UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Integración y control de dispositivos para el IoT con smartphones: Un caso de estudio

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Pedro David Rodríguez Sánchez

Tutores: Pietro Manzoni
Matteo Cesana

Curso 2018-1019

Resumen

El Internet de las Cosas o IoT (Internet of things) es una gran red que se esta desarrollando a nuestro alrededor. Gracias a las nuevas tecnologías, se están conectando todo tipo de objetos y dispositivos en una red mucho mayor que el Internet tal y como se conoce tradicionalmente. Desde una nevera hasta contenedores de basura en la calle, en la actualidad hay una inmensa variedad de dispositivos conectados.
Según las estimaciones de CISCO, una empresa mundial de fabricación de componentes de red, el número actual de dispositivos conectados a esta red es de 8'7 mil millones. Sin embargo, la estimación también prevé que el número de estos dispositivos llegará a 50 mil millones en 2020. Estos datos indican evidentemente que el campo del IoT está creciendo rápidamente y parece ser que seguirá así en los próximos años.

Existen ya varios productos disponibles en el mercado, los cuales pueden mejorar la calidad de vida de muchas personas. A pesar de ello, los precios resultan actualmente poco asequibles para la mayoría de la población. Sin embargo, existen herramientas para producir estos productos a mucho menor coste con una calidad similar, consiguiendo así un producto final más accesible para el usuario promedio.
Para demostrarlo, se ha elegido como caso de estudio un sistema de control y riego automático de plantas, siendo este un ejemplo recurrente en el sector IoT y un producto útil, ya que es muy común el cuidado de las plantas en los hogares. Además, el sector de la agricultura de precisión (AgroIoT) ha sido reconocido como un objetivo muy adecuado para usar este tipo de tecnologías.

Para realizar una implementación de un sistema de control y riego automático de plantas se ha desarrollado un análisis de los protocolos que utiliza esta red y de las tecnologías disponibles, en las cuales han destacado los dos sistemas operativos diseñados para Internet of Things: Contiki OS y TinyOS. También se ha estudiado las alternativas sin sistema operativo con plataformas de desarrollo, como un kit de desarrollo compatible con WiFi llamado NodeMCU.
El resultado del análisis ha descartado TinyOS como una posible solución al problema, siendo la compra de los dispositivos compatibles demasiado cara, además de que su compra no presenta la facilidad de disponer de tiendas online en la mayoría de los casos (es necesario el contacto directo con el distribuidor). Además, su trayectoria se ha dirigido más a sistemas empotrados para funciones industriales.
Por otro lado, Contiki OS ha destacado por su bajo consumo energético y sus bajos precios, pero su soporte ha sido discontinuado desde 2017. Sin embargo, un grupo de desarrolladores ha continuado el proyecto bajo el nombre de Contiki NG (Next Generation), pero su recorrido como sistema operativo está incompleto y su desarrollo todavía no presenta la estabilidad necesaria para el sistema.
Finalmente, el sistema ha podido ser implementado con el kit de desarrollo NodeMCU previamente mencionado. Este kit se ha integrado con sensores de humedad, temperatura e intensidad de luz. Junto con el desarrollo de una aplicación Android y un

servicio backend alojado en la nube, se ha conseguido la implementación del sistema de control y autorriego por un precio realmente asequible con la funcionalidad deseada.

Palabras clave

Internet of Things, Wifi, IEEE 802.15.4, NodeMCU, sensores, riego automático, Contiki, Cooja, TinyOS.

Abstract

The Internet of Things (IoT) is a large network that is developing around us. Thanks to new technologies, all kinds of objects and devices are being connected in a much larger network than the Internet as it is traditionally known. From a fridge to trash bins on the street, there is now an immense variety of connected devices.
According to estimates by CISCO, a global manufacturer of network components, the current number of devices connected to this network is 8.7 billion. However, the estimate also predicts that the number of these devices will reach 50 billion by 2020. These data clearly indicate that the IoT field is growing quickly and it seems that it will continue to do so in the years to come.

There are already several products available on the market, which can improve the quality of life of many people. Despite this, prices are currently unaffordable for the most of the population. However, there are tools to produce these products at much lower cost with similar quality, thus making the final product more accessible to the average user.
To demonstrate this, an automatic plant control and irrigation system has been chosen as a case study, this being a recurrent example in the IoT sector and a useful product, as it is very common to care for plants in homes. In addition, the precision agriculture sector (AgroIoT) has been recognised as a very suitable target for using this type of technology.

In order to implement an automatic plant control and irrigation system, an analysis of the protocols used by this network and the available technologies has been developed, in which the two operating systems designed for the Internet of Things have been highlighted: Contiki OS and TinyOS. Alternatives without an operating system with development platforms have also been studied, such as a WiFi-compatible development kit called NodeMCU.
The result of the analysis has ruled out TinyOS as a possible solution to the problem, being the purchase of compatible devices too expensive, in addition to the fact that their purchase does not present the ease of having online stores in most cases (direct contact with the distributor is necessary). In addition, its trajectory has been directed towards embedded systems for industrial functions.
On the other hand, Contiki OS has stood out for its low energy consumption and low prices, but its support has been discontinued since 2017. However, a group of developers has continued the project under the name of Contiki NG (Next Generation), but its journey as an operating system is incomplete and its development still doesn’t have the steadiness the system would need.
Finally, the system could be implemented with the previously mentioned NodeMCU development kit. This kit has been integrated with humidity, temperature and light intensity sensors. Together with the development of an Android application and a backend service hosted in the cloud, the implementation of the control and self-irrigation system has been achieved at a really affordable price with the desired functionality.

Keywords

Internet of Things, Wifi, IEEE 802.15.4, NodeMCU, sensors, automatic irrigation, Contiki, Cooja, TinyOS.

Resum

Internet de les coses o IoT (Internet of things en anglés) és una gran xarxa que s'està desenvolupant al nostre voltant. Gràcies a les noves tecnologies, tota clase d'objectes i dispositius es connecten a una xarxa molt més gran que Internet tal com el coneguem tradicionalment. Des d'una nevera fins als contenidors de fem del carrer, actualment hi ha una immensa varietat de dispositius connectats.
Segons les estimacions de CISCO, una empresa mundial de fabricació de components en xarxa, el nombre actual de dispositius connectats al IoT és de 8,7 mil milions. A més a més, les estimacions també diuen que el nombre de dispositius arribarà a 50 mil milions en 2020. Aquestes dades indiquen evidentment que el camp del IoT està creixent ràpidament i sembla que continuarà igual en els pròxims anys.

Actualment ja existeix una gran varietat de productes en el mercat, els quals poden millorar la qualitat de vida de moltes persones. Tot i això, els preus resulten massa cars per a la majoria de població. Amb tot, existeixen ferramentes per a produir aquests productes a menor preu amb una qualitat similar, aconseguint un producte final més accessible per a l'usuari mitjà. Per demostrar-ho, s'ha escollit com a cas d'estudi un sistema de control i reg automàtic de plantes, ja que és un exemple comú en el sector IoT i un producte útil, ja que moltes persones cuiden plantes en la seua llar. A més a més, el sector de l'agricultura de precisió (AgroIoT) ha sigut reconegut com un objectiu molt adequat per a utilitzar aquest tipus de tecnologies.

Per a realitzar una implementació d'un sistema de control i reg automàtic de plantes s'ha desenvolupat una anàlisi dels protocols que utilitza aquesta xarxa i les tecnologies disponibles, on han destacat dos sistemes operatius desenvolupats per al IoT: ContikiOS i TinyOS. També s'ha estudiat les alternatives sense sistemes operatius amb plataformes de programació, com el kit de programació compatible amb Wifi anomenat NodeMCU. El resultat de l'anàlisi ha descartat TinyOS com una possible solució al problema, ja que la compra dels dispositius compatibles era massa cara, cal dir que aquesta compra no presentava les facilitats de disposar d'una tenda en línia en la majoria dels casos (és necessari contactar amb el distribuïdor). A més, la seua trajectòria s'ha enfocat més als sistemes encastats per a funcions industrials. En comparació, Contiki OS ha destacat pel seu baix consum energètic i els seus preus econòmics, però el suport al sistema va ser discontinuat des de 2017. No obstant això, un grup de programadors ha continuat el projecte sota el nom de ContikiNG (Next Generation), però el recorregut com sistema operatiu està incomplet i la seua construcció encara no presenta l'estabilitat necessària per al sistema. Finalment, el sistema ha pogut ser implementat amb el kit de programació NodeMCU prèviament mencionat. Aquest kit s'ha integrat amb sensors d'humitat, temperatura i intensitat de llum. Junt amb la creació d'una aplicació Android i un servei backend en el núvol, s'ha aconseguit la implementació del sistema de control i reg automàtic per un preu adequat per a totes les persones, amb la funcionalitat desitjada.

Paraules clau

Internet of Things, Wifi, IEEE 802.15.4, NodeMCU, sensors, reg automàtic, Contiki, Cooja, TinyOS.

Índice

Capítulo 1: Introducción

El campo del Internet of Things ha estado creciendo vertiginosamente estos últimos años. Esta nueva gran red que ha surgido en los últimos años presenta una amplia variedad en cuanto a dispositivos, interacción con los usuarios, sistemas operativos utilizados, infraestructura de redes, etc.

En este trabajo se va a realizar un caso de estudio que abordará la creación de un sistema con las características propias de un sistema de IoT, con varios objetivos como la búsqueda de un diseño de sistema útil al precio más económico posible. También, dado a que una gran parte de estos sistemas se controla mediante aplicaciones web o móvil, se realizará una aplicación para controlar el sistema diseñado.
El sistema elegido es un sistema de control y autorriego de plantas en macetas, como las que muchas personas tienen en su propia casa. La elección ha sido motivada por el hecho de que es un tema muy recurrente en el sector IoT y puede llegar a ser realmente útil en una vivienda. Además, la implementación es sencilla y pueden verse los resultados de forma fácil.

Como se ha mencionado antes, hablar sobre IoT no es una cuestión breve. Esta inmensa red tiene una gran umbral de precios en la compra del hardware y maneras muy diferentes de implementación. Dada esta situación, es necesario un análisis para valorar cuál es la mejor opción a la hora de construir el sistema propuesto.

Motivación

Los sistemas que se pueden desarrollar en IoT pueden mejorar la calidad de vida de las personas. Pero estos sistemas no son siempre asequibles para la gran mayoría de la población, ya que su precio en los productos finales suele tener precios muy altos. Con las nuevas tecnologías y los nuevos medios de compra en tiendas online como Amazon [1] y Aliexpress [2] se puede conseguir un presupuesto mucho más adecuado para el consumidor medio. Además, existe una gran comunidad de licencia abierta que apoya y facilita el desarrollo de estos sistemas.
Respecto al caso de estudio elegido, ha sido motivado por una necesidad común a muchas personas que mantienen sus plantas en su hogar. Este sistema, además de aumentar la comodidad del cuidado de las plantas, aborda el problema de el cuidado de las plantas cuando se está lejos de casa. Con un sistema que se ocupe del riego, las personas podrán conservar las plantas sanas durante estancias fuera del hogar. Además, gracias a la posibilidad de regular el riego y la humedad de la planta, es más fácil mantener plantas con cuidados más específicos.
Personalmente, el problema del cuidado de las plantas al estar de vacaciones es la principal motivación. Además, la facilidad de delegar el cuidado de la planta a una aplicación automatizada permite cultivar plantas útiles para el consumo en casa, como

la hierbabuena. Un sistema de cuidado y riego automatizado motiva a aumentar las plantas en la casa.

Objetivos

Implementar un sistema de riego y control de plantas automatizado:
- El coste de producción debe ser el mínimo posible.
- El sistema debe ser adaptable a los diferentes tamaños que presente el recipiente de la planta.
- La fuente de energía del sistema debe ser adaptable, compatible con baterías recargables, fuentes de energía renovable y la corriente.
- El dispositivo debe ser independiente de la elección del suministro de agua.
- El control del sistema debe ser realizado mediante una aplicación Android.
- El riego de la planta debe funcionar aunque el usuario no se conecte a la aplicación Android.
- El modo de riego se podrá configurar, ofreciendo diferentes modos.

Encontrar y valorar la tecnología que más se adapte al proyecto dentro del abanico de tecnologías existentes en el campo del IoT:
- Analizar los sistemas operativos existentes diseñados para el IoT.
- Analizar el funcionamiento de los sistemas.
- Valorar la comunidad y la documentación disponible en cada una de las opciones.

Estructura

Este trabajo de fin de grado está estructurado de la siguiente forma:
Primero, se analizarán los sistemas de autorriego y control de plantas existentes en el mercado y se expondrán sus principales deficiencias. Se presentará como alternativa un sistema con unas ventajas respecto a los sistemas actuales del mercado.
Después de este análisis, se abordarán los temas necesarios para la implementación de este sistema, empezando por los protocolos que se utilizan en cada capa de la red, seguido de la comparativa entre los sistemas operativos ContikiOS y TinyOS como posible solución para el sistema. En cuanto a estos, los puntos a abordar serán su funcionamiento, las herramientas que ofrecen, la comunidad y documentación disponible y la facilidad de aprendizaje. También se evaluará su compatibilidad con el proyecto. Tras acabar de analizar los dos sistemas operativos se presentará otra opción disponible para el desarrollo de estos sistemas: el uso de plataformas de desarrollo y sensores compatibles.
Al acabar la exposición del contexto del problema, se introducirá el diseño de la solución propuesta, demostrando su compatibilidad con diferentes redes IoT. Este diseño contendrá tres partes: aplicación Android, servicio backend y red IoT.

La siguiente parte del proyecto será la implementación, dividida en dos partes. La primera parte explicará cómo se ha implementado el sistema en un entorno real con el kit de desarrollo NodeMCU. La segunda parte mostrará como se podría implementar el sistema de forma totalmente compatible con el sistema operativo Contiki OS en el simulador que aporta.
Después, se llevará a cabo la implantación del sistema y las pruebas de funcionamiento de forma empírica.
Para finalizar, se realizará una conclusión con los principales conceptos extraídos de todo el proyecto y se explicarán los trabajos futuros del proyecto.

Convenciones

Se indicarán con cursiva nombres de librerías, texto completo de los acrónimos y palabras técnicas en inglés.
Las citas o los apodos de personas relevantes para el proyecto serán marcados entre comillas dobles.
El código fuente se encontrará encuadrado con un fondo gris.
En las enumeraciones de características se usará negrita para clarificar la separación.

Capítulo 2: Estado del arte

El caso del autorriego de plantas es muy común en este campo, así que existen diversas soluciones ya implementadas por diferentes empresas. La solución más parecida al proyecto es *Parrot Pot*, una maceta inteligente diseñada por *Parrot*. [3]

Parrot Pot tiene sensores para la humedad, la intensidad de la luz, la temperatura y la cantidad de fertilizante. Funciona con pilas y tiene un tanque de agua de 2,2 litros.
El control de la maceta se realiza mediante una aplicación móvil conectada por bluetooth.

Fuente: https://www.parrot.com/es/jardin-conectado/parrot-pot

También existe otro producto realmente similar, *Fliwer* [4].

Fliwer usa 3G o WiFi para comunicarse, pero necesita la instalación de un router para realizar la comunicación.
El control se realiza a través de una aplicación web y no hay ninguna clase de aplicación nativa para móvil.
Este producto es el que presenta una mayor cantidad de sensores en comparación a otros productos del sector: intensidad de luz, temperatura, humedad, humedad del suelo, conductividad eléctrica y un medidor de caudal para saber el consumo del agua de la planta. La alimentación se realiza mediante baterías recargables.

Fuente: http://www.fliwer.com/

Estos dos productos son los más parecidos al proyecto. En el mercado también existen otras soluciones también parecidas, y muchas que solo controlan el riego o bien solo controlan la temperatura, humedad y otros parámetros de la planta.

