg, &token);
14.       rc = MQTTClient waitForCompletion(*client, token, TIMEOUT);
15.       return rc;
16. }
```

    - Por último, la función MQTT que es la función DISCONNECT, que realiza la acci de desconectarse de Broker:

```
1.  int mqtt_disconect(MQTTClient *client){
2.      int rc = 0;
3.      rc = MQTTClient_disconnect(*client, 10000);
4.      MQTTClient destroy(client);
5.      return rc;
6.  }
```

4.5 Aplicación bróker mosquitto.

Mosquitto es una de las aplicaciones bróker más importantes y ligeras del mercado de brókers de mqtt, es gratuita, desarrollada por eclipse, fácil de instalar y suele ser el centro de las topologías en estrella que ofrece MQTT. Esto supone un inconveniente, porque hay que buscar algún tipo de servicio de alta disponibilidad que actúe en caso de fallo del centro de la estrella reponiendolo. Esta característica no es exclusiva de mosquitto, ya que, todas las topologías de MQTT son de este tipo.

Mosquitto es instalable en multitud de plataformas y ofrece varios niveles de seguridad para que los clientes se conecten, publiquen y se suscriban. Es capaz de gestionar una gran cantidad de usuarios y subscriptores. A continuación, se muestra una tabla comparativa con algunos brokers, con las características principales que soportan:

Server	QoS 0	QoS 1	QoS 2	auth	bridge	$SYS	SSL	dynamic topics
Mosquitto	✓	✓	✓	✓	✓	✓	✓	✓
RSMB	✓	✓	✓	✓	✓	✓	✗	✓
WebSphere MQ	✓	✓	✓	✓	✓	✓	✓	✓
Apache Apollo	✓	✓	✓	✓	✗	✗	✓	✓
Apache ActiveMQ	✓	✓	✓	?	?	?	?	?
webMethods Nirvana Messaging	✓	✓	✓	§	✗	✗	✓	✗
RabbitMQ	✓	✓	✗	✓	✗	✗	✓	✓
MQTT.js	✓	✓	✓	§	✗	✗	✗	✓
moquette	✓	✓	✗	?	?	?	?	?

Ilustración 31: Tabla comparativa de brokers

En la tabla anterior se ve que algunas funcionalidades no son compartidas por todos los brokers, pero losl más completos son mosquitto y webSphere MQ. La ventaja de mosquitto es que es gratuito por lo que se eligió como la primera opción.

Es importante destacar que la configuración de mosquitto ha sido la que se presenta de serie, sin modificación alguna en el bróker para el funcionamiento del proyecto, es decir que el puerto que usa es el 1883, sin TLS, sin login y, por último, la comunicación usada es la QoS 0. Esta configuración es la más sencilla, que refuerza el propósito de este proyecto que es la integración del protocolo MQTT dentro de las plataformas OpenMote y el sistema operativo OpenWSN.

Sabiendo que mosquitto soporta varias plataformas de instalación, se elige la plataforma Linux y más concretamente DEBIAN, por lo que hay que seguir los siguientes pasos para la instalación del programa:

1. Descargar la llave del repositorio de instalación de mosquito:

   wget http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key

2. Se debe instalar la llave del repositorio para que cuando se actualice e instale permita acceder a él:

   sudo apt-key add mosquitto-repo.gpg.key

3. Dentro de la carpeta de repositorios del sistema se debe crear un archivo que contenga la dirección del repositorio que queremos utilizar para instalar mosquitto. La carpeta que necesitamos es "/etc/apt/sources.list.d/" y el comando que tenemos que utilizar es:

   sudo nano /etc/apt/sources.list.d/mosquitto.list

   En este archivo hay que añadir la siguiente línea:

   deb http://repo.mosquitto.org/debian <versión de debian> main

   Acto seguido se guarda el archivo y se cierra.

4. Ya se dispone el entorno para poder instalar el bróker mosquitto, por lo tanto, se ejecuta lo siguiente:

   sudo apt-get update

   Y a continuación:

   sudo apt-get install mosquitto mosquitto-clients

Con estos pasos la instalación de mosquitto en DEBIAN habría terminado, en los siguientes párrafos se incluyen algunos comandos para poder verificar el buen funcionamiento del bróker.

Para lo que hay que introducir los siguientes comandos:

1. El primer comando consiste en la creación del Topic de recepción de los mensajes, que en este caso se llamará test, indicando que la dirección de escucha es localhost:

   mosquitto_sub -h localhost -t test

2. Cuando ya se tiene en un terminal escuchando el Topic test se ejecuta el comando siguiente en otra terminal Shell distinta:

   mosquitto_pub -h localhost -t test -m "hello world"

   Este comando enviará un mensaje mqtt al bróker y si todo funciona correctamente aparecerá el mensaje "hello world" en la terminal que está abierta con el topic test escuchando.

Este bróker se ha integrado de manera correcta con la librería de desarrollo de mqtt paho.mqtt.c que se ha usado en el desarrollo del Gateway por lo que no ha sido necesario una búsqueda muy extensa de brokers, además de varias experiencias previas en el desarrollo de las aplicaciones, queda validada como mejor opción este bróker.

4.5 Aplicación suscriptor de MQTT.

Esta aplicación es un subscriptor de MQTT que se une al bróker usado en este caso mosquitto y a un Topic, esta aplicación se encuentra detrás del bróker y solo recibe y publica en una base de datos, no tiene una arquitectura interna muy complicada.

La única peculiaridad de esta aplicación es que la recepción de mensajes es asíncrona, por lo tanto, podría recibir una multitud de mensajes a la vez. Lo que se consigue con la configuración de las funciones de recepción de datos a través de la suscripción de forma Callback, es decir, cuando se produzca una variación en el socket de comunicaciones implícito a través de la librería de mqtt que existe entre el bróker y la aplicación subscriptora ejecuta una serie de funciones que permiten el tratamiento de la información.

Este tratamiento consiste en la recepción de la información, extraer los datos del mensaje MQTT, añade una marca de tiempo y el id del cliente que lo ha enviado. Justo después del tratamiento de la información se sube a una base de datos MYSQL, para que sea almacenado y mostrado por la aplicación correspondiente.

El diagrama de funciones de esta aplicación es el siguiente:

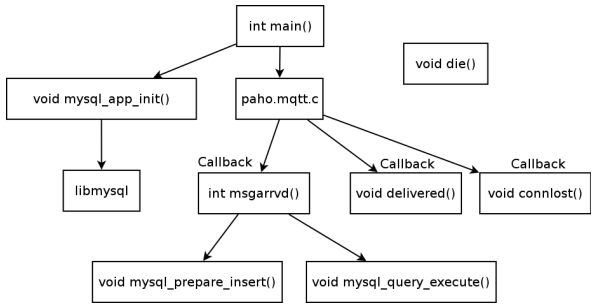


Ilustración 32: Diagrama de funciones de la aplicación subscriptora

A continuación, una breve descripción de las funciones que integran a la aplicación subscriptora:

- **int main(void):** es la función principal de la aplicación, que se ejecuta en primera instancia y que inicializa todo, tanto la conexión con la base de datos mysql, como las llamadas callback a través de la librería paho.mqtt.c.
- **void delivered(void *context, MQTTClient_deliveryToken dt):** informa cuando ha llegado un paquete nuevo desde la suscripción.
- **int msgarrvd(void *context, char *topicName, int topicLen, MQTTClient_message *message)**: gestiona el mensaje recibido del bróker.
- **void connlost(void *context, char *cause):** informa que la conexión con el bróker se ha caído, para que sea reestablecida.
- **void mysql_prepare_query_insert(char* query,char *buf):** se encarga de preparar la query para que sea ejecutada, básicamente extrae información del

paquete mqtt y añade una marca de tiempo. Esta función es llamada desde msgarrvd.

- **void mysql_app_init(void):** realiza la conexión con la base de datos. Es llamada desde la función principal.
- **void mysql_query_execute(char *query):** se encarga de ejecutar la query preparada anteriormente por otra función. Esta función se llama desde msgarrvd.
- **void die(char *s):** en caso de error de la aplicación, esta función se encarga de mostrar el error y salir de la ejecución.

Cabe destacar que esta aplicación tiene una recepción asíncrona gracias a la siguiente función:

MQTTClient_setCallbacks(client, NULL, connlost, msgarrvd, delivered);

Esta función de la librería paho.mqtt.c permite inicializar unas funciones Callbacks reactivas cuando se recibe alguna publicación a través del socket o si ocurre alguna incidencia ejecutarse de manera asíncrona.

Los principales aspectos del software son los siguientes:

- En esta sección de código se muestra la parte más importante de la función principal:

```
1.  // init mysql conexion and socket conexion
2.  mysql app init();
3.  sprintf(query,"TRUNCATE TABLE motes");
4.  mysql_query_execute(query);
5.  MQTTClient_create(&client, ADDRESS, CLIENTID,
6.                    MQTTCLIENT_PERSISTENCE_NONE, NULL);
7.  conn_opts.keepAliveInterval = 20;
8.  conn_opts.cleansession = 1;
9.  MQTTClient_setCallbacks(client, NULL, connlost, msgarrvd, delivered);
10. if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS){
11.       printf("Failed to connect, return code %d\n", rc);
12.       exit(EXIT_FAILURE);
13. }
14. printf("Subscribing to topic %s\nfor client %s using QoS%d\n\n"
15.      "Press Q<Enter> to quit\n\n", TOPIC, CLIENTID, QOS);
16. MQTTClient_subscribe(client, TOPIC, QOS);
17. do{
18.    ch = getchar();
19. } while(ch!='Q' && ch != 'q');
20. MQTTClient_disconnect(client, 10000);
```

En la que se inicializan las conexiones con la base de datos, además de las funciones callback de las librerías de MQTT, permitiendo recibir publicaciones desde el bróker de manera asíncrona.

Además de ello tiene una suscripción al brocker a un determinado TOPIC incrustado dentro del código mediante un DEFINE. También se puede observar que existe una función que ejecuta las querys que se le proporcionen.