Crítica al estado del arte

El mayor problema con las soluciones disponibles actualmente son su elevado precio. Existen productos asequibles como el *Xiaomi Smart Plant Monitor*, que solo cuesta 20€ y monitoriza el estado de la planta, pero no tiene nada que ver con su riego. [5]

También existen algunos problemas con los productos más parecidos mencionados previamente.
Sobre el *Parrot Pot*, el precio ronda los 100€, lo cual no es excesivamente caro, pero no es asequible para la mayoría de las personas, ya que el producto es simplemente una maceta pequeña de unos 20 cm de diámetro y no es un precio comparable al de una maceta de estas proporciones. Además, el producto solo provee una maceta estándar, no se puede usar este sistema con macetas diferentes y tiene un tamaño relativamente pequeño, el cual tampoco se puede ampliar. Adicionalmente, solo presenta la opción de usar su depósito para regar la planta y de usar pilas para proporcionar la energía necesaria, limitando así otras opciones como conectar la maceta a la corriente de la casa o usar algún tipo de energía renovable.

Por otro lado, *Fliwer* es una opción bastante más flexible. Controla el área donde se coloca y puedes elegir diferentes tipos de riego. Además, este producto tiene mas sensores que cualquier otro producto del mercado. Pero el gran problema de *Fliwer* es su precio. Él *Fliwer* más económico cuesta más de 1000 €. Además, necesita la instalación de una infraestructura adicional (router). Esta opción esta realmente alejada de ser asequible para un usuario promedio.

Propuesta

La propuesta de este proyecto es el desarrollo de un sistema de autorriego y control de plantas a mucho menor coste, demostrando así que se puede realizar el desarrollo de esta clase de sistemas para una venta asequible al consumidor.
El sistema a su vez, será adaptable a varios tipos de redes IoT, con diferentes fuentes de alimentación, diferentes modos de conexión y diferentes fuentes de agua para el riego.

Gracias a los nuevos dispositivos, apoyados por una comunidad open source creciente han conseguido que los precios y el coste de desarrollo baje (y continúe bajando). El desarrollo de este sistema permitirá la creación de un producto asequible para toda la población, adaptable a las macetas ya existentes en los hogares.

Capítulo 3: Análisis del problema

El desarrollo del sistema de control y riego de plantas propuesto requiere ciertos requisitos para su realización.

El sistema a realizar debe ser manejado con una aplicación Android, lo que permite que la interacción se pueda realizar mediante protocolos como Bluetooth o WiFi, pero también requiere que funcione cuando la aplicación no se ejecuta o no está en el rango, lo cual crea la necesidad de un intermediario con el cual se conecte el sistema o de una memoria interna para guardar todos los datos hasta una sincronización. Ya que la sincronización se debe realizar de una forma cercana al dispositivo que controle la planta, se desestima esta opción, ya que uno de los objetivos requiere que se pueda consultar el estado de la planta desde cualquier lugar (incluso si la persona se encuentra de vacaciones lejos de su casa). La solución para este problema es la inclusión de un servicio backend en la nube accesible mediante la aplicación en cualquier momento desde cualquier lugar. La comunicación del sistema encargado del riego de las plantas debe comunicarse con este servicio backend mediante un protocolo, el cual debe ofrecer un bajo consumo de energía y estar disponible en la mayoría de los hogares.
Además, para conseguir el mínimo de consumo energético, el numero de transmisiones de datos debe ser el menor posible. Existen diferentes protocolos diseñados para IoT que cumplen estos objetivos, así que se realizará un análisis de los protocolos más interesantes en toda la pila de red, exponiendo las ventajas, desventajas y su conveniencia para el proyecto.

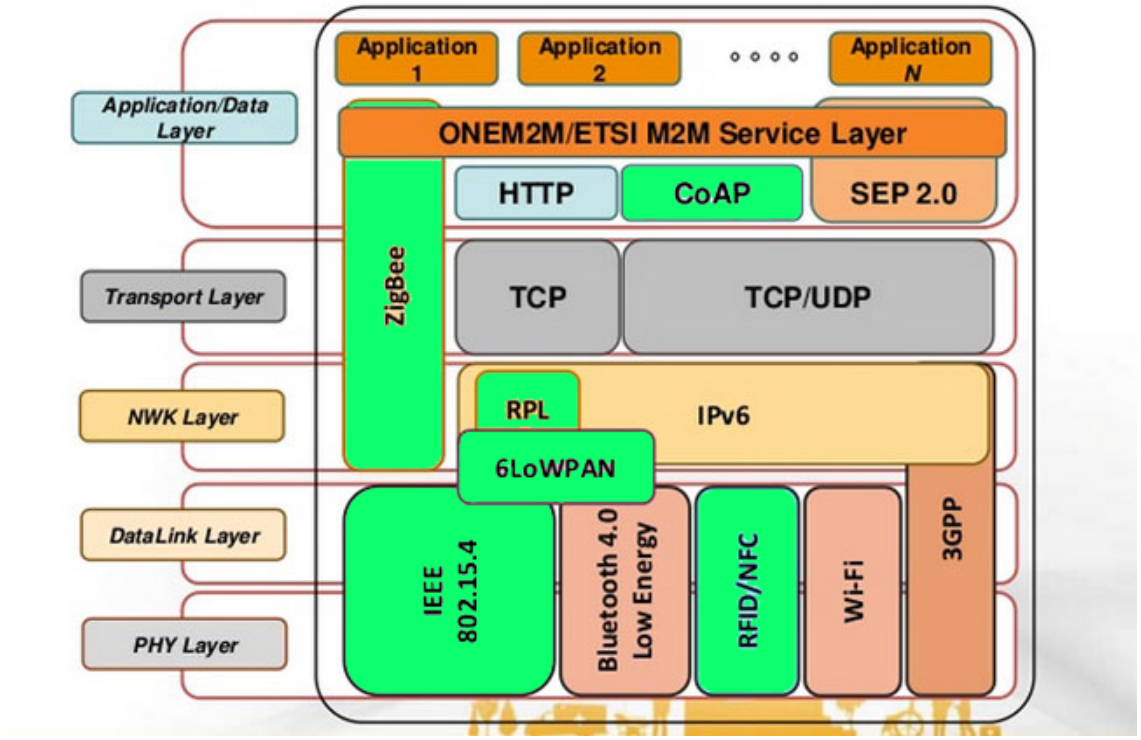
El mayor objetivo del proyecto es conseguir el mínimo precio posible, intentando mantener la mayor calidad y ahorro de energía. El sistema de control y riego requiere de un dispositivo que pueda adquirir muestras de temperatura, humedad y luz solar para controlar el estado de la planta y seleccionar el momento más adecuado para el riego. Así también, requiere de una salida digital para controlar la apertura o cierre de una válvula que permita el paso del agua desde la fuente hasta la planta. Como se ha comentado antes, el sistema también necesita una antena de comunicación para enviar los datos recolectados. Para satisfacer estos requisitos existen diversas soluciones que se deben analizar. Existen sistemas operativos desarrollados para el Internet of Things, los cuales ofrecen una implementación de los protocolos necesarios para conseguir una comunicación con poco consumo energético. Las plataformas compatibles poseen sensores integrados y los ofrecen librerías para su control. Los dos sistemas operativos más relevantes en este campo son ContikiOS y TinyOS, que también serán analizados en este capitulo.

Finalmente se analizará otra tipo de solución posible en este sistema, la cual destaca por su facilidad de implementación, su bajo precio y la gran variedad disponible. Se trata de las plataformas de desarrollo de licencia libre disponibles en el mercado, como la plataforma *Arduino*. Estas plataformas no suelen traer todos los sensores

necesarios integrados en la placa, con lo cual también se analizarán los sensores compatibles más adecuados para el proyecto.

Protocolos IoT

En el IoT, el principal objetivo de los dispositivos es el ahorro de energía.
Hay una gran cantidad de protocolos en el Internet clásico, como los que usamos diariamente (HTTP, TCP, IP, etc). Pero estos protocolos no son la mejor opción para el consumo energético y capacidad de procesamiento de los limitados dispositivos que se utilizan en las redes IoT. Así pues, entidades como IETF (*The Internet Engineering Task Force*) trabajaron en nuevos protocolos para solucionar este problema.
Actualmente, hay muchos protocolos adecuados para estos dispositivos con recursos limitados. Esta imagen define bien la capa de red que ocupan algunos de estos:



Protocolos más populares actualmente. En verde, los protocolos diseñados para el IoT.
Fuente: https://aprendiendoarduino.wordpress.com/2016/07/05/arduino-y-iot-2/

Hoy en día, la estructura reina de la pila de capas de Internet es el *Internet Protocol suite*, TCP/IP. Esta es la implementación del modelo *Open System Interconnection* (OSI), un estándar abstracto que describe la pila de Internet en siete capas.

La red IoT creó la necesidad de una renovación de protocolos de este modelo, para compatibilizarla con los requisitos de los dispositivos empleados en esta red. Estos son los protocolos más conocidos del IoT ordenados por capa de red:

Modelo TCP/IP	Protocolo IoT
Aplicación	HTTPS XMPP CoaP MQQT AMQP
Transporte	UDP TCP
Internet	IPv6 6LowPAN RPL
Enlace y físico	IEEE 802.15.4 WiFi (802.11) Ethernet GSM CDMA

Capa física y de enlace

En este nivel se van a exponer las diferencias entre Wi-Fi y IEEE 802.15.4.

El estándar Wi-Fi es conocido por todos. Esta es una de sus fortalezas, ya que se ha convertido en una red accesible por la mayor parte de la población mundial. Lo más común es que cada persona tenga su red Wi-Fi en su propia casa.
A pesar de su omnipresencia, este protocolo presenta un problema para el IoT. Este protocolo no ha sido diseñado para la eficiencia energética, sino que se orienta a la velocidad, mediante un gran ancho de banda y un corto rango de alcance. Este gran ancho permite una gran velocidad, pero con un consumo de energía mayor. Este protocolo también es incompatible con redes de largo alcance. Wi-Fi no está diseñado para el ahorro de energía, pero su ubicuidad permite su uso en cualquier hogar como sistema de comunicación con la red.

Por otro lado, el estándar IEEE 802.15.4 ha sido diseñado para conseguir eficiencia energética, baja tasa de transmisión de datos y una banda de ancha mucho más pequeña. También ofrece un corto alcance, ya que está diseñado para redes pequeñas.
En IEEE 802.15.4 hay dos tipos de nodos:

- *Full Function Devices* (FFD)
  Estos dispositivos pueden encargarse de coordinar la red.

- *Reduced Function Devices* (RFD)
  Estos dispositivos son extremadamente simples, limitados en recursos y solo pueden comunicarse con otros FFD. Nunca pueden coordinar la red, ya que sus recursos de comunicación y cálculo son muy limitados.

Las redes que funcionan con el protocolo IEEE 802.15.4 necesitan coordinar dispositivos FFD y RFD, con lo cual su instalación para el proyecto requeriría de la instalación adicional de un dispositivo que realice el papel de coordinador. Esto es una desventaja en cuanto a la instalación de la red, aunque termina realizando un consumo energético menor.
Como las redes WiFi se encuentran en prácticamente casi todo los hogares, no necesitaría una infraestructura adicional, pero su consumo energético sería mayor.
En este punto, la solución se decidirá dependiendo de las características del dispositivo elegido.

Existen otros estándares que sí contemplan un gran alcance, como el protocolo de comunicación *LoRa*. En este proyecto no se contemplarán este tipo de estándares, ya que para la solución propuesta una solución con un corto alcance es adecuada.

Capa de red

En esta capa se van a exponer las características de IP(IPv4 y IPv6), RPL y 6LowPAN.

El *Internet Protocol (IP)* se ha convertido en un requisito para estar conectado a Internet, es el sistema de direcciones de la red. La expansión de la red IoT y los nuevos dispositivos en la red de Internet también necesitan direcciones IP, y IPv4 (versión 4 de este protocolo) tiene un tamaño realmente limitado. Esa es la razón de por qué IPv6 es tan importante para el IoT. IPv6 dispondrá de unas 4,3 mil millones de direcciones, lo que es suficiente para las necesidades de comunicación en el futuro.

IPv6 es realmente importante para el Internet of Things, pero hay muchas diferencias entre el protocolo IPv6 y los requisitos del estándar 802.15.4. Por ejemplo, la unidad máxima de transmisión (MTU) en IPv6 requiere al menos 1280 bytes, mientras que el tamaño estándar de un paquete en IEEE 802.15.4 es de 127 bytes.
6LowPAN es la solución a estas diferencias. Se trata de una adaptación del IPv6 para dispositivos de recursos limitados. Este protocolo fue desarrollado por él antes mencionado IETF, con el objetivo de implantar el protocolo IP en las redes de dispositivos con recursos limitados característicos de las redes IoT.

Por otra parte se encuentra el protocolo RPL (*IPv6 Routing Protocol for Low-Power and Lossy Networks*). Se trata de un protocolo de enrutado para las redes que presenta el IoT, redes que buscan el mínimo consumo de energía y que sufren muchas pérdidas de paquetes. La comunicación en estas redes está caracterizada por un entorno que provoca muchas perdidas de paquetes, con lo cual era necesario un protocolo que soportara estas peculiaridades.

RPL ofrece un mecanismo de enrutado que soporta trafico multipunto a punto (dispositivos a dispositivo de control), punto a multipunto (punto de control a dispositivos) y punto a punto para las comunicaciones entre los propios dispositivos. Con este protocolo se soportan las topologías que propone el estándar IEEE 802.15.4. El funcionamiento de RPL se basa en el cálculo de la ruta optima mediante la construcción de un grafo de nodos, calculado según métricas dinámicas para minimizar el consumo de energía y la latencia.

Mediante 6LowPAN y RPL, se ha establecido un protocolo de red adecuado para los dispositivos con recursos limitados propios de redes IoT. Estos protocolos son los que utilizan los sistemas operativos ContikiOS y TinyOS. Los dispositivos que utilizan WiFi no usan estos protocolos.

Capa de transporte

En esta capa, se mostrarán las diferencias entre los dos protocolos más usados, TCP y UDP.

- Protocolo de control de transmisión (TCP)
  Este protocolo es el más usado por la mayoría de interacciones de humanos con la web (correo electrónico, búsquedas en la web).
  TCP ofrece la noción de una conexión lógica, la aceptación de paquetes transmitidos, retransmisión de paquetes perdidos y control de flujo. Todas estas características son muy buenas, pero son una gran sobrecarga para los sistemas con pocos recursos utilizados en las redes IoT. Esta es la razón de por qué se prefiere el protocolo UDP para estos dispositivos.

- Protocolo de datagramas de usuario (UDP)
  Este protocolo permite el envío de datagramas a través de la red sin que se haya establecido previamente una conexión, ya que el propio datagrama incorpora suficiente información de direccionamiento en su cabecera. Tampoco tiene confirmación ni control de flujo, por lo que los paquetes pueden adelantarse unos a otros; y tampoco se sabe si ha llegado correctamente, ya que no hay confirmación de entrega o recepción. Esta falta de formalidad característica de UDP orienta este protocolo a aplicaciones que pueden aceptar algunos errores, perdidas o duplicaciones. La gran ventaja de UDP es que es mucho más ligero, siendo así la mejor opción para las redes IoT.

Esta capa ofrece la base para los protocolos de aplicación que se considerarán en el siguiente apartado.

Capa de aplicación

Hay una gran cantidad de protocolos de aplicación existentes, muchos desarrollados específicamente para el IoT. En esta capa se realizara el análisis de los dos que se utilizarán en el proyecto, HTTP y CoaP.

- HTTP (*Hypertext Transfer Protocol*)
  El protocolo de transferencia de hipertexto es el protocolo de aplicación utilizado en los sistemas web. Este funciona sobre el protooclo TCP/IP.

Es un protocolo muy versátil, utilizado por todo el World Wide Web. El problema que presenta es la sobrecarga de información que presenta, además de que al sustentarse en TCP/IP, los paquetes de datos son demasiado grandes para los sistemas con pocos recursos comunes en las redes IoT.

Este protocolo será necesario para la comunicación con el backend, con lo cual será utilizado o bien los dispositivos WiFi de forma directa o en los dispositivos IEEE 802.15.4 a través del dispositivo FFD que se encargue de coordinar la red.

- CoaP (*Constrained Application Protocol*)
  El protocolo de aplicación restringida es un protocolo de la capa de aplicación con petición y respuesta síncrona. Este esta diseñado también por la IETF para los dispositivos con recursos limitados. Su diseño se basa en HTTP, ya que introduce un subconjunto de los métodos disponibles en HTTP como GET, POST, PUT y DELETE.
  Esta característica le da interoperabilidad con el protocolo HTTP, pero necesita la intervención de un dispositivo que realice la conversión de un protocolo a otro.
  A diferencia de HTTP, CoaP funciona sobre UDP, para así mantener la implementación del protocolo mucho más ligera.
  Este protocolo también ofrece diferentes recursos como el modo de observación para ahorrar peticiones.

Contiki OS

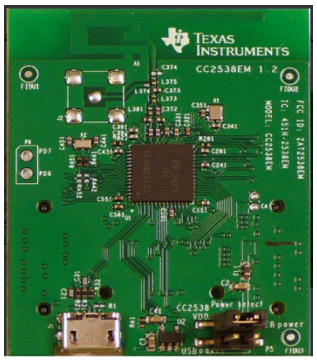
Contiki es un sistema operativo de licencia abierta diseñado específicamente para el Internet of Things. Este sistema conecta dispositivos consiguiendo un bajo coste energético, soporta IPv4 y IPv6, además de los estándares más populares del IoT como 6LowPAN, RPL, CoaP, etc.
Contiki está escrito en C y sus aplicaciones se escriben también en el mismo lenguaje. Además, este sistema provee un simulador muy completo llamado Cooja, el cual puede emular el comportamiento de algunos dispositivos hardware compatibles con este sistema operativo.

Este sistema operativo ocupará la mayor parte del análisis del proyecto, ya que se ha considerado muy interesante y prometedor para el Internet of Things.

Compatibilidad de dispositivos

Contiki ofrece un gran numero de dispositivos compatibles, pero ha sido diseñado sobre todo para 3 plataformas en concreto:

CC2538

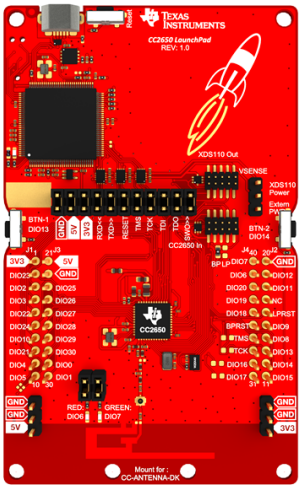


Es una plataforma *System-on-Chip* (SoC) para aplicaciones de gran rendimiento con el protocolo IEEE 802.15.4.
Este dispositivo combina el potente microcontrolador ARM Cortex-M3-based, dotado de 32KB de RAM en el propio chip y hasta 512KB de memoria flash en el chip con una fuerte radio IEEE 802.15.4.
El precio de este chip es realmente asequible:
Menos de 7$ [6].

SensorTag

Basado en el microcontrolador inalámbrico con ultra-ahorro de energía CC2650, soporta el desarrollo para el protocolo IEEE 802.15.4, Bluetooth LE y 6LoWPAN.
Posee sensores de temperatura, movimiento, humedad y presión barométrica.
El precio es un poco mayor, pero continua siendo asequible para el proyecto: Unos 15$ [7].

CC2650



Es una plataforma inalámbrica diseñada para BluetoothLE, IEEE 802.15.4 y 6LoWPAN. Su microcontrolador pertenece a la familia de dispositivos CC26xx de bajo coste y ultra-ahorro de energía.
El modo de ahorro de energía del microcontrolador provee un excelente tiempo de vida de las baterías y permite el uso de energías renovables para la recarga.
El precio es muy bajo, lo que convierte esta opción en la que sería la más adecuada para hacer el proyecto con este sistema operativo: Menos de 5$ [8].

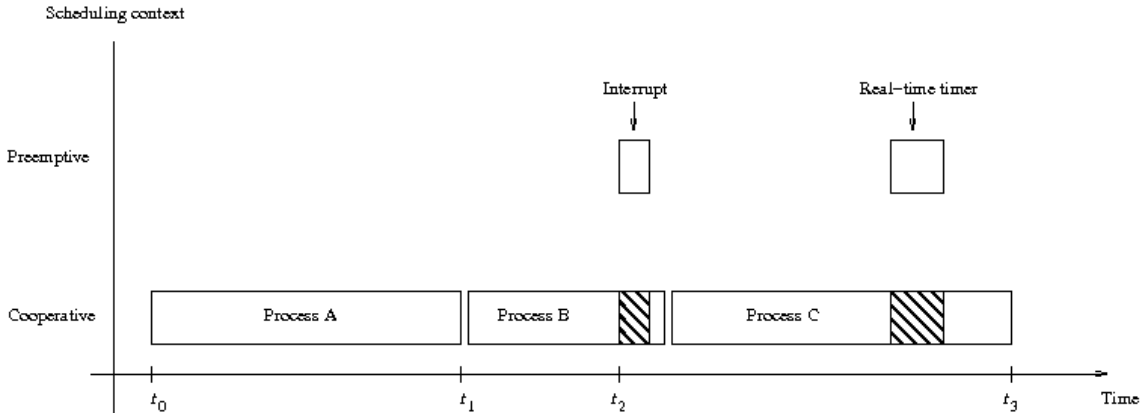
Hay más plataformas compatibles como la Re-Mote de Zolertia, algunos chips de Atmel como el Atmel RF230 y incluso es posible incluir Contiki en otras plataformas como Arduino [9].
La lista completa del hardware que soporta Contiki OS se encuentra en la página oficial de Contiki, indicada en la bibliografía.

El funcionamiento de Contiki

Existe una gran cantidad de documentación sobre el funcionamiento interno de Contiki, así que se analizarán los puntos más importantes:

- **Procesos**

  Todos los programas de Contiki son procesos. Los procesos son piezas de código. El código en Contiki puede ejecutarse en dos contextos de ejecución diferentes: cooperativo (*cooperative*) y prioritario (*preemptive*).
  El contexto cooperativo es el visible a los programadores, es donde el programa escrito será ejecutado. Este contexto ejecuta el código secuencialmente y debe completar una pieza de código antes de que la siguiente empiece a ejecutarse.
  Por otro lado, el contexto prioritario es el contexto para las interrupciones. El contexto cooperativo se detiene cuando hay código prioritario que ejecutar, dando paso a este ultimo. Los eventos que suceden fuera del programa, como interacción con el exterior o tareas programadas con un temporizador se ejecutaran en el contexto prioritario.
  La siguiente ilustración representa como funcionan los cambios de contexto:
  Los procesos tienen un bloque de control y un hilo de proceso.



Fuente: https://github.com/contiki-os/contiki/wiki/Processes

El bloque de control tiene información sobre el estado del proceso, su nombre y el puntero en memoria a él mismo. Esta información es usada únicamente por el kernel, y el proceso en sí no puede tener acceso a ella.
Los bloques de control están implementados de la siguiente manera:

```
struct process {
   struct process *next;
   const char *name;
   int (* thread)(struct pt *, process_event_t, process_data_t);
   struct pt pt;
   unsigned char state, needspoll;
 };
```

El campo *pt* almacena el estado del protohilo en el hilo del proceso. Los protohilos son una clase de hilos extremadamente ligeros diseñados para dispositivos memoria restringida, para los que Contiki fue diseñado. Pueden ser usados con o sin un sistema operativo que provea manejadores de eventos bloqueantes. Además, tienen un sobrecoste de memoria muy ligero, unos dos bytes por protothread. Fueron inventados por el diseñador de Contiki, Adam Dunkels.
Los campos *state* y *needspoll* son indicadores internos que almacenan el estado del proceso. Para crear un proceso, Contiki define la macro *PROCESS*. Esta macro necesita el nombre de la variable del bloque de control y un nombre para el proceso:

```
PROCESS(hello_world_process, "Hello world process");
```

Para ejecutar el proceso, se define la macro *AUTOSTART_PROCESSES*, que permite lanzar uno o más procesos:

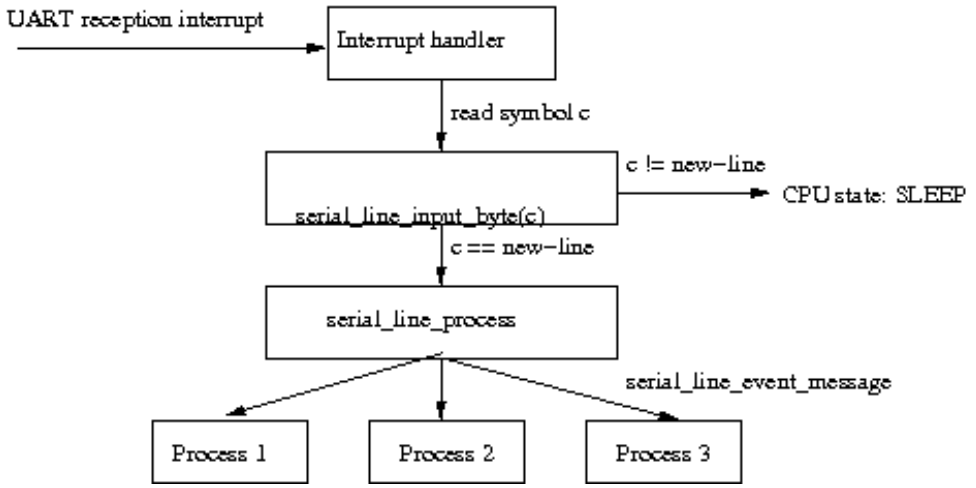
```
AUTOSTART_PROCESSES(&process1, &process2);
```

Finalmente, para definir el código de un proceso, Contiki provee la siguiente estructura:
El código debe estar envuelto entre las dos macros de control de inicio y fin: *PROCESS_BEGIN()* y *PROCESS_END()*.

```
PROCESS_THREAD(hello_world_process, ev, data){
   PROCESS_BEGIN();
   // Dentro de estas dos macros se escribe el programa
   PROCESS_END();
}
```

- **Eventos**
  La programación de los programas en Contiki está dirigida por eventos. Este paradigma de programación hace que la estructura y la ejecución del código este controlada por los eventos del sistema, definidos por el usuario o provocados por el propio programa.
  Este paradigma es considerado muy difícil para el desarrollo y mantenimiento. Pero tal y como está estructurado en Contiki, mediante los protohilos, facilita escribir el código de forma dirigida por hilos.

- **Entrada/Salida**
  El tipo de dispositivos I/O disponibles depende de gran manera de la plataforma utilizada.
  Contiki tiene dos interfaces genéricas de entrada y salida: Serial y LEDs. Estas interfaces son soportadas por la gran mayoría de plataformas compatibles de Contiki.
  La comunicación por puerto serial usa las funciones estándar de C, *printf()* para la salida y tiene funciones específicas de Contiki para la entrada de datos.
  El funcionamiento es el siguiente: Una interrupción es generada cuando un carácter esta listo para ser leído por el puerto serial. El manejador de entrada ejecuta la función *serial_line_input_byte()*, almacenando los caracteres en un buffer hasta recibir el símbolo de salto de linea. Cuando una linea es completada, Contiki realiza una difusión a todos los procesos en ejecución.
  Un error que se tuvo en un proceso de desarrollo fue el no enviar el carácter de salto de línea, quedando así sin recibir ningún tipo de respuesta por el puerto serial. Hasta que no se recibe el carácter, el puerto serial no muestra ningún tipo de información.
  Esta es la representación de su funcionamiento:



Fuente: https://github.com/contiki-os/contiki/wiki/Input-and-output

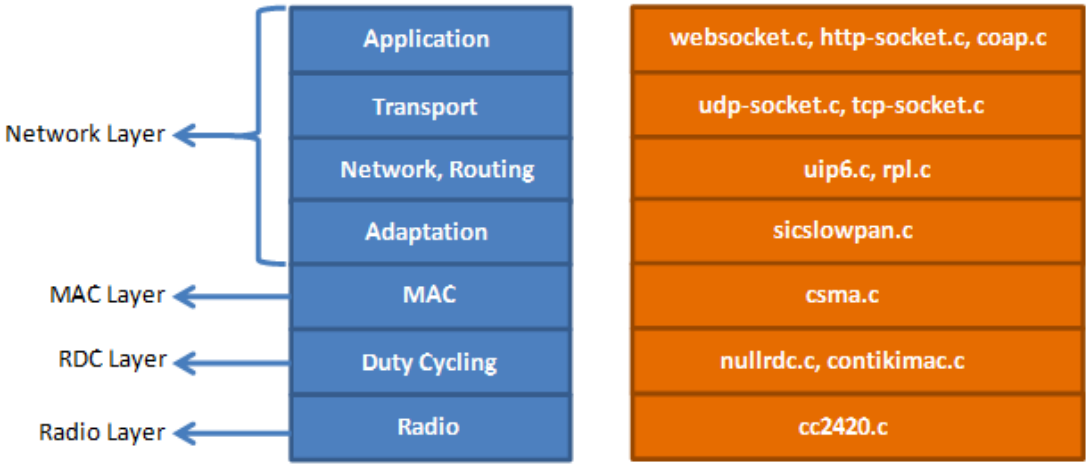
La librería LED está totalmente implementada en Contiki y teóricamente está soportada por todas las plataformas compatibles. Su uso es muy sencillo y la implementación es totalmente legible:

```
 /* Inicia el controlador de LED. */
void leds_init(void);
/* Devuelve el estado de un LED. */
unsigned char leds_get(void);
/* Enciende los LEDs indicados. */
void leds_on(unsigned char ledv);
/* Apaga los LEDs indicados. */
void leds_off(unsigned char ledv);
/* Cambia el estado. */
void leds_toggle(unsigned char ledv);
/* Invierte el estado. */
void leds_invert(unsigned char ledv);
```

Red en Contiki

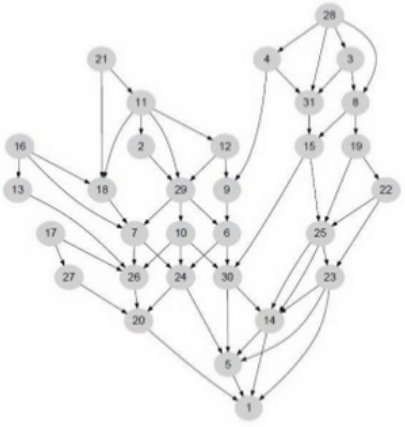
Contiki provee tres pilas de red: IPv4, IPv6 y Rime. También introduce la pila uIP, diseñada por el desarrollador de Contiki. Esta pila es una implementación muy ligera de TCP/IP, con el objetivo de ser ejecutada en dispositivos con recursos limitados. El tamaño de uIP es realmente pequeño en comparación de otras pilas basadas en TCP/IP.

La adaptación de la pila de red está hecha con archivos propios de Contiki, en esta imagen se aprecia como ha sido implementada cada capa:



Fuente: http://anrg.usc.edu/contiki/index.php/Protocols_stack

- **Capa de red**
  Contiki adapta la capa de aplicación, transporte, red y adaptación en una sola capa, la de red.
  Contiki crea de forma automática una red inalámbrico IPv6 mediante el protocolo previamente analizado RPL. Mediante este protocolo, se crea un grafo acíclico llamado DODAG (*Destination Oriented Directed Acyclic Graph*):



  Contiki también soporta TCP y UDP, y tiene una implementación de los protocolos de la capa de aplicación HTTP y CoaP.