- El siguiente segmento de código a destacar es la función que recibe la publicación, es decir, la función callback de recepción de datos:

```
1.  int msgarrvd(void *context, char *topicName, int topicLen, MQTTClient_message *message){
2.       int i;
3.       char* payloadptr;
```

```
4.          printf("Message arrived\n");
5.          printf("     topic: %s\n", topicName);
6.          printf("   message: ");
7.          payloadptr = message->payload;
8.          for(i=0; i<message->payloadlen; i++){
9.              putchar(*payloadptr++);
10.         }
11.         putchar('\n');
12.         payloadptr = message->payload;
13.         mysql_prepare_query_insert(query,payloadptr);
14.         mysql_query_execute(query);
15.         MQTTClient_freeMessage(&message);
16.         MQTTClient_free(topicName);
17.         return 1;
18. }
```

Esta función es la encargada de recibir, muestra por pantalla el mensaje recibido,y prepara la query INSERT para poder subir a la base de datos los valores que llegan junto con una marca de tiempo y la sube a la base de datos.

En cuanto al trabajo que se realiza en esta aplicación SUSCRIBER hacia la base de datos, no se ha considerado relevante destacarlo, ya que no trata sobre MQTT. Si se está interesado en el estudio de esta parte del código, se puede acudir a los anexos donde se ha reflejado el código completo de las aplicaciones que hemos desarrollado.

4.6 Aplicación base de datos MYSQL.

Es un sistema de gestión de base de datos relacional comercializado por Oracle, considerado como el sistema de código abierto de base de datos más popular del mundo y uno de los más populares junto con Oracle y Microsoft SQL Server.

Este sistema de gestión tiene un servidor de gestión que se llama mysql-server que tiene fácil instalación en cualquier plataforma, más adelante se indica cómo se instala este servidor para que se pueda ejecutar este proyecto en cualquier ordenador con un DEBIAN instalado.

Para poder definir una base de datos en este sistema es necesario hacerlo en el lenguaje de programación SQL que es un lenguaje propio de este tipo de servidores de datos relacionales que ayuda en la manipulación de los datos que contienen los servidores.

Por lo que se ha tenido que realizar un pequeño script en SQL para que se cree la base de datos relacional que necesitamos para el proyecto. A continuación, vamos a detallar el proceso de instalación del servidor en un ordenador con Linux, en nuestro caso DEBIAN: (Todo desde una cuenta de root, si no tenemos que añadir sudo delante de los comandos)

1. Es necesario descargar el paquete que instala el repositorio de mysql en DEBIAN:

   wget https://dev.mysql.com/get/mysql-apt-config_0.8.6-1_all.deb

2. Lo siguiente es instalar el paquete te instalación gdebi:

   apt-get install gdebi-core

3. Acto seguido instalamos el paquete con la herramienta anterior:

   gdebi mysql-apt-config_0.8.6-1_all.deb

4. Continuando se instala el servidor mysql-server, porque en los anteriores comandos habíamos instalado el repositorio de mysql para la instalación y futuras actualizaciones:

   apt-get update && apt-get install mysql-server

Con estos comandos realizados, se tiene disponible el servidor de bases de datos mysql-server, en el que podemos entrar de la siguiente manera:

mysql –u root –p

Se introduce la contraseña definida en la instalación entrando en el servidor pudiendo ejecutar el script siguiente:

```
CREATE DATABASE mrandom;
use mrandom;
CREATE TABLE motes (id int NOT NULL AUTO_INCREMENT,
                    number int NOT NULL,
                    fecha DATETIME,
                    PRIMARY KEY(id));
```

Ilustración 33: script mysql que genera base de datos.

El anterior script SQL pertenece al archivo mysql_dabase_mrandom.sql, este script se encuentra por la rama de la aplicación mrandom en el nivel 3, esto se puede ver en la ilustración 24 de este proyecto. La forma de ejecutar este script es la siguiente:

mysql –u root –p < mysql_dabase_mrandom.sql

Si no ocurre ningún error es que se ha instalado correctamente, el único error que puede existir es que la base de datos este ya definida, por lo tanto, tenemos que borrarla antes, eso lo haremos con estos comandos:

1. En primer lugar, entramos en el servidor de mysql:

   mysql –u root –p

2. Cuando estas dentro debes ejecutar la query mysql siguiente:

   DROP DATABASE mrandom;

   Justo después de esto ejecutar un exit.

4.7 Aplicación que ofrece un dashboard (Grafana).

Grafana es una aplicación abierta que ofrece una plataforma agradable para la monitorización y el análisis. Con esta plataforma representaremos todos los datos que llegan desde el cliente a la base de datos. Para poder configurar grafana es necesario una fuente de datos y que esta fuente de datos tenga una marca de tiempos para que grafana pueda ordenar los datos de manera sencilla.

En nuestro caso la fuente de datos es una base de datos que tiene un parámetro o valor aleatorio de 0 a 100 junto con una marca de tiempo. Por lo tanto, podemos mostrar una gráfica.

Ilustración 34: Grafica resultado de la arquitectura en grafana

En las siguientes líneas, se explica cómo poder instalar y preparar el dashboard que se ha desarrollado para esta arquitectura, estos son los pasos necesarios: (todo realizado desde la terminal de root)

1. El primer paso es añadir el repositorio a la carpeta /etc/apt/source.list.d/ con los comandos siguientes:

   nano /etc/apt/sources.list.d/grafana-repo.list

   y se añade la siguiente línea:

   deb https://packagecloud.io/grafana/stable/debian/ stretch main

2. El siguiente paso es la validación de la key del repositorio:

   curl https://packagecloud.io/gpg.key | sudo apt-key add -

3. Por último, queda poder instalar la herramienta con los dos siguientes comandos:

   apt-get update
   apt-get install gafana

Cuando se han completado los pasos anteriores tenemos la herramienta grafana instalada, lista para instalar el dashboard que hemos desarrollado para esta arquitectura. Para poder acceder al portal grafana es necesario introducir en la barra de direcciones del navegador http://localhost:3000 esto dará acceso al siguiente portal:

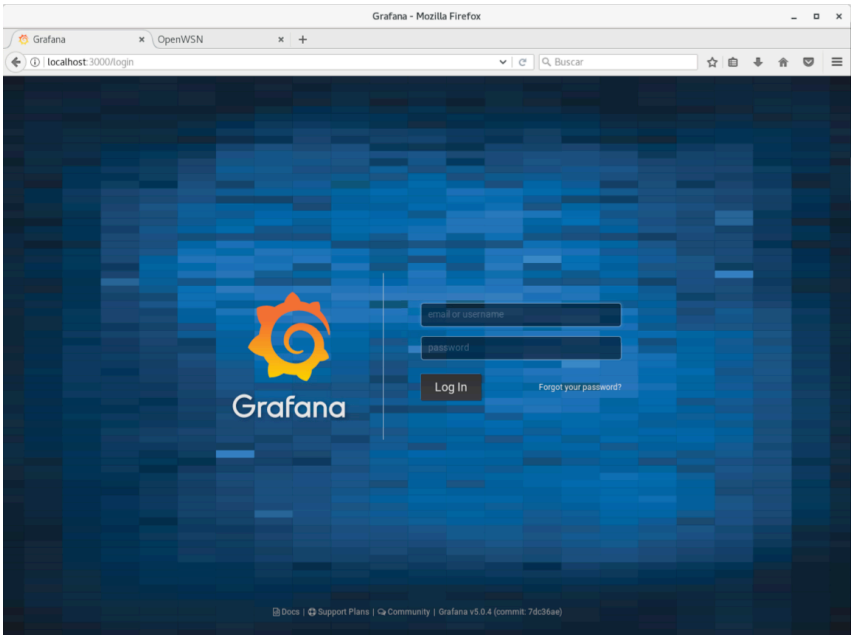


Ilustración 35: Portal grafana

La contraseña por defecto y el usuario son admin admin respectivamente, con eso se puede acceder a la configuración del entorno, para ponerlo a funcionar es necesario definir un datasource en primer lugar.