- **Capa MAC**
  Contiki usa CSMA/CA (*Carrier Sense Multiple Access with Collision Avoidance*) en esta capa. En resumen, este protocolo funciona de la siguiente manera: antes de enviar los datos, el dispositivo comprueba que el medio de transmisión compartido esté libre. Si lo está durante un tiempo determinado, transmite los datos. Si no, el

dispositivo espera un tiempo semialeatorio antes de intentarlo de nuevo.

- **Capa RDC (*Radio Duty Cycling*)**
  Esta capa se encarga de controlar el transceptor del dispositivo, con el principal objetivo de mantenerlo apagado la mayor cantidad de tiempo posible para ahorrar energía.
  Para determinar los ciclos que el transceptor se enciende, Contiki provee tres diferentes mecanismos: ContikiMAC, X-Mac y LPP. El primero de ellos es el diseñado específicamente para el sistema Contiki, mientras que los otros son adaptaciones de mecanismos previamente existentes.

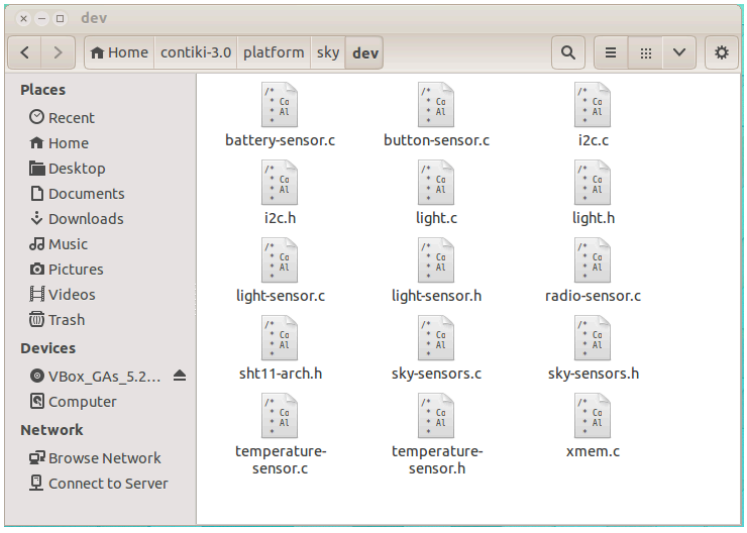
- **Capa de radio**
  Es la capa final de la red de Contiki. Los datos son recibidos en forma de bytes o paquetes vía manejadores de interrupciones.

Sensores en Contiki

Contiki ofrece macros genéricas para acceder a los sensores de los dispositivos. Gracias a esta genericidad, permite compatibilizar la gran variedad de sensores que se encuentran en los diferentes dispositivos. Para activar, leer el valor de un sensor y después desactivarlo, sólo se necesita esta pieza de código:

```
SENSORS_ACTIVATE(sensor);
value = sensor.value(0);
SENSORS_DEACTIVATE(sensor);
```

Como muestra la imagen, el código es fácil y legible. El nombre de la variable sensor esta definida en cada implementación de dicho sensor.
Las implementaciones de los sensores de cada plataforma se pueden encontrar en *contiki/platform/PLATFORM_NAME/dev*.
Por ejemplo, para el dispositivo sky, los sensores se encuentran en *contiki/platform/sky/dev*:



En esta carpeta se encuentran las implementaciones de diversos sensores, como el de temperatura (*temperature-sensor*), que mide la temperatura dentro del microcontrolador, de luz (*light-sensor*), que mide la intensidad de luz ambiente o de humedad y temperatura (*sht11-arch*), que puede medir la temperatura y la humedad del ambiente.

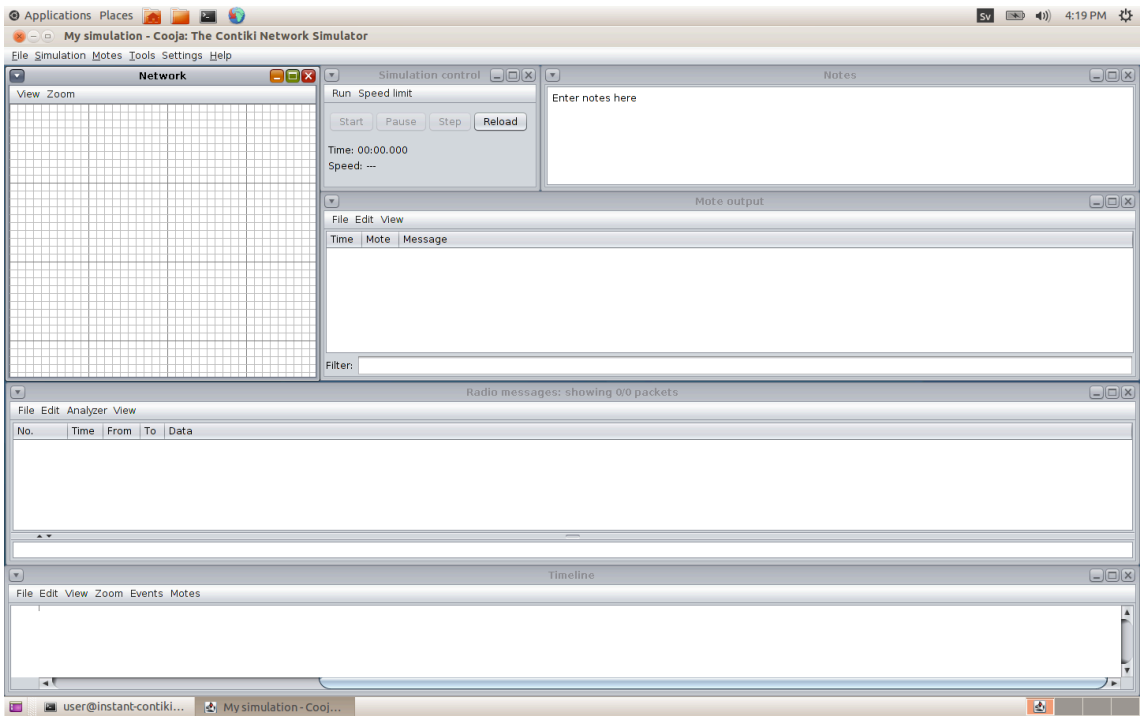
Ahorro de energía en Contiki

Como se ha mencionado antes, Contiki implementa tres mecanismos de ciclo de trabajo del transceptor: ContikiMAC, X-MAC and LPP.

Para conseguir un tiempo de vida de la batería más largo, una radio de bajo consumo no es suficiente. Los mecanismos de ciclo de trabajo son necesarios para mantener el transceptor el mayor número de tiempo apagado.
Otra técnica implementada para el ahorro de energía es la escucha de bajo consumo, donde los receptores encienden la radio periódicamente para comprobar si un mensaje está siendo transmitido y apagarse en caso contrario.

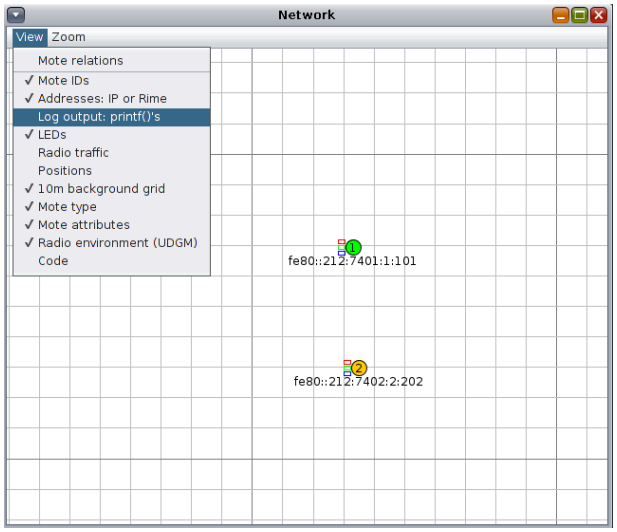
El simulador de Contiki: Cooja

Una de las grandes fortalezas de Contiki es su simulador, Cooja.
Esta herramienta permite simular redes Contiki con dispositivos reales. Cooja posee a su vez un simulador de dispositivos de la serie MSP430, MSPSim. Gracias a este simulador, Cooja puede emular el comportamiento de los dispositivos a nivel de instrucción y obtener datos aproximados de cómo sería su funcionamiento real.
Además, Cooja también ofrece una interfaz visual para controlar las comunicaciones de una forma mucho más sencilla:



Cooja ofrece una gran cantidad de opciones para las redes en Contiki. Las más destacadas son las siguientes:

- **Ventana de red**
  Esta ventana muestra la posición de los dispositivos. Estos simulan un alcance de señal real, que también se puede mostrar en esta ventana. Hay diferentes opciones disponibles que aportan información sobre los nodos, como la dirección IP, los LEDs encendidos, etc:

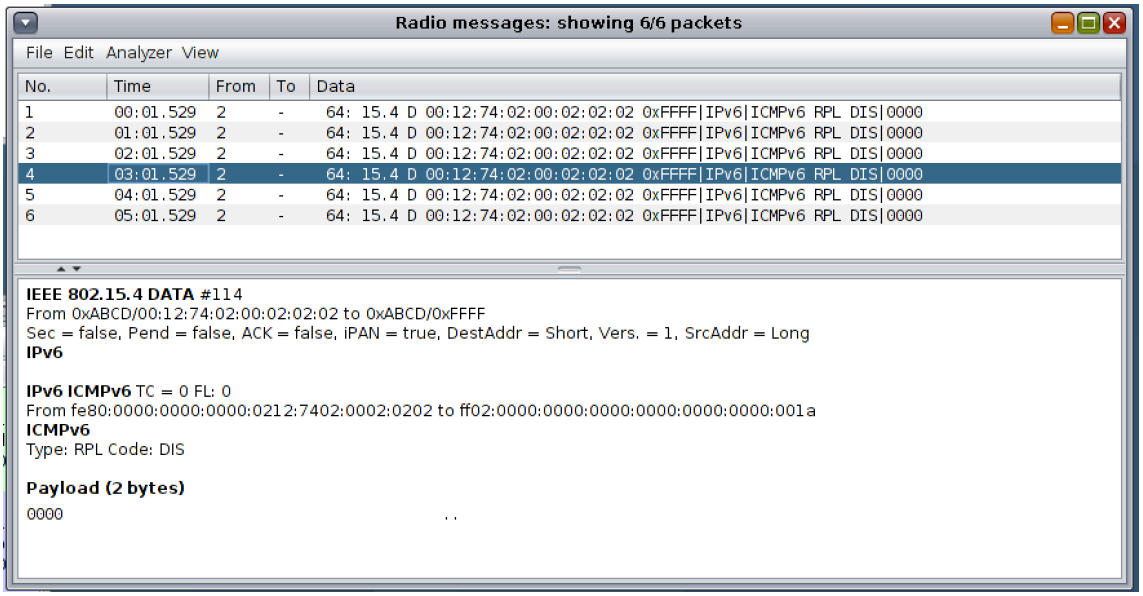


- **Ventana de control de tiempo**
  Esta ventana permite comenzar la simulación, detenerla, reiniciarla y incluso ejecutarla por pasos de un milisegundo, lo cual es muy útil para depuración. También permite ajustar la velocidad de simulación, para facilitar la prueba de sistemas con pocos eventos o eventos que ocurren tras una gran cantidad de tiempo.

- **Serial de dispositivos**
  Muestra todos los mensajes que los dispositivos muestran por su puerto serial. Dispone de un filtro, útil para la depuración.

- **Linea de tiempo**
  La linea de tiempo muestra los eventos que ocurren a través del tiempo. Puede ser utilizada para observar actividades de dispositivos individuales o bien las interacciones entre dos o más dispositivos. Cada dispositivo tiene su linea de tiempo. Cada tipo de evento tiene un color diferente.

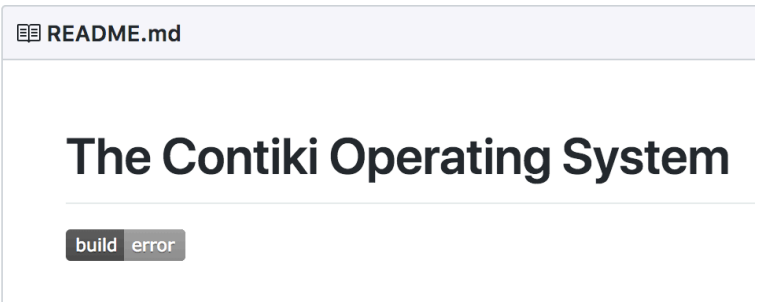
- **Mensajes de radio**
  Registra cada mensaje que se envían los nodos. Provee un analizador IPv6, que decodificacaído los mensajes y los divide en campos para facilitar la lectura.



Cooja soporta la simulación de los siguientes dispositivos: Tmote Sky, Wismote, Zolertia Z1, ESB y nativo (se crea un dispositivo mediante Java con todas las opciones de Contiki).

Comunidad de Contiki

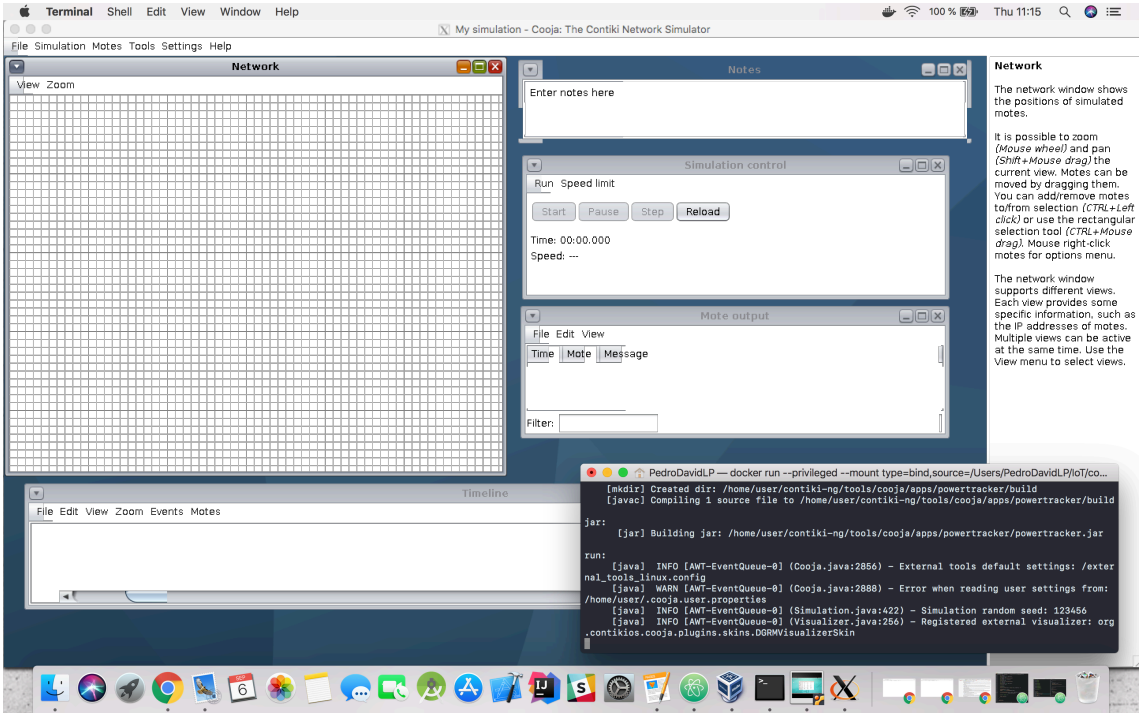
La comunidad de Contiki no es muy grande, y la actividad del proyecto principal ha caído significativamente los últimos años. Los cambios mas recientes fueron en 2017 y, a pesar de que la última versión del proyecto principal fue la versión 3, casi toda la documentación está escrita para la versión 2.7, la cual no es 100% compatible con la última versión.
La versión 3 de Contiki fue lanzada el 25 de agosto de 2015 y no ha recibido más actualizaciones desde entonces.
El proyecto oficial en Github falla desde hace 2 años y el desarrollador que empezó el proyecto, Adam Dunkels, parece que ya no colabora en su desarrollo.



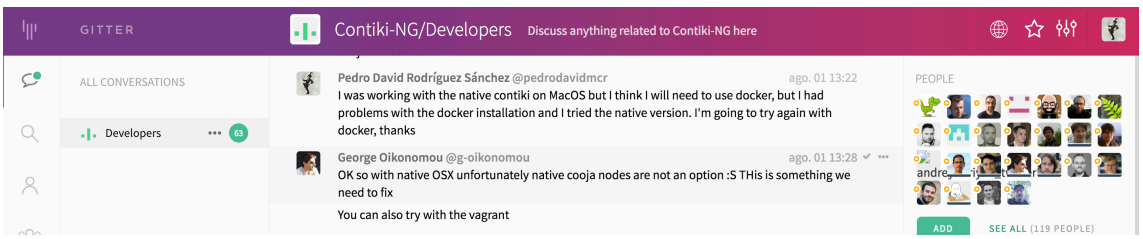
Error presente en el repositorio de ContikiOS.
Fuente: https://github.com/contiki-os/contiki

El número de preguntas en Stack Overflow sobre Contiki no supera las 1000 preguntas y la documentación de Contiki 3 es casi inexistente.

A pesar de que el proyecto original parece que está prácticamente muerto, un grupo de desarrolladores empezó un proyecto a partir del proyecto original en noviembre de 2017, llamado Contiki-NG (*Next Generation*). Este proyecto aún tiene la mayoría de características del original Contiki, pero ya ha sido mejorado en varios aspectos, como añadiendo compatibilidad. El anterior Contiki solo podía ser ejecutado en algunas versiones de Linux, mientras que ContikiNG se puede usar en MacOs, Windows, Linux, como imagen Vagrant y con Docker. Por ejemplo, así se ve el simulador Cooja ejecutándose con la imagen Docker y usando XQuarz como sistema de ventanas en el sistema operativo MacOS High Sierra:



El proyecto está muy activo, la versión actual del proyecto es la 4.2, lanzada en noviembre de 2018.
Este proyecto parece realmente prometedor. Tienen una wiki en Github, la cual cada vez es más grande. También tienen una comunidad en Gitter, donde los propios desarrolladores contestan las dudas de los desarrolladores:



El principal problema de Contiki-NG es el poco recorrido que tiene, ya que solo llevan un año con el proyecto. Hay una gran cantidad de bugs aún no resueltos, como la incompatibilidad con los nuevos sistemas de MacOS (High Sierra y Mojave no están soportados, sólo se puede ejecutar mediante su imagen Docker).
Por lo menos, los desarrolladores son muy colaborativos y activos en la comunidad, y continuan trabajando en el proyecto.

Compatibilidad de Contiki con el proyecto

El proyecto original de Contiki ya no parece una opción posible para cualquier proyecto, ya que se encuentra desactualizado. El nuevo sistema heredero, ContikiNG, es realmente prometedor y parece que será muy útil para el desarrollo de aplicaciones IoT en un futuro.

El gran problema que presenta ContikiNG es el corto tiempo de vida que lleva en desarrollo, lo que provoca que no exista una documentación extensa, estabilidad y soporte de errores. La dificultad del desarrollo de programas con este sistema operativo es reconocida hasta por su propio diseñador, Adam Dunkels, quien dijo en un post: “los sistemas Contiki son simplemente demasiado difíciles de desarrollar. Tienes que se un experto en Contiki sólo para saber por dónde empezar.”[10]
Esta gran dificultad y la inestabilidad antes mencionada descartan este sistema como el elegido para la realización del proyecto.

A pesar de todo, se reconoce a este sistema operativo como el idóneo para una versión futura del proyecto, que necesite un control de gasto de energía mucho más avanzada. Adicionalmente, las características que presenta el hardware compatible con el sistema cumple todos los requisitos establecidos. La placa elegida para el desarrollo del proyecto con este sistema operativo sería la previamente comentada *CC2650*.

TinyOS

TinyOS es un sistema operativo de licencia abierta y una plataforma de desarrollo para dispositivos inalámbricos de bajo consumo. Se creó en 1999 y está orientado a dispositivos empotrados.
Está escrito en nesC, un dialecto del lenguaje C que incluye optimizaciones para este tipo de dispositivos.
Como Contiki, TinyOS también tiene un simulador llamado TOSSIM, pero presenta un gran número de limitaciones en comparación a Cooja. Este simulador solo es compatible con la plataforma MicaZ y no posee interfaz gráfica, con lo cual es una opción mucho más tediosa que la que Contiki posee.

Compatibilidad de plataformas

TinyOS tiene un gran número de dispositivos soportados. Algunos son compartidos con Contiki, como el Zolertia Z1 y el TMote Sky.
Uno de los problemas que presenta TinyOS es la dificultad para un usuario normal de encontrar estas plataformas, ya que no todas tienen una tienda online donde realizar el pedido. Las más fáciles de comprar son:

Mulle sensor mote

Es un sistema empotrado inalámbrico en miniatura, adecuado para sensores inalámbrico conectados al Internet of Things y diseñado para un prototipado rápido. Está equipado con un mitrocontrolador de ultra-ahorro de energía y una radio IEEE 802.15.4.
El mínimo precio con la antena equipada es de 139 €. Además, no hay una manera de comprarlo online, hay que comunicarse con el vendedor vía correo electrónico. [11]

Tmote Sky for TinyOS (CM3300)

El dispositivo CM3300 es un dispositivo con sensores compatible con IEEE 802.15.4 basado en el diseño de la plataforma de licencia abierta “TelosB” diseñada y publicada por la Universidad de Berkeley, California.
Su precio es de 105€, lo cual sigue siendo un precio demasiado elevado para los requerimientos del proyecto. Su compra se puede realizar mediante su tienda online.[12]

Hay más plataformas disponibles, pero no es fácil encontrar dónde se pueden comprar los dispositivos. Adicionalmente, los precios son significativamente más altos, con lo cual este sistema operativo es bastante incompatible con los objetivos del proyecto.

Funcionamiento de TinyOS

La documentación de el funcionamiento de TinyOS es muy breve en comparación a la del funcionamiento de Contiki, ya que en las guías se centran en la programación de dispositivos con TinyOS.
Los programas en TinyOS se desarrollan mediante componentes de software combinados con abstracciones del hardware. Los componentes se conectan unos a los otros mediante interfaces. TinyOS ofrece interfaces y componentes para abstracciones comunes como comunicación de paquetes, enrutado, recolección de datos de sensores, actuadores y almacenamiento.
En contraste con Contiki, TinyOS tiene un comportamiento de hilos no bloqueante y solo tienen una pila de llamada. Todas las operaciones de entrada y salida que duran

más que unos cientos de microsegundos se comportan de forma asíncrona y responden con una callback.
Esta característica de solo tener una pila de llamada obliga a los programadores a construir una lógica compleja mediante la union de pequeños manejadores de eventos.
El código de TinyOS se une estáticamente con el código del programa y se compila en una pequeña librería mediante una herramienta GNU.

Red en TinyOS

La red en TinyOS está implementada con la pila BLIP(*Berkeley Low-power IP*). Esta es una implementación del protocolo IP.
Esta implementación permite formar redes multi-hop que se comunican mediante protocolos compartidos. BLIP no obliga a usar ningún protocolo en especifico, sino que ofrece compatibilidad con algunos protocolos. Por ejemplo, ofrece total soporte para el protocolo UPD y tiene una implementación de CoaP basada en la librería de CoaP para C, libcoap [13].

Sin embargo, el soporte protocolo TCP aún se encuentra en estado experimental y no garantiza una buen desempeño o fiabilidad. Por ejemplo, BLIP no adapta el tamaño del buffer necesario para la transmisión de los paquetes, con lo cual hay que hacerlo a mano, provocando que escribir un programa usando TCP sea dificultoso.

Sensores en TinyOS

Para usar y leer información de los sensores en TinyOS, hay unas interfaces estándar de adquisición de datos: Read, ReadStream and ReadNow. Estas interfaces esconden los detalles de la configuración de los sensores. De esta manera, no hay ninguna forma de saber cual es el sensor que esta conectado a la interfaz:

```
interface Read<val_t> {
  /** Inicia la lectura de un valor.
   * @return SUCCESS si se devuelve un evento readDone()
   */
  command error_t read();
  /** Señala la finalización de read().
   * @param result SUCCESS si se ha completado con éxito read()
   * @param val valor leído
   */
  event void readDone( error_t result, val_t val );
}
```

En consecuencia, al contrario de Contiki, se debe implementar el código para cada dispositivo diferente.

Comunidad de TinyOS

La comunidad de TinyOS es bastante más grande que la comunidad de Contiki. Al ser un proyecto más grande y más antiguo, hay mucha más documentación. Además, TinyOS empezó como un proyecto en la Universidad de Berkeley, formando parte del programa DARPA NEST, un programa de la agencia de investigación militar de Estados Unidos. Con estas fuertes bases, TinyOS ha sido muy usado en el campo de los sistemas empotrados.

Actualmente, el repositorio de el TinyOS académico sigue existiendo, pero sin actividad, ya que la actividad se ha movido a tiny-prod, un sistema operativo orientado más a la producción industrial de sistemas empotrados.
A pesar de esto, TinyOS tiene una gran cantidad de documentación y varios libros donde puedes aprender a programar con este sistema operativo, como el libro TinyOS Programming. [14]

Compatibilidad de Contiki con el proyecto

TinyOS, al ser un sistema operativo más orientado a los sistemas embebidos y a la producción industrial de estos, además de la dificultad de encontrar distribuidores de dispositivos compatibles y sus altos precios, convierte este sistema operativo en incompatible con el objetivo del proyecto.
El elevado precio por sí solo ya descarta totalmente esta opción, pero además se reconoce este sistema como el adecuado para procesos mucho más complejos. La capacidad de procesamiento que necesita el sistema en este proyecto es muy pequeña, con lo cual no son necesarias las prestaciones ofrecidas en TinyOS.
Otras deficiencias son el poco soporte del protocolo TCP y la mayor dificultad para abstraer los sensores de una plataforma en el código.

Por estas razones se ha decidido descartar totalmente el sistema operativo TinyOS para las siguientes fases del proyecto.

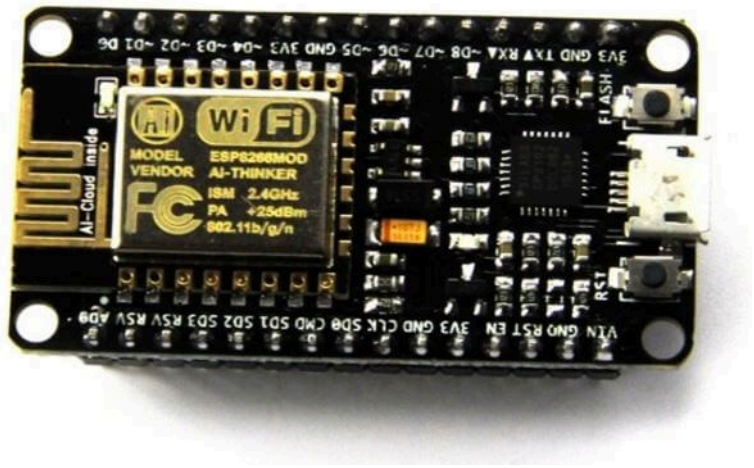
Plataformas de desarrollo

En el mercado actual existen una gran variedad de plataformas de desarrollo, con un rango de precios y funciones muy amplio. Una de las plataformas más conocidas es Arduino, la cual se ha popularizado estos últimos años en el campo del IoT. Estas plataformas pueden combinarse con otros módulos para ampliar su funcionalidad, como sensores y antenas.
Uno de los módulos más conocidos en este campo es el ESP8266, que permite conexión WiFi y su coste es realmente bajo. Gracias a este módulo se han creado un gran número de placas que juntan este módulo con un microcontrolador para así crear kits de desarrollo a muy bajo precio, como son los kits Wemos D1[15] y NodeMCU [16].

Wemos D1 tiene un precio que ronda los 2€ y fue inicialmente elegido como el kit de desarrollo para el proyecto. Sin embargo, tras su compra fue imposible la programación de este a través del entorno de programación Arduino, ya que podrujo errores de conexión con el ordenador y resultó imposible el acceso al firmware. Dada está situación, fue descartado y se eligió NodeMCU, que posee unas características similares.

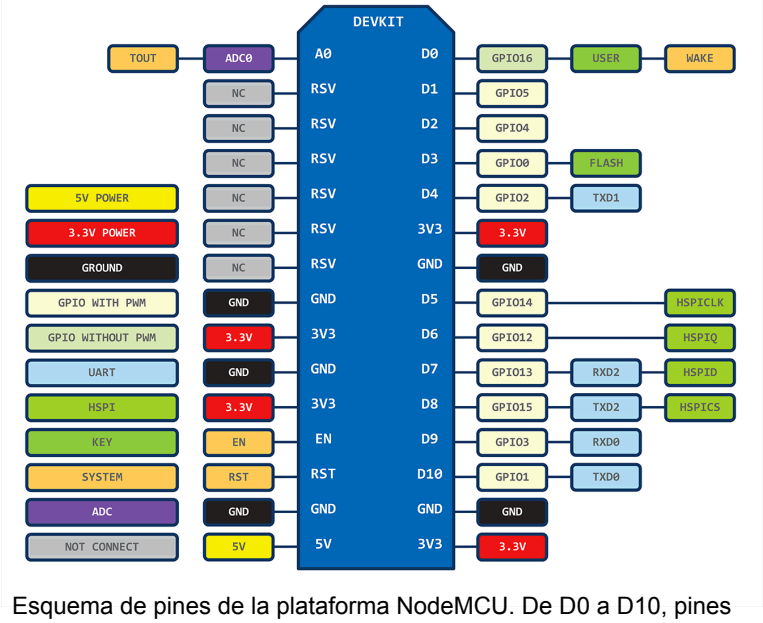
NodeMCU

El kit de desarrollo NodeMCU es un dispositivo basado en el modulo ESP8266. Esta plataforma es de licencia abierta a nivel hardware y software y tiene un microcontrolador ligero.



De todas las plataformas disponibles, esta ha destacado por su bajo precio y su facilidad de uso. El precio ronda los 2€ en Aliexpress. Esta plataforma se puede programar directamente con el entorno de programación que ofrece Arduino o con el lenguaje de programación Lua. Hay mucha documentación para la plataforma, librerías y soporte para los protocolos de comunicación más usados en el Internet of Things, como CoaP. [17]

Esta plataforma no ofrece ningún tipo de sensor, pero posee 11 puertos de entrada y salida digitales y uno analógico, con lo cual puede satisfacer el requisito de ejecutar la medida de la temperatura, humedad y luz.