En este caso hay que definir el datasource del tipo mysql, porque es lo que se está utilizando para guardar los datos con los que se va a trabajar, y para ello se realiza de la siguiente manera:

1. Se accede al portal, apareciendo lo siguiente:

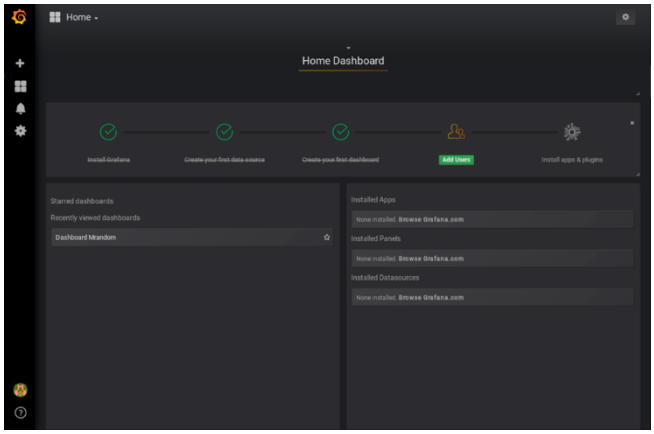


Ilustración 36: home de grafana después del login

2. Se accede al menú de datasource que aparece cuando se pulsa sobre el piñón que aparece a la derecha:

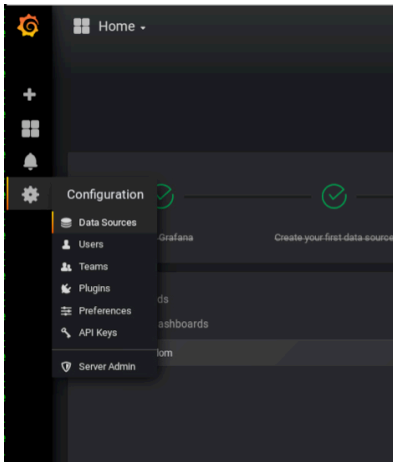


Ilustración 37: Menú de configuración, seleccionamos datasource

3. En el menú datasource hay que configurar uno del tipo mysql:

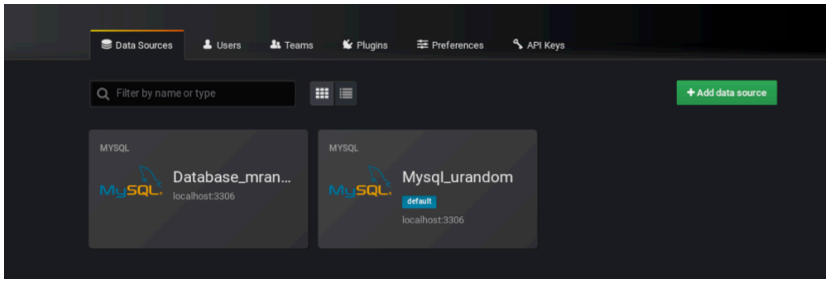


Ilustración 38: Gestión de las fuentes de datos

Como se ve en la ilustración 38, hay varios datasources configurados del tipo mysql, pues es necesario hacer lo mismo con el botón add data source verde que aparece a la derecha de la imagen. Cuando se pulsa hay que configurar el datasource como mysql y la configuración debe de ser de esta forma:

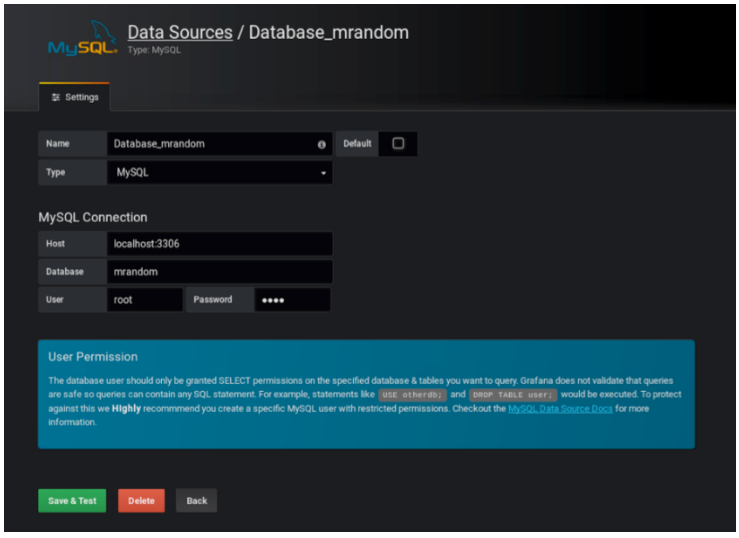


Ilustración 39: configuración del datasource

4. Cuando se tiene el datasource definido es momento de crear el dashboard, para ello se utiliza un archivo json que está preparado en el directorio de nivel 3 en la carpeta de mrandom, se entra en este menú:

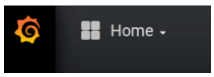


Ilustración 41: menú desplegable de los dashboards

Una vez pulsado el botón del menú, que se encuentra arriba a la izquierda justo al lado del home, simbolizado como una flecha hacia abajo, muestra lo siguiente:

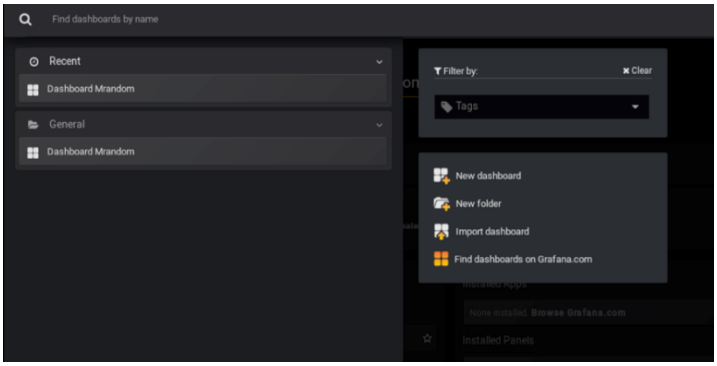


Ilustración 40: Menú de manipulación de dashboards

5. Estando en la pantalla anterior se debe pulsar sobre el botón import dasboard, el cual lleva a la siguiente pantalla:

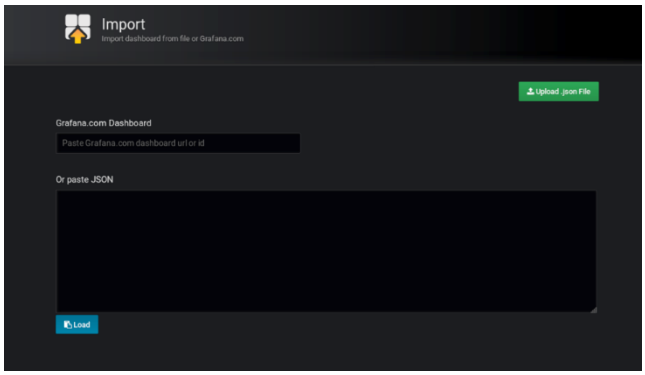


Ilustración 42: Pantalla de Importación de dashboard

Se pulsa sobre el botón de upload json file, y hay que subir el archivo dashboard_motes.json que se encuentra en el tercer nivel de la rama de mrandom según la estructura de archivos que se ha definido en el tema.

Después de ejecutar todos estos pasos se dispone de un dasboard necesario para el proyecto. Para poder especificar mejor el dashboard a continuación se explica la query que contiene el dashboard para funcionar.

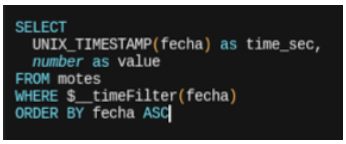


Ilustración 43: Query de grafana

Gracias a esa Query, que muestra el valor en función a la fecha y ordenado por ella permite dibujar una gráfica en función del tiempo.

5. Detalles de las pruebas

A continuación, se detallan las pruebas especificadas para poder validar la arquitectura, para ello se usa el analizador de paquetes wireshark, que se puede instalar directamente desde un repositorio de DEBIAN, con el siguiente comando:

```
apt-get install wireshark
```

Con el programa instalado, es necesario ejecutarlo como root. Cuando este programa es ejecutado, detecta todos los interfaces de red que están activos en el equipo, esto se hace para poder seleccionar el interfaz donde sniffar paquetes de datos.

Por eso es necesario ejecutar el programa openvisualizer para poder observar el interfaz "tun" simulando un interfaz de red conectado a la red de motes simulada. En este momento el firmware de los motes es ejecutada y tal y como está diseñada la aplicación permanece esperando y mandando mensajes connect hasta que el Gateway es detectado.

Con todo iniciado, comienza el análisis de los mensajes para poder validar esta arquitectura. Para ello se van a utilizar los diagramas de secuencia expresados en el punto 3.1 de esta memoria del proyecto.

5.1 Análisis de los mensajes entre el firmware y el Gateway

Para desarrollar este punto se va a utilizar la ilustración 13, que nos muestra una serie de mensajes entre los integrantes definidos en el título de este punto, utilizando un análisis mediante el analizador de paquetes y utilizando el filtro siguiente:

```
ipv6.src=bbb:::1 || ipv6.src= bbbb::1415:92cc:0:2
```

Con lo que se comienza analizando mensajes de la secuencia de conexión:

- CONNECT: En este mensaje se envía el id de cliente, que en el ejemplo utilizado en el firmware sería el siguiente "d:mqttsn:openwsn:1" para poder verificar el envío del mensaje, hay que buscar la traza correspondiente a este mensaje y en el programa se encuentra lo siguiente:

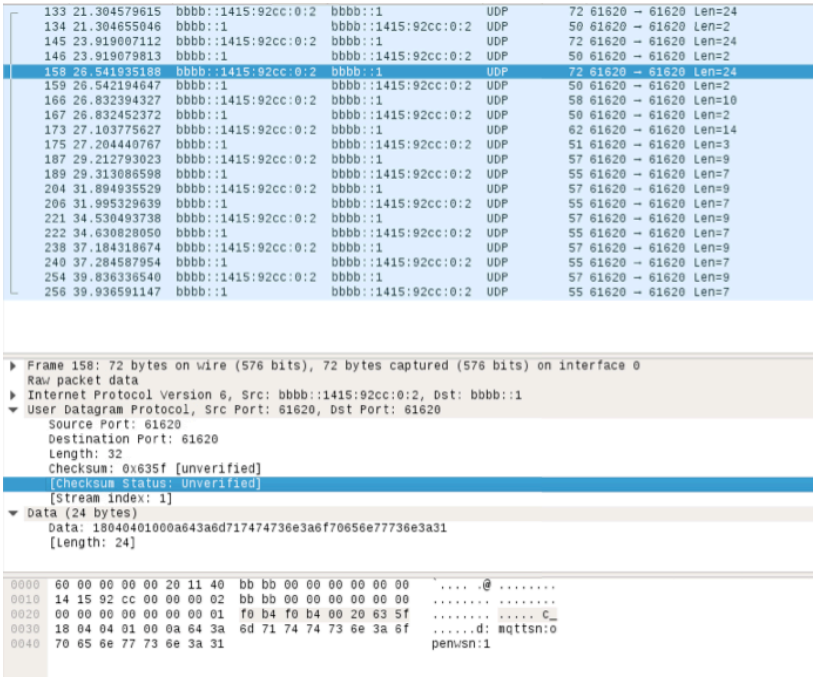


Ilustración 44: Mensaje connect

En la ilustración 44 se puede ver que es un paquete UDP con un contenido serializado que corresponde con lo que se ha querido enviar en el contenido desde el firmware.

- El siguiente mensaje a analizar es WILLTOPICREQ, hay que buscar el siguiente mensaje en las capturas de las trazas. El valor en hexadecimal del mensaje WILLTOPICREQ es 02x0 06x0.