Esquema de pines de la plataforma NodeMCU. De D0 a D10, pines digitales. A0 es su único pin analógico
Fuente: https://developer.ibm.com/tutorials/iot-nodemcu-open-why-use

Respecto al consumo de energía, NodeMCU ofrece 3 modos de ahorro de energía:

- ***Modem-sleep***
  Este es el modo por defecto en el cual funciona esta placa. Este modo intenta mantener la antena WiFi el mayor tiempo posible desactivada, ya que es el componente con mayor consumo de energía.
- ***Light-sleep***
  Este modo desactiva la antena WiFi, desactiva el reloj del microcontrolador y mantiene la CPU suspendida, pero sin apagarla.
- ***Deep-sleep***
  Mediante este modo, la placa se desactiva casi totalmente (sólo mantiene despierto el reloj de tiempo real, RTC) y duerme durante un número finito de microsegundos. Para habilitar el modo deep-sleep de la placa NodeMCU se debe conectar el pin *RESET* con el pin digital D0, que como muestra la imagen mostrada previamente, contiene la función *WAKE*.
  La placa limita tiempo de deep-sleep a unos 71 minutos. [18]

Cada uno de estos modos de funcionamiento presenta un consumo de energía diferente:

Item		Modem-sleep	Light-sleep	Deep-sleep
Wi-Fi		OFF	OFF	OFF
System clock		ON	OFF	OFF
RTC		ON	ON	ON
CPU		ON	Pending	OFF
Substrate current		15 mA	0.4 mA	~ 20 μA
Average current	DTIM = 1	16.2 mA	1.8 mA	-
	DTIM = 3	15.4 mA	0.9 mA	
	DTIM = 10	15.2 mA	0.55 mA	

Corriente utilizada en los diferentes modos de ahorro de energía. También se muestra qué partes de la placa están activas en cada momento. El parámetro DTIM depende del beacon del router al cual esté conectada la placa.
Fuente: https://www.losant.com/blog/making-the-esp8266-low-powered-with-deep-sleep

La placa NodeMCU se alimenta a 5 voltios.
Como se puede apreciar en la tabla, el consumo en modo *deep-sleep* es realmente bajo, con lo cual cumple el requisito de ahorro de energía, ya que la placa se encontrará en este estado la mayor parte del tiempo.

En resumen, esta placa cumple con los requisitos necesarios para el proyecto, ya que su precio es realmente bajo, el consumo energético se adapta a las necesidades establecidas, se puede conectar con sensores que recolecten la información requerida y su implementación es sencilla.

Sensores

Una gran parte de los dispositivos que funcionan con los sistemas ContikiOS y TinyOS incorporan sensores en la misma placa, como el SensorTAG compatible con Contiki. Pero en el caso de la plataforma NodeMCU, se necesita conectar los sensores adecuados a los pines de la placa para realizar las correspondientes medidas.
Por ello, se han elegido el sensor DHT22 y un módulo sensor fotosensible que usa el comparador de voltaje LM393.

Todos los precios que se mencionan en el análisis de estos sensores son de la aplicación Aliexpress.

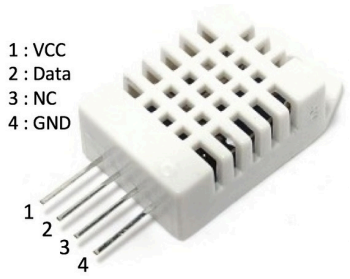
Sensor de temperatura y humedad

En el proceso de la elección del sensor de humedad y temperatura se ha hecho una comparativa entre dos candidatos potenciales: el sensor DHT22 y el DHT11. El sensor DHT11 tiene un costo de 0,55€, lo que lo convertía en la opción más atractiva frente a los 2,03€ que cuesta el DHT22. Aunque ambos precios son bajos, el mayor objetivo del proyecto es la reducción máxima de costes, con lo cual se debe valorar si las ventajas que ofrece el DHT22 son necesarias.
La comparativa entre las características de ambos se resume en esta tabla [19]:

Característica	DHT11	DHT22
Rango de medición de temperatura	0 ℃ ~ 50 ℃	-40 ℃ ~ 80 ℃
Precisión en la medición de temperatura	±2 ℃	±0,5 ℃
Precisión en la medición de humedad	± 5%	± 2%

Dado que el sensor DHT11 puede tener problemas de medición en temperaturas inferiores a los 0 grados y que el sistema de control y riego debe poder realizar el cuidado de plantas con necesidades muy específicas, se ha llegado a la conclusión de que el sensor DHT22 es necesario a pesar de que su precio es más elevado. Gracias a la precisión de medida de humedad del sensor DHT22 se puede controlar más fiablemente cuando una planta necesita ser regada.

El sensor DHT22 funciona con un voltaje de 3’3V, lo cual lo convierte en compatible con la alimentación que proporciona la propia placa NodeMCU. Para la lectura de datos existen librerías para el entorno de programación Arduino. Los datos se transmiten de forma digital, así que puede ser conectado a uno de los pins digitales de la placa. El consumo medio del sensor es de solamente 0,3 mA. [20]



Pins del sensor DHT22. El pin número 3 no tiene utilidad para el usuario, simplemente es utilizado en su fabricación para realizar pruebas de calibración. Fuente: https://www.espruino.com/DHT22

Módulo sensor fotosensible LM393

Este módulo basa su funcionamiento en una fotoresistencia que posee. Posee dos salidas, una digital que indica cambios en intensidad lumínica y una analógica que ofrece una medida de la intensidad que recibe el sensor. Este sensor ha sido elegido por su muy bajo precio, ya que se puede encontrar por 0’27 € y sus características son aceptables para los requerimientos del proyecto.

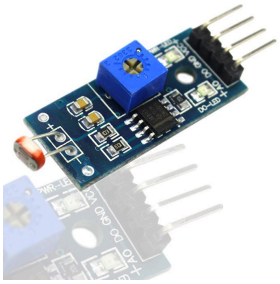


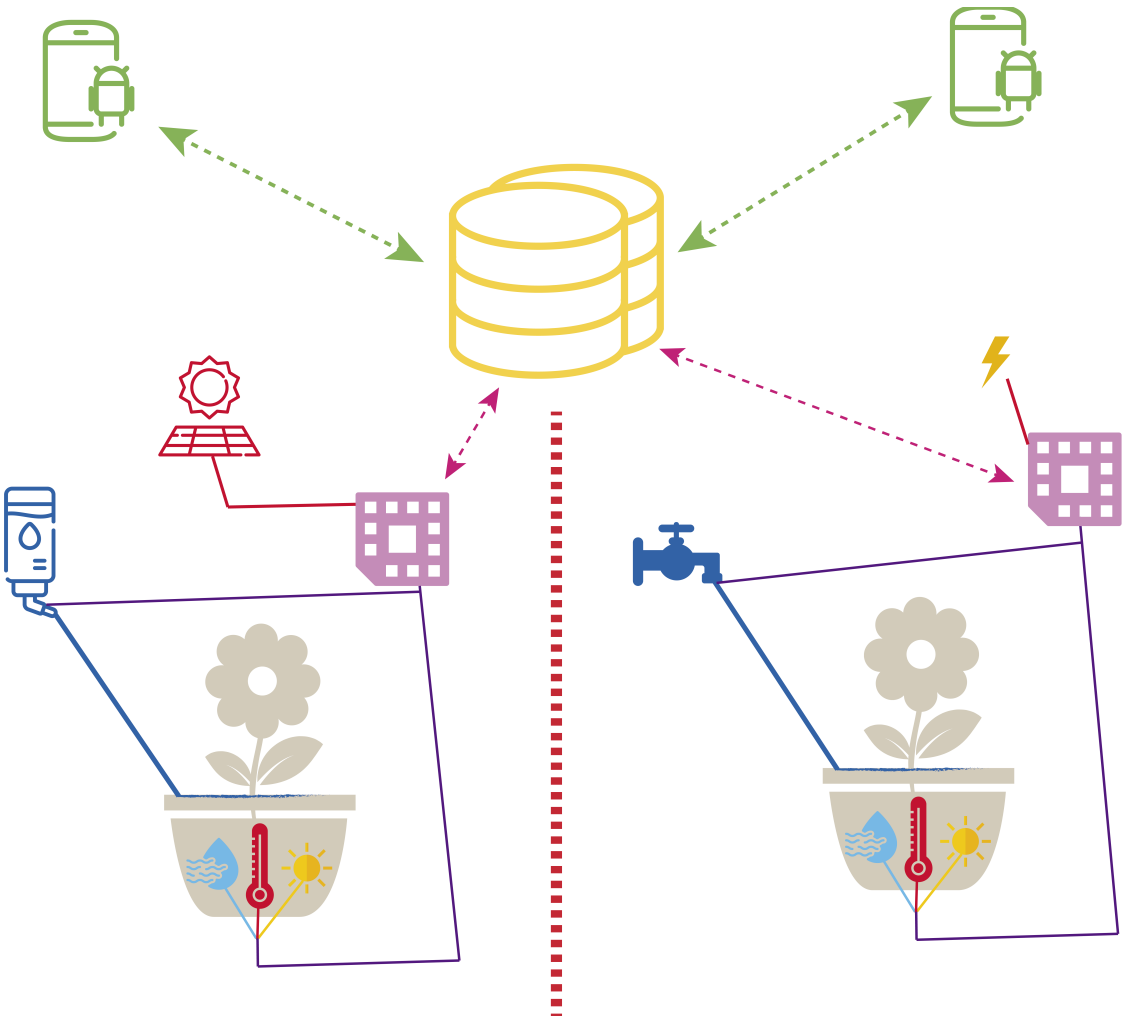
Imagen del módulo.
Fuente: Aliexpress

Capítulo 4: Diseño de la solución

La propuesta de diseño es la creación de un sistema estructurado, con una parte fija que sea común a los usuarios, siendo esta la aplicación Android y los sistemas necesarios para la comunicación con la red IoT.
La otra parte será la propia red IoT, y esta será intercambiable, es decir, podrá ser implementada de diferentes formas (diferentes dispositivos y diferentes protocolos). Gracias a esta estructura, la parte que el usuario final tenga del producto se podrá ir renovando a nivel hardware pero todas las versiones anteriores serán compatibles, ya que la interfaz de control mediante la aplicación Android será constante. La funcionalidad se podrá aumentar, pero todas las redes IoT serán compatibles.

Arquitectura del sistema

La arquitectura general del sistema debe ser la siguiente:



La aplicación Android se comunicará con un intermediario, que será un servicio backend alojado en la nube. Este se encargará de almacenar la información sobre las plantas, los riegos y los modos de riego seleccionados para cada planta.

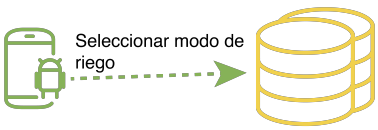
Cada dispositivo que controle una planta se comunicará directa o indirectamente (dependiendo de la estructura de la red IoT escogida) con el servicio backend. El dispositivo se encargará de enviar las muestras de temperatura, humedad y luz. También se encargará de realizar el riego de la planta mediante la apertura de una válvula.

Como se aprecia en la imagen, la fuente de energía del dispositivo podrá ser o bien una energía renovable (como la luz solar) o de la propia corriente de la casa. Por otro lado, el suministro de agua será independiente del sistema, se podrá conectar a un grifo o a un depósito.

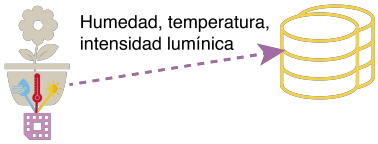
Diseño detallado

El funcionamiento del sistema de forma detallada será el siguiente:

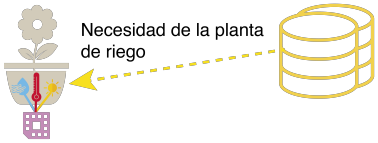
Se añade una planta al sistema

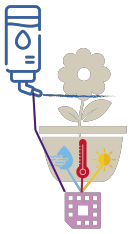


Desde la aplicación Android, se configura el modo de riego y se especifica la humedad requerida por la planta. La configuración se guarda en el servicio backend. La configuración se puede cambiar en cualquier momento.



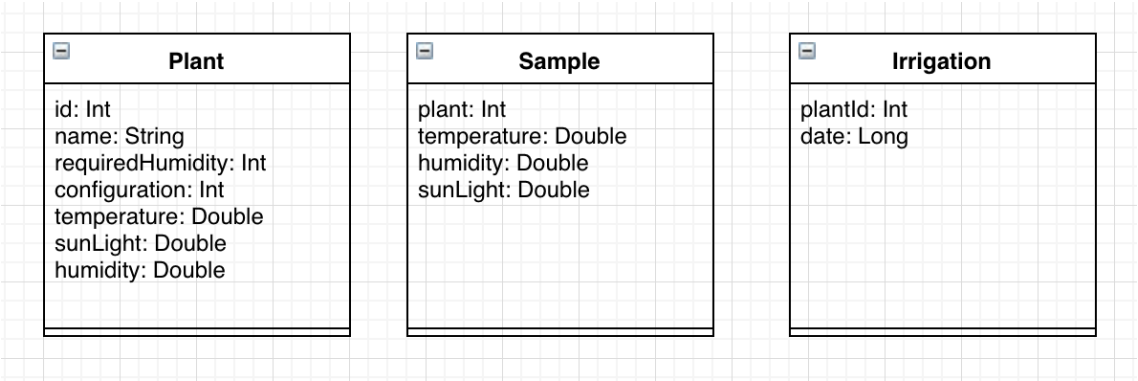
Cada hora el dispositivo recoge una muestra de humedad, temperatura y intensidad lumínica. Después, la envia al servicio backend.
Después de enviar la muestra, el dispositivo comprueba si debe regarse. El servicio backend comprueba la configuración y la muestra enviada y envia los datos de riego (si debe regarse y con cuanta cantidad)





Si el dispositivo recibe que la planta debe regarse, se abre la válvula y se efectúa el riego hasta que la humedad conseguida sea el adecuado a las necesidades especifícadas de la planta.

El backend y la aplicación Android compartirán el diseño de las clases utilizadas:

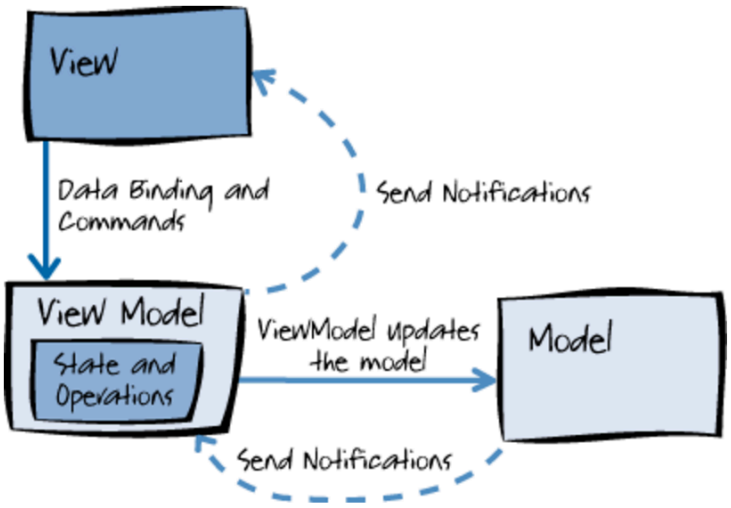


Los dispositivos que controlen las plantas solo conocerán la implementación de las muestras (*Sample*).

Los campos *temperature*, *sunLight* y *humidity* que se encuentran en la clase Planta son una elección que facilita el almacenamiento de la última muestra de sensores de una planta en específico. Estos valores son temporales y se irán reescribiendo.
Los campos *requiredHumidity* y *configuration* almacenarán un código que se corresponderá al porcentaje de humedad requerido y la configuración de riego respectivamente. La configuración de riego estará almacenada en una clase de enumeración, y tendrá 4 opciones:

- *BY_HUMIDITY*
  Esta configuración realizará el riego de la planta cada vez que la planta baje de la humedad establecida como requerida.
- *BY_TIME_SUNRISE*
  Esta configuración realizará el riego de la planta al amanecer.
- *BY_TIME_SUNSET*
  Esta configuración realizará el riego de la planta al anochecer.
- *BY_TIME_BOTH*
  Esta configuración combina las dos opciones anteriores.

Respecto a arquitecturas, la aplicación Android utilizará MVVM (*model-view-viewmodel*), la arquitectura que Google presentó en 2017 como la oficial de Android. Además, se han usado conceptos relativos a la arquitectura *clean* propuesta por el famoso diseñador de arquitecturas “Uncle Bob” (Robert C. Martin). [21]



Funcionamiento de MVVM. El ViewModel permite guardar los datos de una vista sin depender de su ciclo de vida, facilitando así tratar con los eventos de Android como girar la pantalla, pasar una aplicación a segundo plano, etc.
Fuente: https://android.jlelse.eu/exciting-secrets-about-mvvm-that-nobody-tells-you-a95548ea684b

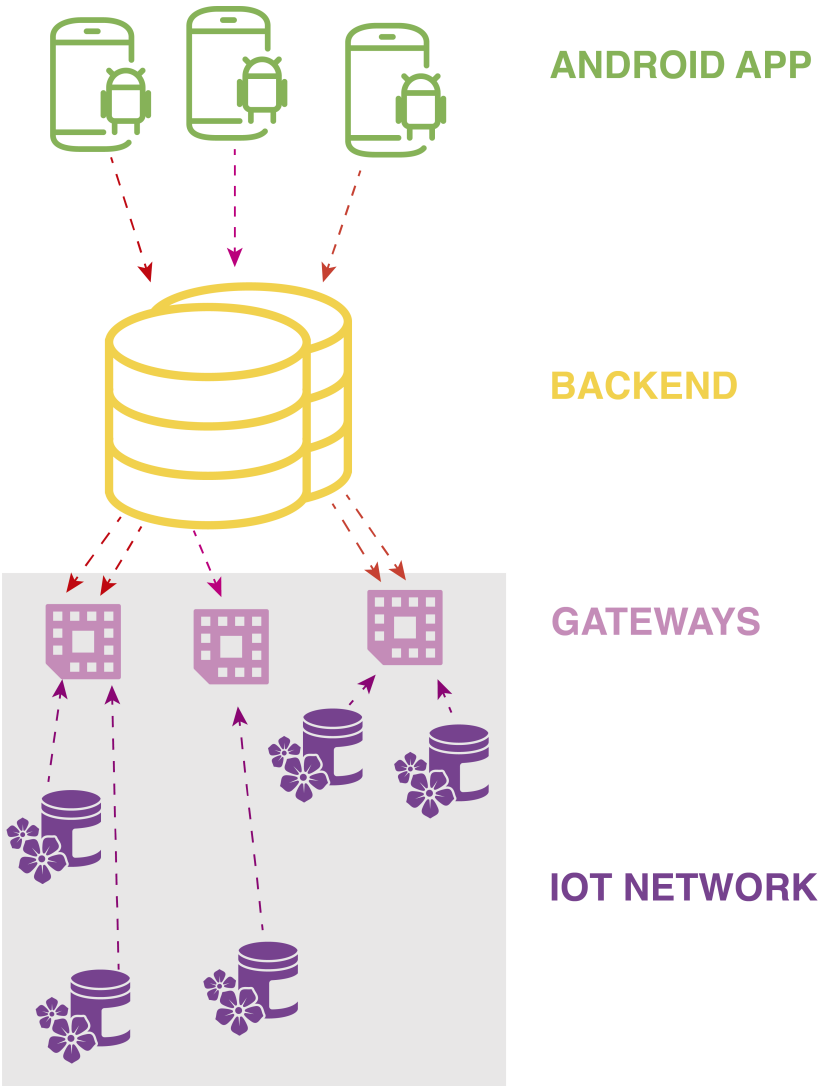
El backend será estructurado con una arquitectura de casos de uso. La API que será publicada será la siguiente:

METODO	RUTA	DESCRIPCIÓN
GET	/plants	Devuelve todas las plantas del sistema.
GET	/plant/{id}	Devuelve los datos de una planta
GET	/sample/{id}	Devuelve todas las muestras de temperatura, luz y humedad de una planta.
POST	/sample/{id}	Añade una muestra a una planta.
GET	/irrigate/{id}	Devuelve -1 si la planta no debe regarse. Si se debe efectuar el riego, devuelve valores de 0 a 3, dependiendo la humedad que requiere la planta.

PUT	/plant/{id}	Cambia la configuración de riego o la humedad requerida de una planta.
GET	/irrigation	Devuelve una lista con los últimos 10 riegos efectuados
GET	/numberirrigation	Devuelve el número de riegos efectuados en el día

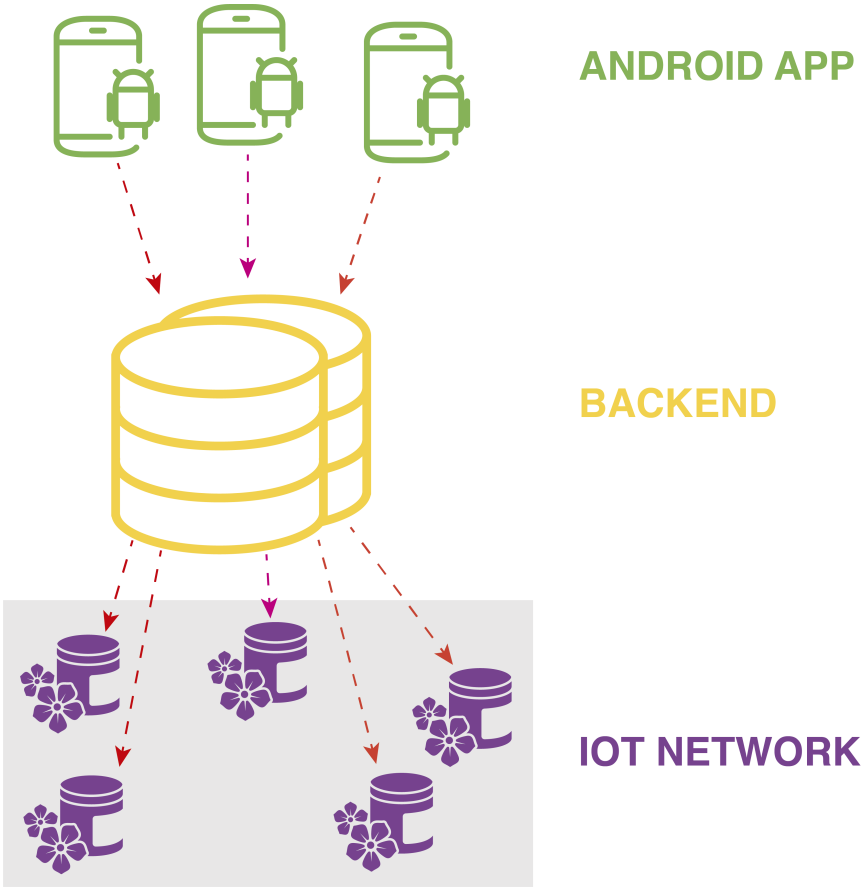
El fondo verde de la tabla indica que la ruta será accedida por la aplicación Android, mientras que el fondo rosa significa que es la aplicación IoT quien accederá a la consecuente ruta.
No será necesaria ninguna clase de seguridad en el servicio backend, ya que está fuera del ámbito del proyecto.

En cuanto a la red IoT, su diseño dependerá de la tecnología utilizada. Para su implementación con Contiki, la estructura debe ser la siguiente:

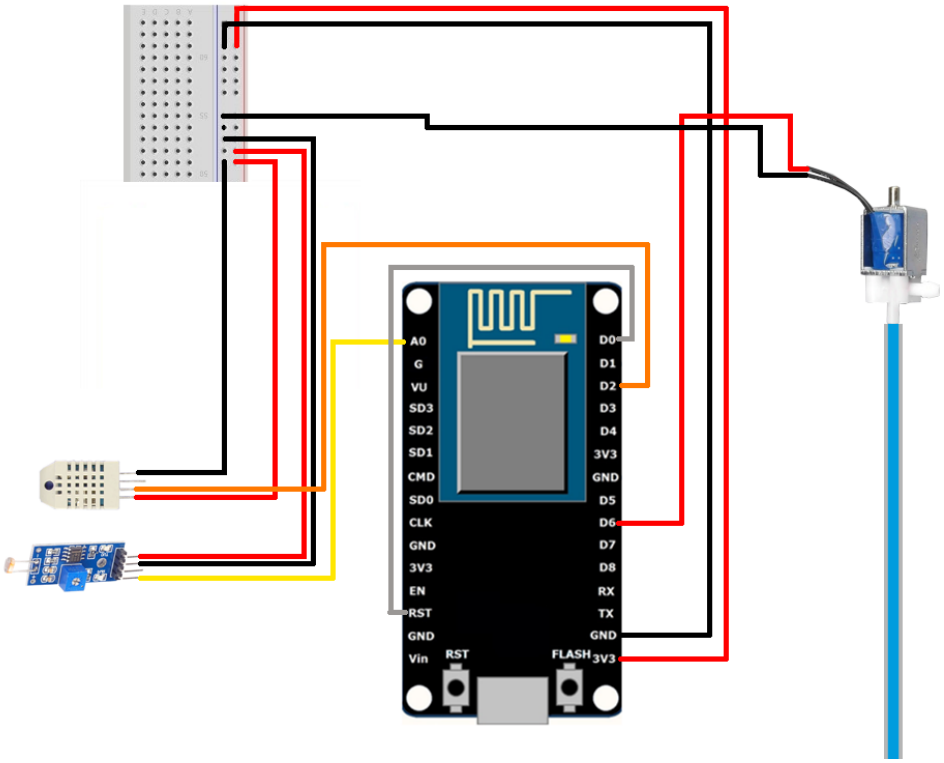


Ya que los dispositivos que tomarán las muestras con Contiki priorizarán el ahorro de energía, estos no serán los encargados del envío de los datos al backend. Se añadirá un coordinador que transmita los datos de los dispositivos al backend y viceversa de forma transparente. La comunicación entre los dispositivos y el coordinador se realizará mediante el protocolo CoaP, minimizando así el tamaño de los paquetes retransmitidos. Los dispositivos enviarán los datos al servidor CoaP, que será el coordinador y este enviará mediante una petición HTTP los datos de cada dispositivo de forma transparente. Esta implementación presenta la desventaja de una instalación de un dispositivo adicional al sistema, el coordinador.

Por otro lado, si la implementación se realiza con el kit de desarrollo elegido, NodeMCU, la estructura será:



Como la plataforma NodeMCU está diseñada para las comunicaciones WiFi, las comunicaciones se realizarán directamente con el backend.
Las conexiones que se realizarán a nivel físico con los sensores y la válvula en la placa NodeMCU están descritas en el siguiente esquema:



Como se puede apreciar, el pin *RESET* está conectado al pin D0 para permitir así el *deep-sleep* de la placa. También, el control de la válvula que deja pasar el agua se realiza mediante el pin D6.

Un buen intervalo de recogida de muestras seria de 1 hora, ya que en un día normal no hay cambios drásticos de ninguno de los parámetros que se van a medir en ese intervalo de tiempo. Así también se conseguirá disminuir el consumo eléctrico del dispositivo.

Tecnología utilizada

Para la aplicación Android y el backend el lenguaje de programación elegido ha sido Kotlin. Kotlin es un lenguaje de programación nuevo, expresivo, conciso y potente. Es totalmente interoperable con Java.
La elección de este lenguaje viene dada por la compatibilidad del entorno de programación de Android con el lenguaje (los diseñadores de Android Studio están fuertemente relacionados con la empresa desarrolladora de Kotlin, JetBrains).
También, se calcula que la media de lineas de código ahorrados programando en

Kotlin en vez que Java es de un 40%. Además, Google nombró en 2017 a Kotlin lenguaje oficial de Android.

En la aplicación Android se destaca el uso de las siguientes librerías:

- ***Fuel.*** [22]
  Utilizada para la interacción HTTP. Esta librería está diseñada para Kotlin, usando las ventajas del lenguaje.
- ***MPAndroidChart.*** [23]
  Utilizada para las gráficas necesarias. Es, sin duda, la librería open source de gráficas más versátil actualmente.

Respecto al backend, se ha elegido *Ktor* como framework. Ofrece el desarrollo de servidores asíncronos fácilmente usando la potencia del lenguaje Kotlin. [24]
También será necesaria la API pública *Sunrise Sunset* [25] para conocer el amanecer y anochecer de cada localización y día, ya que uno de los modos de riego que se incluirán en la configuración de la planta funcionará activando el riego en estos momentos del día.

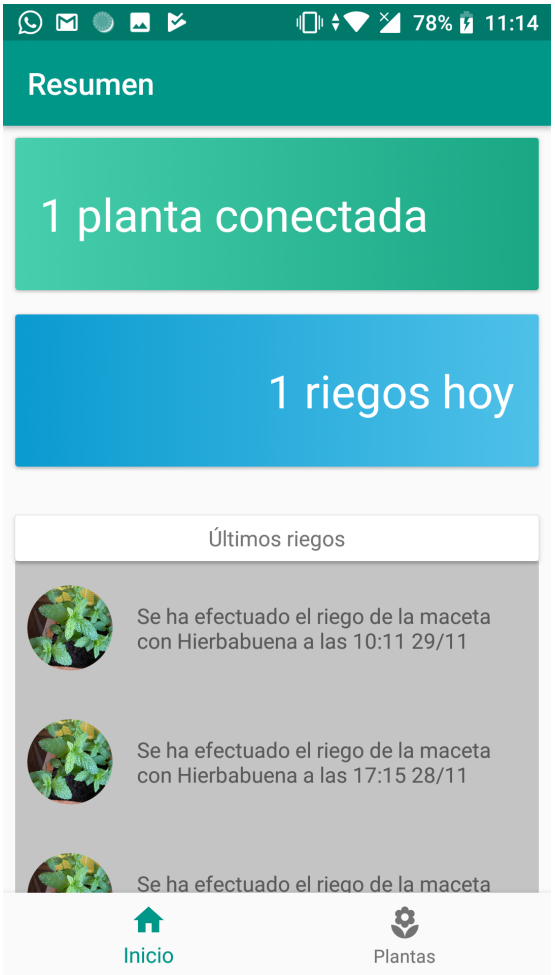
Capítulo 5: Implementación

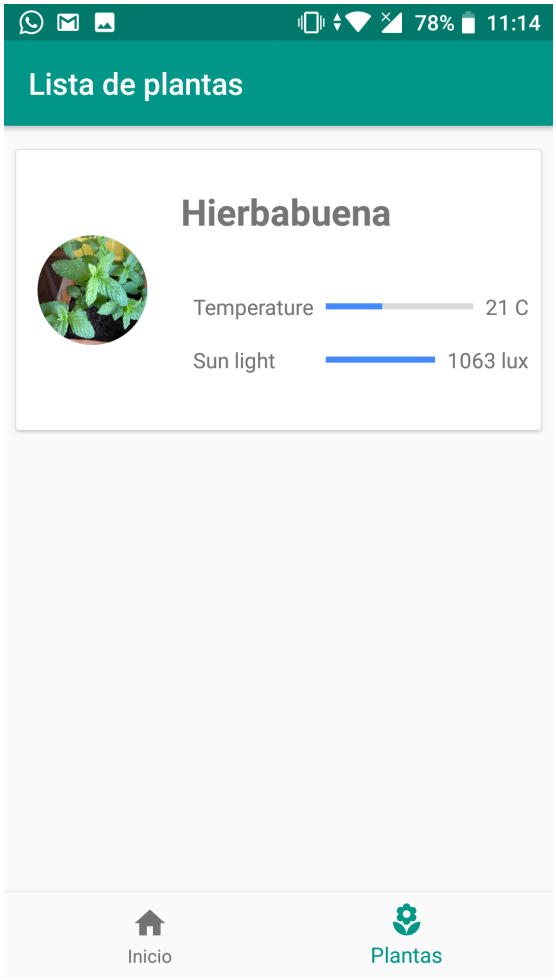
La implementación final de este proyecto ha sido realizada mediante una aplicación Android llamada a-garden, un servicio backend llamado a-garden y una red IoT con el kit de desarrollo NodeMCU.
También se ha intentado realizar y simular el funcionamiento de la red en ContikiOS, con el simulador Cooja.