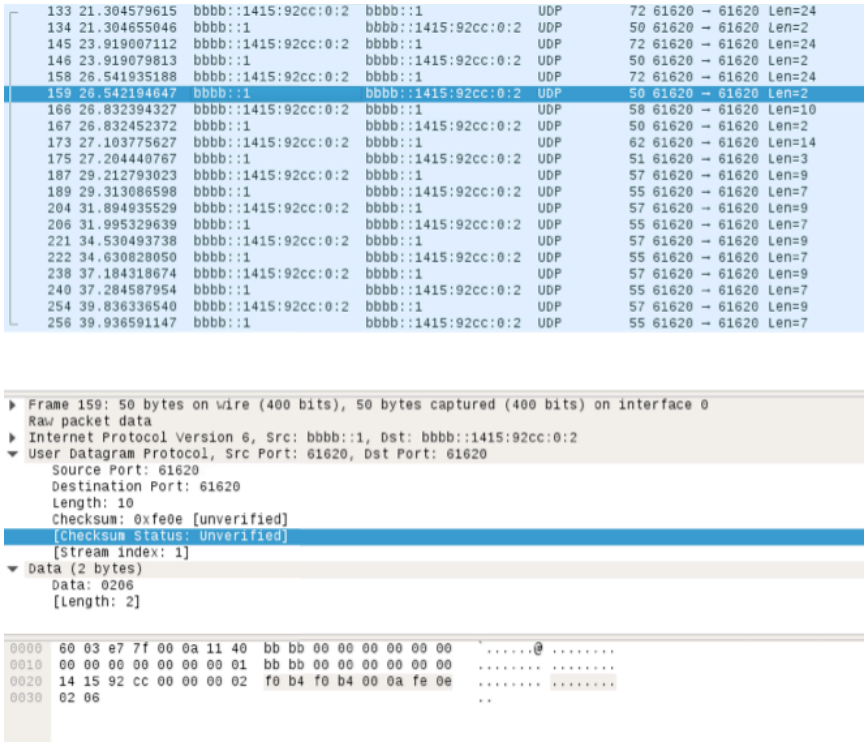


Ilustración 45: Mensaje WILLTOPICREQ

Se puede constatar según la ilustración 45 que el mensaje WILLTOPICREQ se envía correctamente y justo después de que el mensaje CONNECT se envíe desde el firmware.

- El mensaje WILLTOPIC es el siguiente mensaje a buscar para poder validar la arquitectura, sabiendo que el Topic que se envía es “Pruebas”, la siguiente ilustración que muestra el resultado:

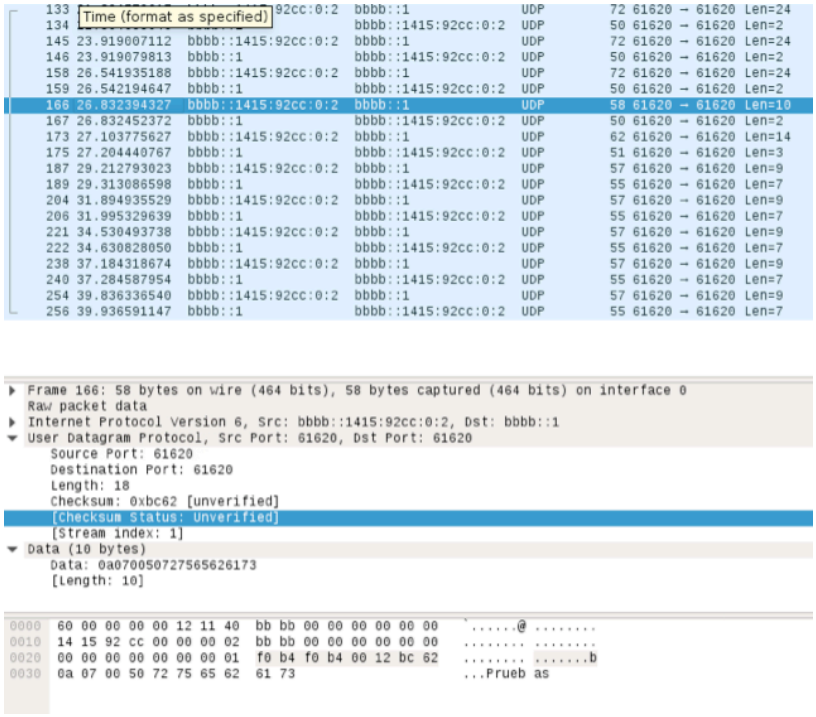


Ilustración 46: Mensaje WILLTOPIC

El mensaje consiste en una serialización del Topic en él, y cómo se puede observar en la ilustración 46 aparece la palabra pruebas en el mensaje UDP.

- Cuando el mensaje anterior llega al Gateway correctamente, se obtiene el mensaje WILLMSGREQ, que es una respuesta automática pidiendo el mensaje del Topic, teniéndolo que buscar en hexadecimal es 02x0 08x0:

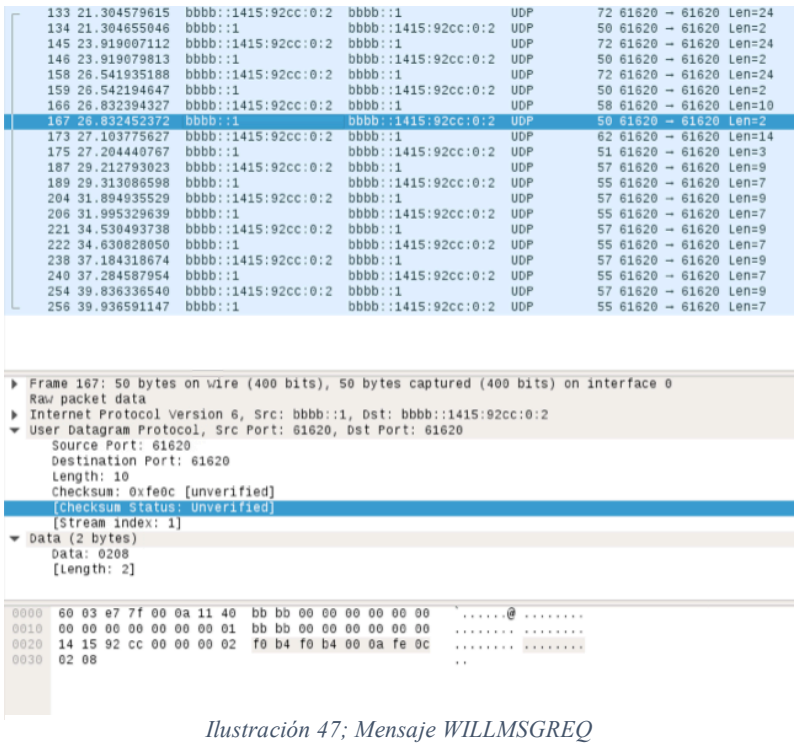


Ilustración 47; Mensaje WILLMSGREQ

En la ilustración 47 en el campo de datos se pueden ver los valores enviados en el mensaje en hexadecimal se corresponden con lo que queremos enviar desde el Gateway al firmware.

- El último mensaje que lanza el firmware como respuesta a WILLMSG, este mensaje lo tendrá serializado del firmware hacia el Gateway, este mensaje es "MQTTSNClient"

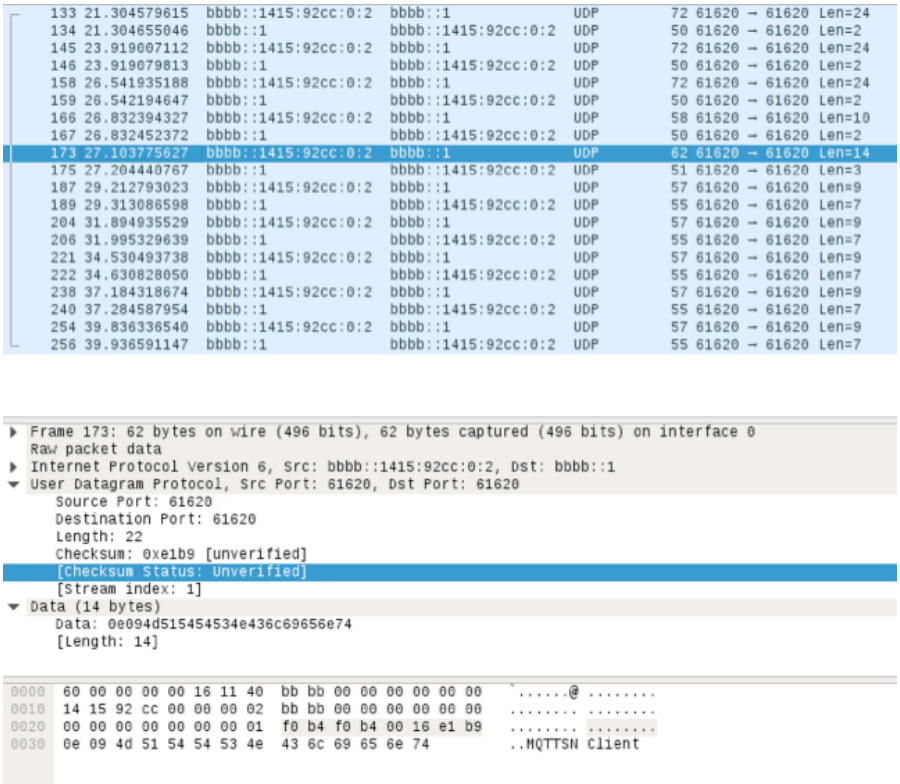


Ilustración 48: Mensaje WILLMSG

En la ilustración 48 se puede observar que en el mensaje se serializa MQTTSNClient correctamente. A continuación, se enviaría el CONNECT desde el Gateway al Broker, esperando la respuesta.

- Por último y para terminar la secuencia de conexión entre el firmware y el Gateway, se envía después de WILLMSG un mensaje desde el Gateway al firmware, es el mensaje CONNACK que consiste en un mensaje que retransmite el ACK que se recibe desde el Broker permitiendo o no la conexión MQTT.

  En la captura siguiente se observa dentro del paquete en los datos 02x0 05x0 que corresponde con la cabecera del mensaje CONNACK y luego el valor del mensaje CONNACK que en este caso debería ser 00x0 para que signifique que la conexión con el bróker ha sido aceptada. Si devuelve algún valor distinto es que no ha sido aceptada la conexión.

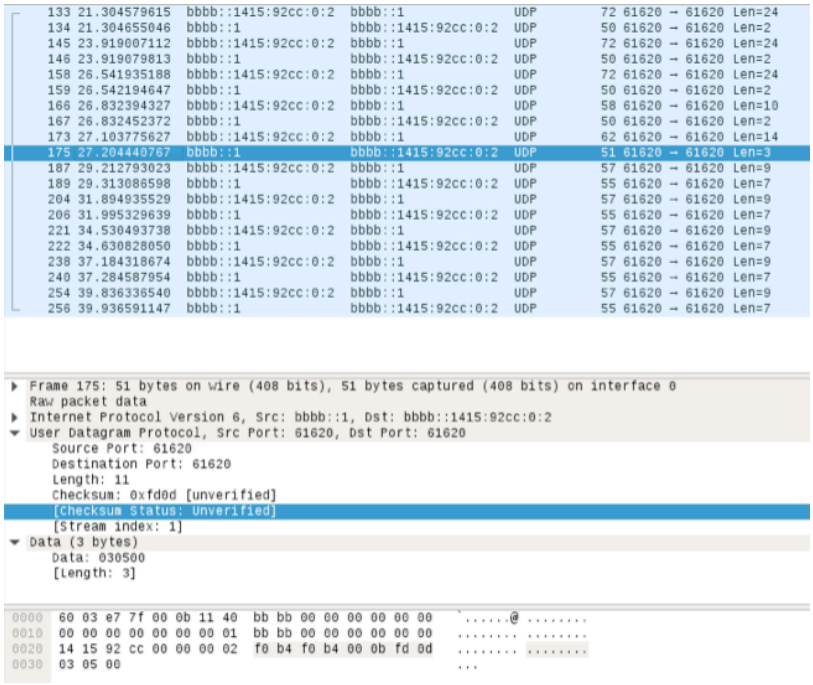


Ilustración 49: Mensaje CONNACK

El paquete enviado da un resultado correcto y ha sido aceptado por el bróker. Con este mensaje se concluye la secuencia de mensajes de conexión, después se continúa con los mensajes de publicación.

Una vez especificados y validados todos los mensajes de la secuencia de conexión hay que continuar con los mensajes de publicación y su ACK:

- El siguiente mensaje es PUBLISH que consiste en enviar una cadena de datos serializada con un valor entero entre 0 y 100, lo que se refleja en la siguiente ilustración:

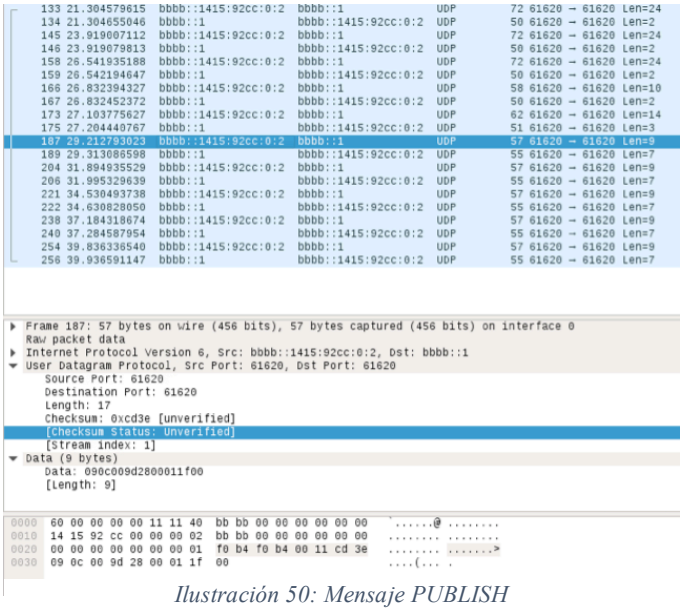


Ilustración 50: Mensaje PUBLISH

El paquete de datos lo siguiente: 09 0c 00 9d 28 00 01 1f 00.

Disgregando el mensaje PUBLISH:

- Octeto 0: Es la longitud del mensaje, en este caso es 09.
- Octeto 1: Es el tipo de mensaje, en este caso 0c.
- Octeto 2: Son los flags del mensaje, en este caso 00.
- Octeto 3-4: Es el TOPIC ID, en este caso 9d 28.
- Octeto 5-6: Es el MSG ID en este caso es 00 01.
- Octeto 7-n: Es el valor serializado que es 1f 00.

Sabiendo que el valor 00 es la parte de mayor peso del paquete de datos y el 1f la parte inferior y sabiendo que se envía un entero, se puede deducir que el entero enviado es 31. Por eso podemos afirmar que el paquete de datos ha sido enviado correctamente y validado.

- El último mensaje a validar dentro de la secuencia es el mensaje PUBACK, este mensaje se corresponde con la ilustración siguiente:

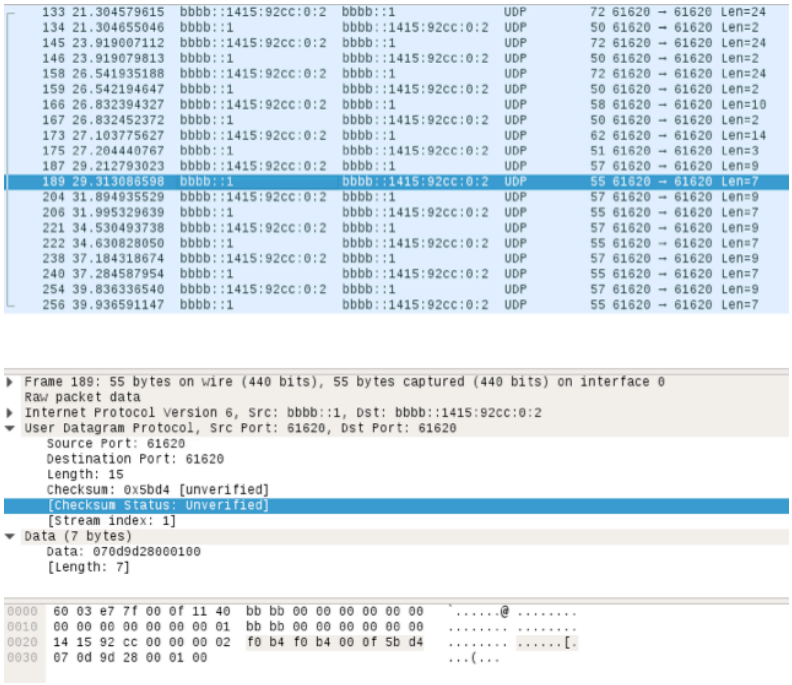


Ilustración 51: Mensaje PUBACK

Este mensaje contiene lo siguiente: 07 0d 9d 28 00 01 00

Desgranando este mensaje tenemos lo siguiente:

- Octeto 0: Es la longitud del mensaje, en este caso es 07.
- Octeto 1: Es el tipo de mensaje, en este caso 0d.
- Octeto 2-3: Es el TOPIC ID, en este caso 9d 28.
- Octeto 4-5: Es el MSG ID en este caso es 00 01.
- Octeto 6: Es el return code, en este caso el valor es 00, proviene retransmitido desde el Broker diciendo que ha sido aceptado.

Con la descripción de estos mensajes dentro del protocolo MQTT-SN se termina la sección de la comunicación entre el firmware y el Gateway.

5.2 Análisis de los mensajes entre el Gateway y el Broker

Estos mensajes se van a producir solamente en el interfaz loopback ya que se comunica el Gateway con un Broker que están en el mismo equipo, este protocolo no es una variante de mqtt si no que es el protocolo original. Para ello el filtro introducido es "mqtt" en la barra de filtros de wiresharck.

Los mensajes van a ser enviados desde el Gateway después de que la secuencia de conexión del Gateway se complete, en este caso el protocolo contiene menos mensajes a describir porque es un protocolo basado en TCP y está orientado a la conexión.

Se validan dos secuencias, la secuencia de conexión y la secuencia de publicación. Por lo tanto, se comienza con la secuencia de conexión:

- El mensaje CONNECT, que figura en la siguiente ilustración donde se pueden ver las características del mensaje:

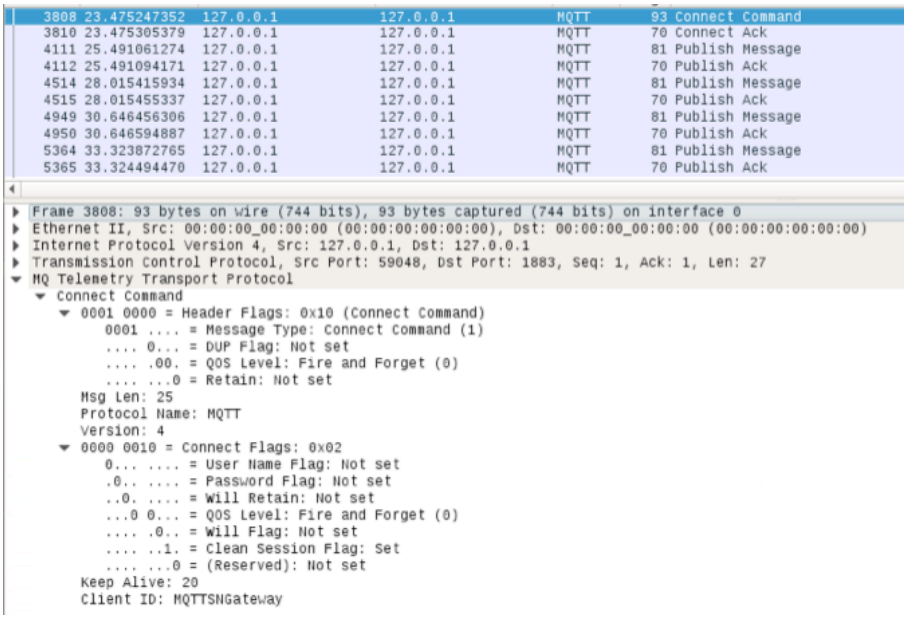


Ilustración 52: Mensaje CONNECT

El protocolo MQTT, wireshark ofrece un análisis exhaustivo del paquete de datos MQTT, en él se puede ver como el Cliente MQTTSNGateway quiere conectar con el Broker, el siguiente mensaje analizado será la respuesta del Broker al Gateway.

- El mensaje de ACK con un result code desde el Broker, es el resultado retrasmitido al firmware a través del CONNACK. En la siguiente ilustración se muestra el contenido de este mensaje:

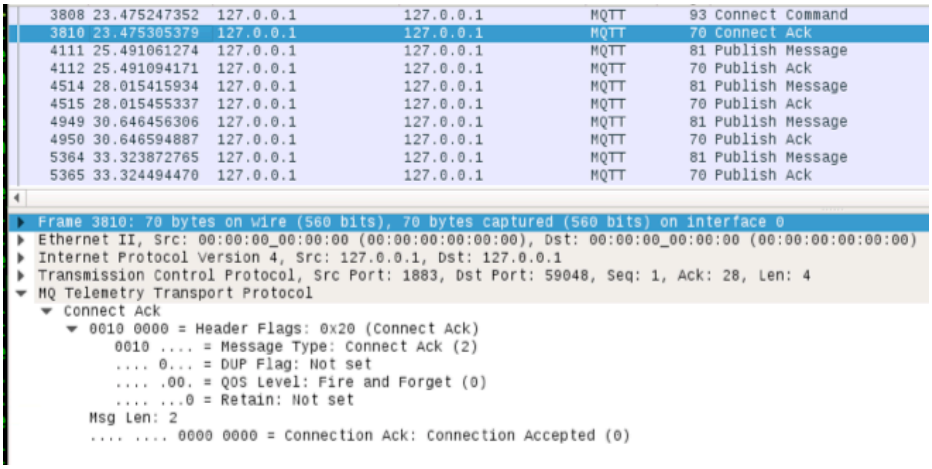


Ilustración 53: Mensaje ACK del Broker

Este mensaje como queda descrito en la ilustración 53, responde con un mensaje de aceptación.

Con esos dos mensajes se termina la secuencia de conexión, en este protocolo distinto al anterior y que esta entre el firmware y el Gateway se observa que la secuencia de conexión es mucho más sencilla. A continuación, se valida la secuencia de publicación desde el Gateway al Broker:

- El mensaje PUBLISH se muestra en la siguiente ilustración:

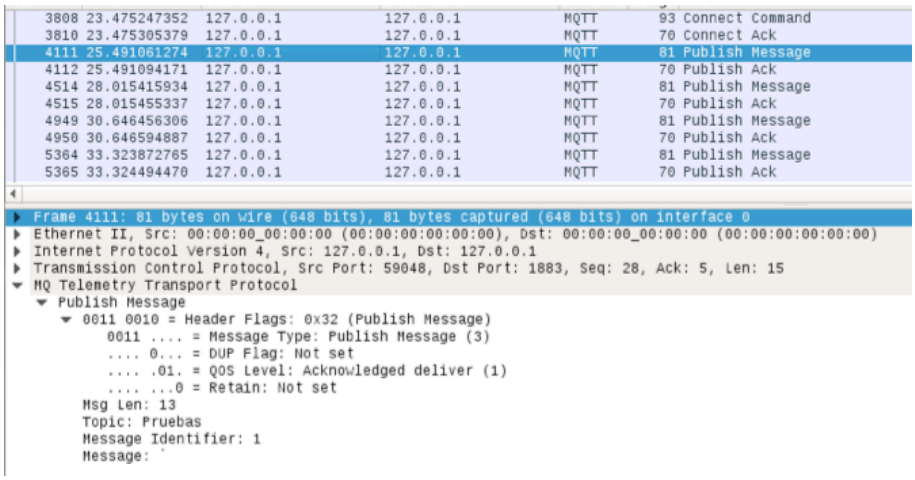


Ilustración 54: Mensaje PUBLISH MQTT

El mensaje PUBLISH se describe de manera muy detallada en la ilustración 54, quedando claro que el mensaje va dirigido al Topic “Pruebas”, teniendo un ID de mensaje 1 y como contenido del mensaje “ ‘ ”, pero realmente es un valor en hexadecimal que en este caso es el 60 00 que traducido a decimal es 96. Pero ¿Porque sale reflejado ese valor en wireshark? Es porque 60 traducido a una tabla ASCII corresponde con ese texto.

- Mensaje ACK respuesta del PUBLISH:

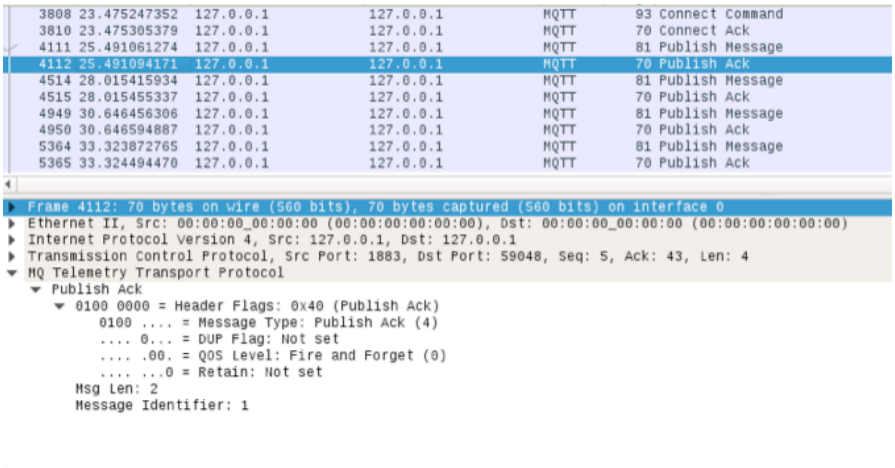


Ilustración 55: Mensaje ACK de PUBLISH MQTT

Este mensaje en sí mismo refleja una respuesta afirmativa de publicación, por lo tanto, devuelve el identificador de mensaje nada más.

La comunicación entre el Gateway y el Broker queda validada porque se produce todo tal y como se ha planificado en los criterios de validación del tema 3, verificando los mensajes que deseamos tener en la comunicación.

5.3 Análisis de los mensajes entre el Broker y el Suscriptor MQTT

Esta sería la última secuencia de mensajes a validar que consiste en 4 mensajes que concentran dos secuencias de comunicación. Estas secuencias son la secuencia de suscripción y la secuencia de publicación desde el bróker. La secuencia será activada cuando el firmware conecta con el bróker ya que este empezará a publicar después de que se conecte el cliente MQTT-SN. Es necesario destacar que estos mensajes estarán mezclados con los mensajes que se producen desde el Gateway por lo que hay que poder diferenciarlos.

Primero se describe la secuencia de Suscripción:

- El primer mensaje es el mensaje CONNECT, este mensaje es necesario que se produzca desde el suscriptor para que sea identificado en el bróker como interlocutor en esta arquitectura por lo tanto es necesario especificarlo, pero no se considera suficiente para aportar una validación en esta parte de la arquitectura.

  Este mensaje es importante porque identifica el puerto de comunicaciones que va a utilizar el suscriptor hacia el bróker, lo que permitirá identificar los mensajes que son del Gateway de los mensajes que provienen del suscriptor.

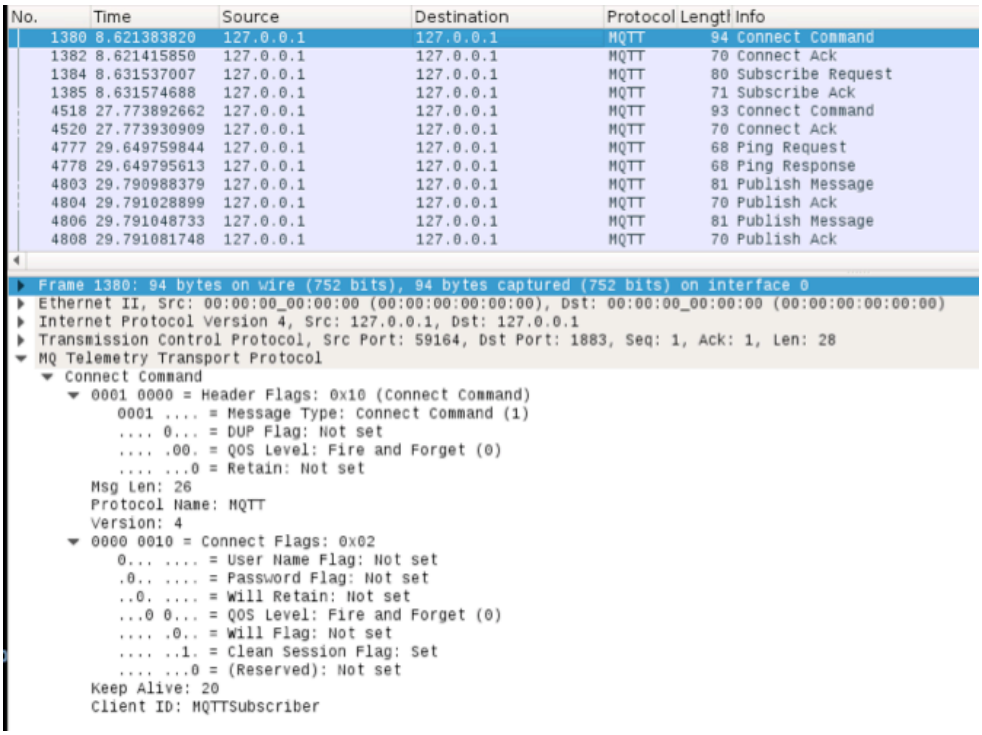


Ilustración 56: Mensaje CONNECT Suscriptor MQTT

La ilustración 56 muestra el puerto que se va a utilizar para conectarse al bróker y recibir todas sus publicaciones, este puerto es 59164.

- Seguidamente a este mensaje se envía un ACK de respuesta al mensaje CONNECT:

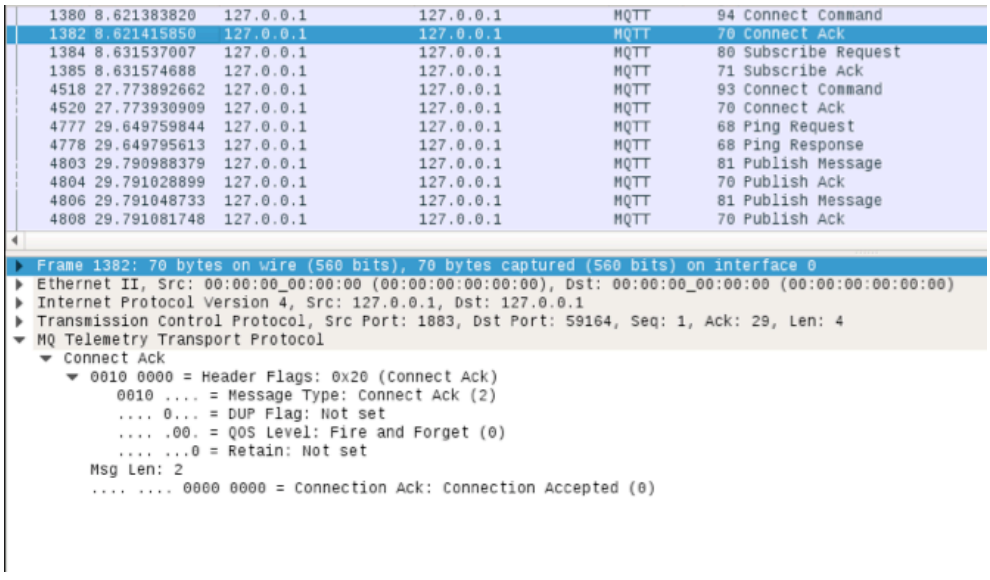


Ilustración 57: Mensaje ACK

La conexión del suscriptor ha sido aceptada, los siguientes mensajes son los que se utilizan para validar la comunicación entre el Broker y el suscriptor.

- El mensaje importante de la secuencia de conexión es el mensaje de SUSCRIBE, que se muestra en la siguiente ilustración:

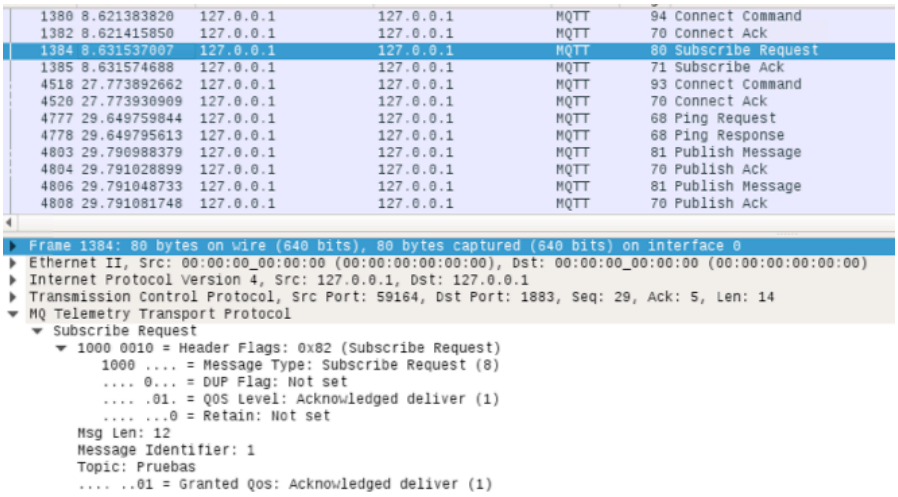


Ilustración 58: Mensaje SUSCRIBE

En la ilustración 58, se observa el Topic de suscripción "Pruebas" para poder empezar a recibir publicaciones desde el Broker, esto es lo esperado para poder cerrar el círculo y poder insertar los datos en la base de datos.

- Poco después se espera un ACK desde el Broker aceptando la conexión:

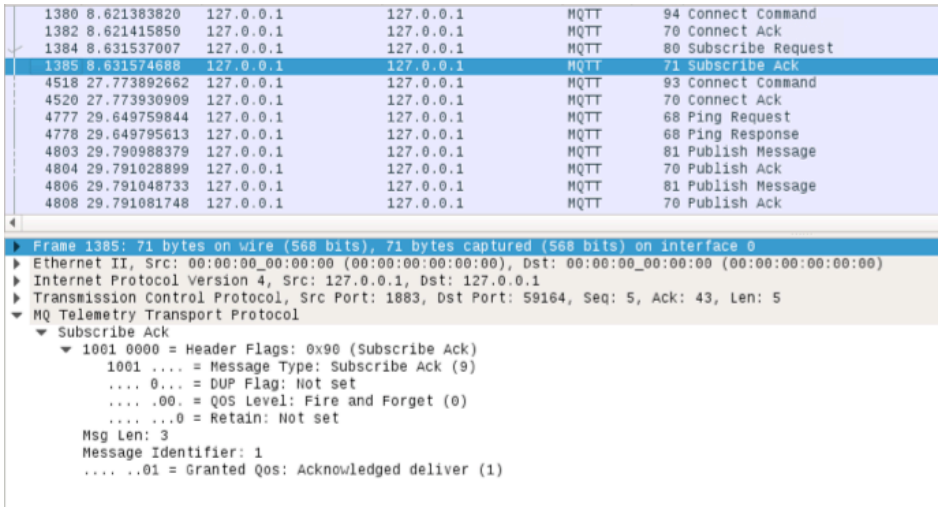


Ilustración 59: Mensaje ACK Suscriber

Una vez aceptada la conexión, en la ilustración 59 vemos que hay otro CONNECT dentro de los mensajes. Esto es porque comparten interfaz el Gateway y el Suscriptor, pero se puede diferenciar la comunicación a través del puerto reflejado en el mensaje CONNECT de esta secuencia.

En esta captura y en las anteriores, se muestra un mensaje de PING hacia el Suscriptor antes de enviarle una publicación, pero como es algo inherente al protocolo, se considera que no es necesario utilizar para validar el protocolo de estos mensajes, Es más importante que una publicación llegue con éxito al suscriptor de manera correcta.

A continuación, la siguiente secuencia es la secuencia de publicación. Esta publicación se produce desde el Broker hacia el suscriptor:

- El mensaje PUBLISH:

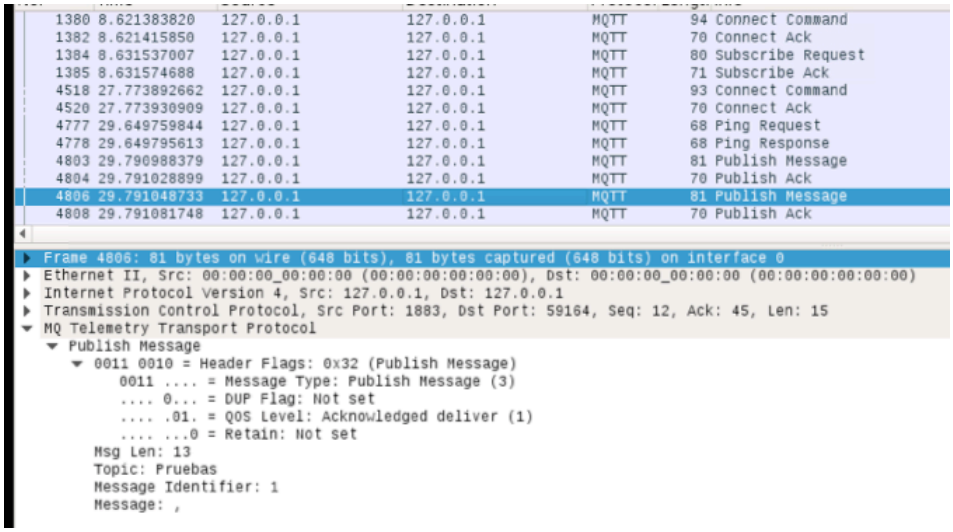


Ilustración 60: Mensaje PUBLISH mqtt Suscriber

En el detalle de este mensaje, se ve cómo se envía al Topic "Pruebas" y un valor ASCII que corresponde con un número entero. Queda validada la comunicación porque se envía lo que lo esperado.

- El mensaje ACK hacia el Broker:

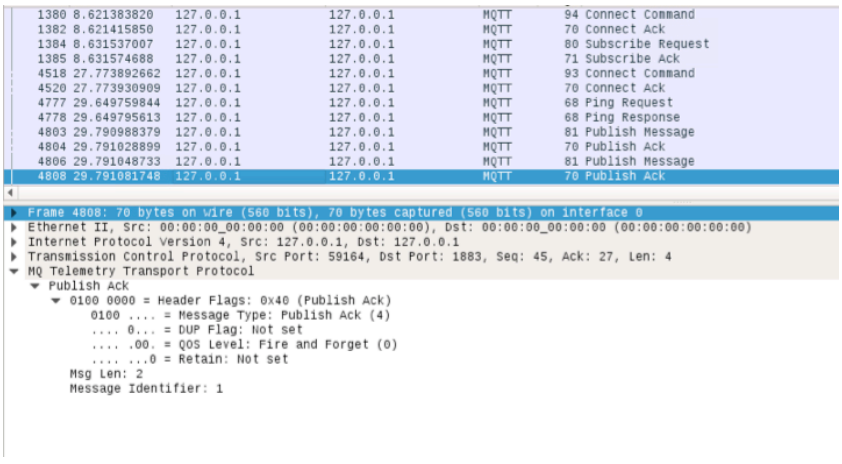


Ilustración 61: Mensaje ACK mqtt Suscriber

Este mensaje es la confirmación de que el mensaje publicado llega correctamente a su destino, informando al bróker que puede mandar el siguiente mensaje.

Con esto se termina el estudio de la secuencia de mensajes de las diferentes aplicaciones con el protocolo MQTT. Esta última aplicación, utilizaría los datos publicados para subirlos a una base de datos MYSQL, pero eso quedaría validado mediante la visualización de los datos reflejados en Grafana.

6. Conclusiones.

Después de una validación del protocolo, se puede afirmar que la arquitectura funciona correctamente como primer paso para el desarrollo de aplicaciones en el sistema operativo OpenWSN. Se puede por tanto afirmar, que esté justificado su uso frente a otros protocolos de comunicaciones o diferentes maneras de trabajar con este sistema operativo en base a las ventajas que han quedado patentes en el estudio y que se describen seguidamente.

Del estudio realizado de ventajas e inconvenientes de esta forma de trabajar, describimos las obtenidas con el desarrollo de estas librerías para el protocolo de MQTT.

Las ventajas verificadas son las siguientes:

- La principal ventaja es la facilidad de desarrollo de las aplicaciones de MQTT, dado que solo es necesario usar las funciones de la librería openmqtt.
- Este protocolo permite poder enviar mensajes de tamaño mínimo, 2 Bytes, por lo tanto, ofrece una mayor capacidad de canal, ya que permite mandar una cantidad de mensajes mayor.
- Y dada la gran capacidad de canal que tiene este protocolo, se puede afirmar que se puede trabajar con un gran número de clientes simultáneamente, gracias a las capacidades asíncronas de este protocolo.
- Capacidad de poder disgregar la comunicación de los clientes en diferentes Tópicos de publicación, pudiendo tener diferentes redes de sensores con diferentes propósitos sobre un mismo canal.
- Con respecto a la integración del protocolo, ha sido, dentro de la dificultad de crear el propio interfaz dentro del sistema operativo OpenWSN, sencillo porque el propio protocolo ofrece variantes del mismo para que se ajuste a los diferentes casos de uso que puede tener.

Y los inconvenientes verificados, son las siguientes:

- Debido a la utilización del protocolo MQTT-SN (variante de MQTT) es necesario utilizar un Gateway intermedio que traduzca de protocolo de transporte UDP a TCP.
- Además del Gateway es necesario tener un Broker MQTT para poder establecer la comunicación, por lo que conlleva a tener más programas activos para realizar lo mismo que se hacía con un cliente y un servidor.
- Sería necesario estudiar las posibilidades de alta disponibilidad del Broker MQTT porque al ser una topología en estrella, la comunicación depende del buen funcionamiento de un solo punto.

Teniendo en cuenta los resultados que se han obtenido de la validación, de la arquitectura, se puede afirmar que es un protocolo valido para futuras implementaciones sobre esta plataforma, ofreciendo multitud de posibilidades de desarrollo y escalado, esto último por el alto número de clientes que permite.

No obstante, lo anterior, este proyecto es un primer paso en la implementación del protocolo MQTT quedan algunos desarrollos que es necesario acabar para poder considerar completamente integrada las librerías de MQTT-SN dentro del software OpenWSN, y que son los siguientes:

- Variar el software del interfaz para que pueda ser utilizado por diferentes aplicaciones a la vez y con un funcionamiento asíncrono.
- Conseguir que el Gateway funcione de manera asíncrona hacia el lado UDP de la integración.
- Además, hacer que el Gateway pueda funcionar como un aglutinador de clientes UPD o de forma transparente.
- Contemplar todos los casos de uso y añadir timers de espera.
- Construir archivos Makefile para la compilación de la aplicación Suscriber y Gateway.

ANEXO A

A.1 Funciones que trabajan con MYSQL en el Suscriber:

```
1.  void mysql_prepare_query_insert(char* query,char *buf){
2.        uint32_t num;
3.        char fecha[50];
4.        num = (uint32_t)buf[1] << 8 | (uint32_t)buf[0];
5.        int number = (int)num;
6.
7.        printf("Data: %d\n" , number);
8.
9.        time_t t = time(NULL);
10.       struct tm tm = *localtime(&t);
11.
12.       sprintf(fecha,
13.               "%d-%d-%d %d:%d:%d",
14.               tm.tm_year + 1900,
15.               tm.tm_mon + 1,
16.               tm.tm_mday,
17.               tm.tm_hour,
18.               tm.tm_min,
19.               tm.tm_sec);
20.       sprintf(query,
21.               "INSERT INTO motes (number,fecha) VALUES (%d,'%s')",
22.               number,
23.               fecha);
24. }
25.
26. void mysql_app_init(void){
27.       conn = mysql_init(NULL);
28.       if (!mysql_real_connect(conn, server, user, password, database, 0, NULL, 0)){
29.             die((char *)mysql_error(conn));
30.       }
31. }
32.
33. void mysql_query_execute(char *query){
34.       if (mysql_query(conn, query)){
35.             die((char *)mysql_error(conn));
36.       }
37. }
```

A.2 Comandos de compilación de la aplicación Gateway y la aplicación Suscriber:

- Aplicación Suscriber:

gcc read_and_upload_db.c -o read -l paho-mqtt3c -L/usr/lib/x86_64-linux-gnu -lmysqlclient -lpthread -lm -lrt -latomic -ldl -I/usr/include/mysql

- Aplicación Gateway:

gcc read_and_forward.c MQTTSNConnectServer.c MQTTSNPacket.c MQTTSNDeserializePublish.c MQTTSNSerializePublish.c MQTTSNConnectClient.c -o read_mqttsn -l paho-mqtt3c -Wall -lm

ANEXO B:

B.1 Diagrama funcional openmqtt.

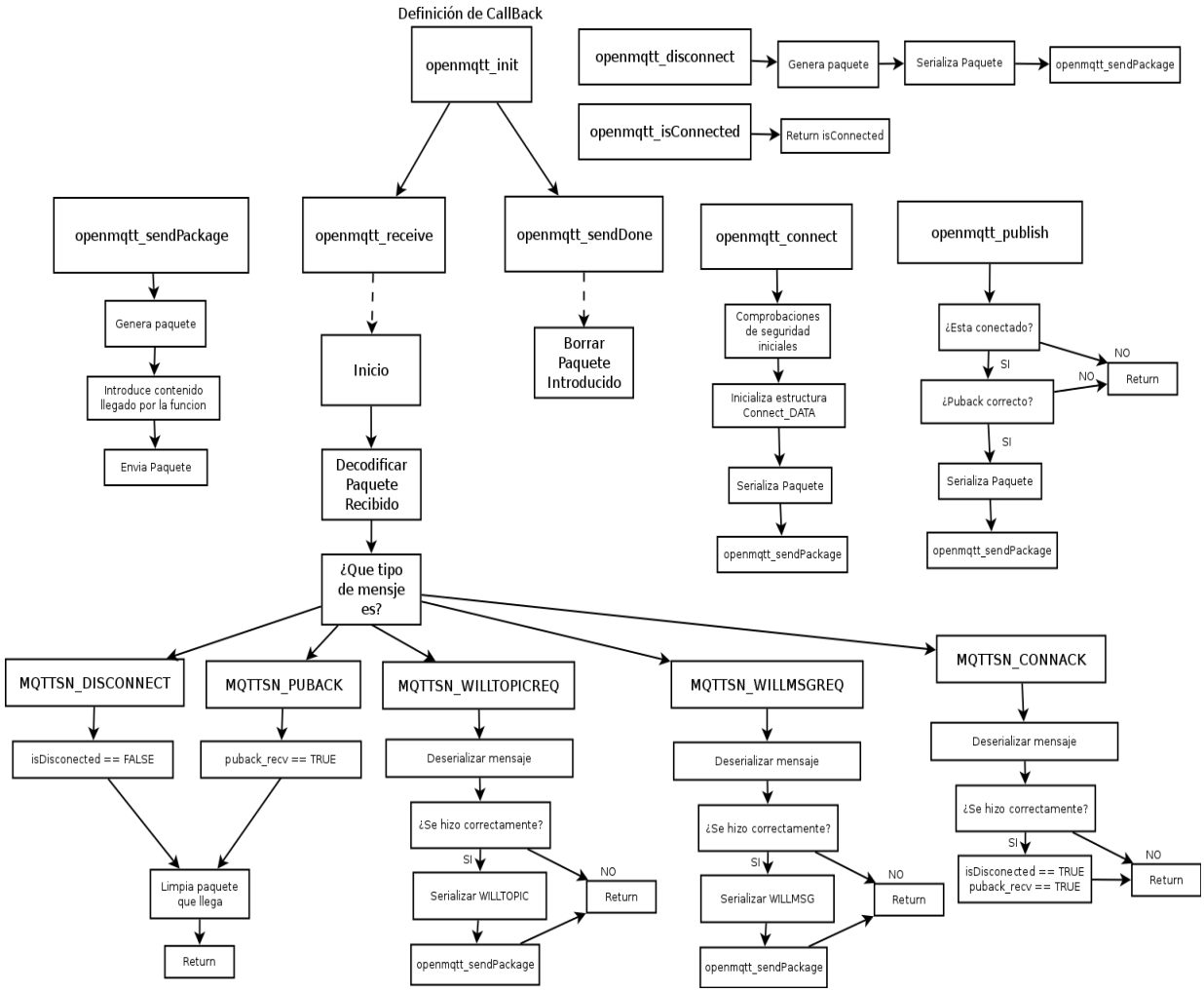


Ilustración 62: Esquema funcional openmqtt

Glosario

- MQTT-S: Message Queuing Telemetry Transport for sensors.
- IoT: Internet of things.
- IEEE 802.15.4e: Estándar que define el nivel físico y el control de acceso al medio de redes inalámbricas de área personal con tasas bajas de transmisión de datos.
- Raspberry Pi: Plataforma hardware de mini-computadora ARM compatible con Linux.
- OpenMOTE: Plataforma hardware de software embebido dedicada al IoT.
- Publish-Suscribe: Forma de trabajar que consiste en publicar información para que el subscriptor la lea.
- Footprint: Huella o cobertura.
- Broker: Corredor MQTT-S.
- OpenWSN: Sistema operativo que llevan los dispositivos OpenMOTE.
- Grafana: Herramienta web para crear paneles de control.
- Border router: Enrutador de paquetes de borde.
- Protoboard: Placa de contactos metálicos para poder hacer pruebas de circuitos electrónicos.
- MQTT-SN: Variante de MQTT que funciona con UDP
- M2M: Machine to Machine.
- PROTOCOLO RIP: Protocolo de enrutamiento.
- Scons: herramienta de compliacion de c en Linux.
- Printf: orden en código c para que escriba por pantalla algún resultado.
- Debuging: verificación de errores de código mediante trazas o herramientas para ello.
- Python: lenguaje de programación, en este caso se utiliza para realizar los test de funcionalidad.
- UDP: paquete de protocolo de transporte no orientado a conexión.
- Openvisualizer: herramienta del entorno de programación del sistema OpenWSN.
- Root: usuario o nombre que se indica en distribuciones Linux el usuario administrador del sistema o usuario principal.
- Query: petición de información a una base de datos o programa, en este caso a una base de datos MYSQL.
- MYSQL: es un tipo de base de datos relacional.
- SQL: lenguaje de programación de las queries.
- MAC: direccionamiento de red de nivel 2.
- Timestamp: tipo de variable mysql que indica una marca de tiempo.
- Dashboard: en inglés panel de control.
- Mysql-server: Servidor de bases de datos del tipo mysql.

Bibliografía

[1] Wikipedia Internet de las cosas: https://es.wikipedia.org/wiki/Internet_de_las_cosas
[2] OpenMote.com: http://www.openmote.com/
[3] OpenWSN: https://openwsn.atlassian.net/wiki/spaces/OW
[4] UOC: http://www.uoc.edu/portal/es/index.html
[5] MQTT-SN: http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf
[6] Mosquitto: https://mosquitto.org/
[7] MYSQL Wikipedia: https://es.wikipedia.org/wiki/MySQL
[8] MYSQL: https://www.mysql.com/
[9] Grafana: https://grafana.com/