Aplicación Android

La aplicación Android ha sido implementada con el diseño más sencillo y dinámico posible.
Consta de 4 pantallas:

Pantalla inicial
Esta pantalla es un resumen que muestra cuantas plantas hay conectadas y el numero de riegos efectuados en el día. Además, hay un registro de las últimos riegos efectuados, que incluye la hora y la fecha de cuando se han realizado.

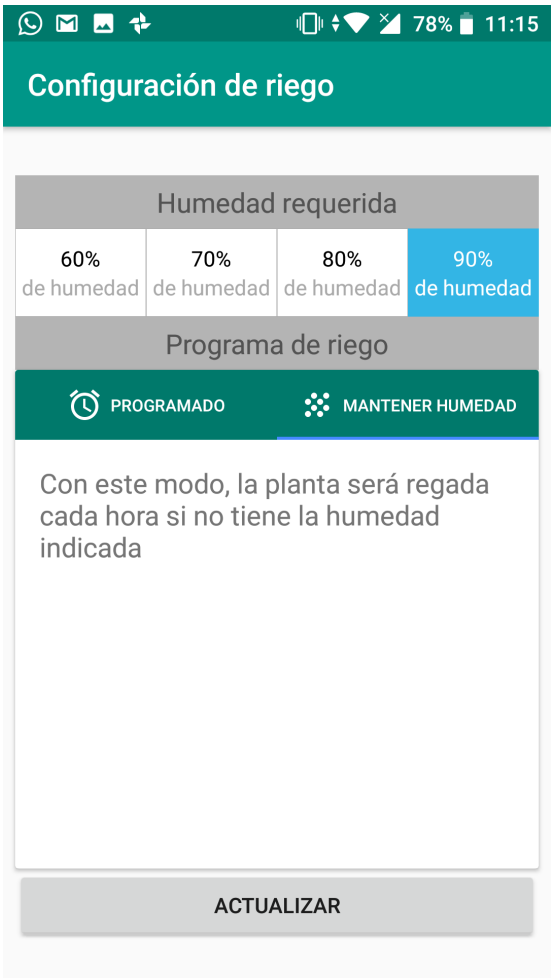




Lista de plantas
Esta pantalla muestra una lista de las plantas conectadas al sistema, mostrando la última medida de temperatura y intensidad de luz.

Detalle de la planta
Esta pantalla muestra los detalles de la planta (temperatura, humedad y luz), con un gráfico con las últimas medidas de temperatura y humedad.

Configuración de riego
Esta pantalla permite configurar el riego de las plantas , estableciendo la humedad necesaria para cada planta y seleccionando un modo de riego. Hay dos modos de riego: Mantener una humedad constante hora tras hora o regar la planta a ciertas horas del día.



Durante el desarrollo de la aplicación Android no se ha encontrado ningún problema a destacar, ya que los conocimientos de programación de este lenguaje eran sólidos. La única cuestión que ha consumido más tiempo ha sido la migración de la aplicación inicial a la arquitectura MVVM, ya que no había experiencia previa con esta arquitectura.

Servicio backend

La implementación del servicio backend se ha realizado a través de Heroku, una plataforma web (*Platform as Service*) que permite alojar una aplicación en la nube de forma gratuita. El plan gratuito ofrece 550 horas de ejecución de la aplicación al mes, con lo cual la aplicación se puede ejecutar durante aproximadamente 22 días sin interrupción de forma gratuita. Para realizar las pruebas y la implementación ha resultado ser la opción más interesante.

Heroku soporta varios lenguajes de programación, incluyendo Kotlin, el elegido para el desarrollo.
Durante el desarrollo y el despliegue del backend no se ha encontrado ningún problema destacable.

El servicio backend se aloja en la siguiente API: https://a-garden.herokuapp.com

Implementación de la red IoT con NodeMCU

Cada dispositivo NodeMCU se coloca acoplado con su maceta. En el proceso de implementación se ha debido de tener en cuenta el orden de las acciones para evitar utilizar la antena WiFi durante el mayor tiempo posible. Este ha sido el orden seguido:

```
void setup() {
  pinMode(D6, OUTPUT);
  configureDTHSensor();
  readDataFromSensors();
  connectToWiFi();
  sendSensorDataToServer();
  getIrrigationConfig();
  needToIrrigate();
  ESP.deepSleep(3600e6);
}
```

Esta serie de funciones es la que se repite una y otra vez en la ejecución del dispositivo. En rojo se han marcado las funciones que más energía consumen, ya que necesitan la activación de la antena WiFi. La explicación de cada función es la siguiente:

pinMode(D6, OUTPUT);
Esta instrucción configura el pin D6 como salida, para así poder activar el voltaje de salida en este pin, que está conectado a la válvula.

configureDTHSensor();
readDataFromSensors();
Estas dos funciones se encargan de configurar los sensores y adquirir los datos de lectura. DTH necesita ser configurado previamente con el uso de la librería <*DHT.h*>.

El sensor de luz no necesita configuración alguna, ya que ofrece una salida analógica que funciona en todo momento.

connectToWiFi();
Esta función conecta el dispositivo NodeMCU a la red WiFi configurada previamente.

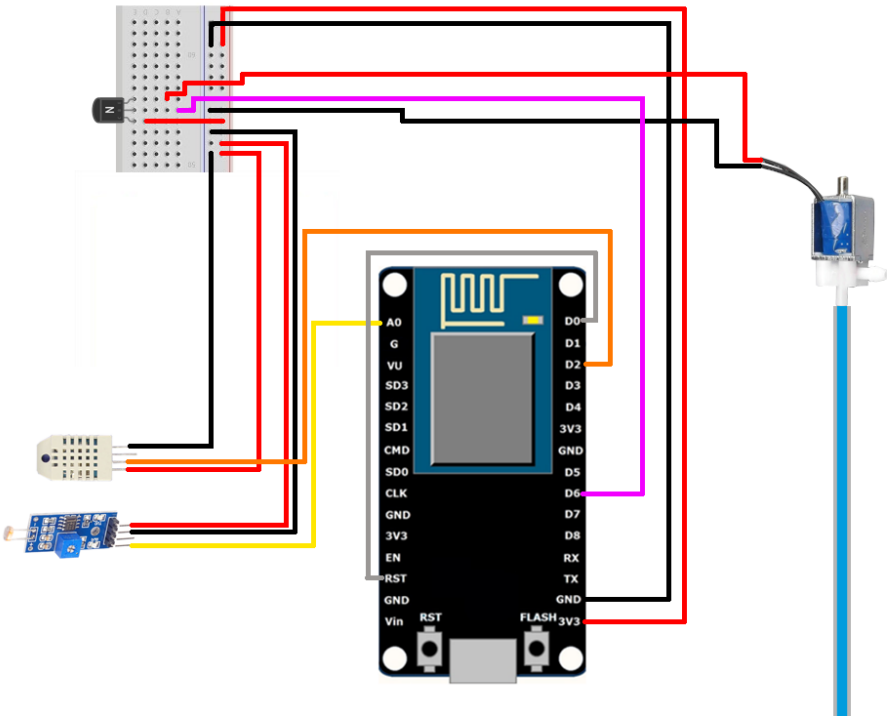
sendSensorDataToServer();
Esta función envía las muestras al backend para su almacenamiento mediante un POST HTTP.

needToIrrigate();
Esta función efectúa un GET al backend para comprobar si la planta ha de ser regada.

irrigateIfNeeded();
Esta función procesa el riego de la planta si este es necesario. El riego se activa hasta que la planta alcance el nivel de humedad necesaria.

ESP.deepSleep(3600e6);
Esta función desactiva la placa y lo deja en modo *deep-sleep* durante 1 hora.

Un problema encontrado en la implementación del sistema de riego fue la apertura de la válvula. El voltaje de salida del pin D6 (y de los pines en general) ha resultado ser menor de 3,3 V, el voltaje con el cual funciona la placa. Debido a esto, aún configurando el pin D6 en salida para poder abrir la válvula, el voltaje ha sido insuficiente.
Para solucionar este asunto se ha utilizado un transistor BC33. El nuevo esquema del sistema de control y riego es el siguiente:



Como se puede apreciar, el colector del transistor se ha conectado a la fuente de 3,3 V de la placa, el emisor a la válvula y la base al pin D6. Mediante esta configuración, cuando se activa el pin D6 permite el paso de corriente entre la fuente de la placa NodeMCU y la válvula.

El presupuesto del sistema de control y riego desarrollado es el siguiente (todas las compras se han realizado mediante la aplicación AliExpress):

COMPONENTE	COSTE
NodeMcu v3 Lua WIFI	€2,07
Sensor DHT-11	€2,03
Sensor LM393	€0,30
Válvula con solenoide	€1,98
Transistor BC33	€0,2
TOTAL	**€6,58**

El presupuesto conseguido es muy reducido, con lo cual satisface los objetivos del sistema.

Prueba de implementación de la red IoT con ContikiOS

Gracias al simulador de Contiki, Cooja, se puede hacer una implementación de la red IoT sin necesidad de implementarla en dispositivos reales.

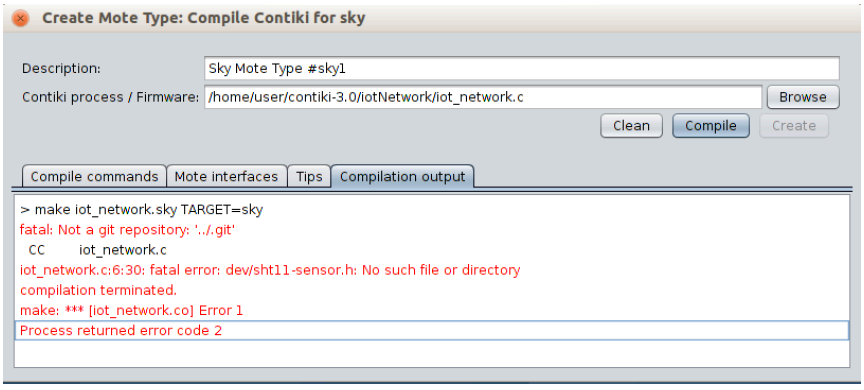
Las muestras tienen que ser enviadas después de cada medida para ser almacenadas en el servicio backend. Como la API que el backend ofrece es únicamente accesible mediante el protocolo HTTP, se utilizará la estructura que utiliza un coordinador para recoger las muestras y enviarlas al backend. Este coordinador debe realizar este papel de forma transparente al backend y a los dispositivos.
En la versión 3 de Contiki se introdujo el soporte a HTTP, con lo cual la implementación de esta estructura con ContikiOS es posible.

Con el diseño propuesto se implementará una red CoaP, donde el coordinador estará conectado a la corriente de la casa y estará siempre encendido, mientras que los nodos que realicen la medida de las plantas se conectarán a él mediante red CoaP, y tras enviar los datos y comprobar si la planta necesita ser regada, entrarán en modo *sleep* para ahorrar energía.

La implementación de esta red en el simulador Cooja se explicará paso por paso, exponiendo los problemas encontrados. Característica por característica, se explicará su implementación:

- **Adquisición de muestras de temperatura, luz y humedad**
  Para elegir un dispositivo con el cual simular el comportamiento de la red, se ha de elegir uno compatible con estas necesidades, que tenga los sensores adecuados. En la primera prueba, se usó la plataforma Tmote Sky, ya que casi todos las guías de Contiki utilizan esta plataforma.
  En la ruta *contiki/platform/sky/dev* se puede encontrar la implementación de estos sensores. Los necesarios serán:
  sht11-arch.h
  light-sensor.c

  Por alguna razón, el sensor *sht11* no compila en Contiki 3.0, mostrando el siguiente error:



El error indica que el archivo *sht11-sensor.h* no se encuentra en la carpeta, cuando sí que se encuentra. Se ha determinado que el error viene dado por cambios en la implementación de Cooja en la versión 3 de Contiki.

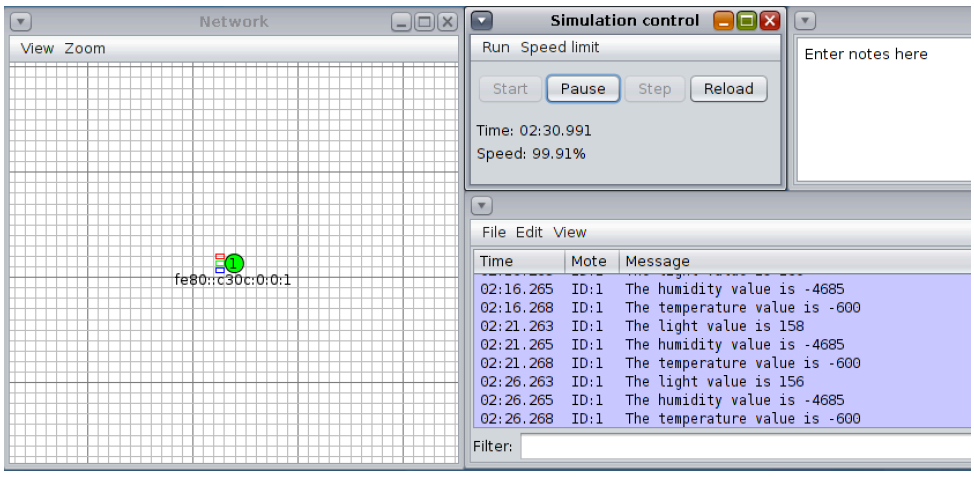
En Contiki 2.7 la compilación funciona sin ningún problema, a pesar de que no hay cambios en el archivo *sht11-sensor.h* en ambas versiones.
En el desarrollo de la aplicación se necesita usar la versión 3.0 de Contiki, así que se necesita realizar la prueba con otro dispositivo.
Con la plataforma Zolertia Z1 se puede comprobar que tiene los sensores necesarios en *contiki/platform/sky/z1*:
light-sensor.h
sht25.h

Con esta plataforma, el programa funciona perfectamente. Para activar y tomar medidas con los sensores se necesita la siguiente pieza de código:

```
//En la cabecera del programa
#include “dev/sht25.h"

//Dentro del programa
SENSORS_ACTIVATE(sht25);
value = sht25.value(SHT25_VAL_HUM);
printf("The temperature value is %d\n", value);
SENSORS_DEACTIVATE(sht25);
```

En esta pieza de código se puede apreciar como se activa el sensor *sht25*, se realiza la medición de la humedad, se imprime por pantalla y se vuelve a desactivar el sensor.
Este es el resultado al medir humedad, luz y temperatura en Cooja:



Como se aprecia en la imagen, al ser un dispositivo simulado, los valores que aparecen son aleatorios.

Con este paso se demuestra que el primer requisito del sistema (medir la temperatura, humedad y luz) se puede implementar en ContikiOS.

- **Servidor CoaP actuando como enrutador de borde**
  El dispositivo que hará la función de coordinador necesitará recolectar las muestras que le envían los demás nodos y enviarlas mediante HTTP al servicio backend.
  En la actualización de Contiki 3, se dio soporte a las principales operaciones HTTP (GET, POST, PUT y DELETE). El código necesario para realizar una petición HTTP es el siguiente:

```
//Declaración de variables en la cabecera del programa
//Direcciones uIP versión 4 y 6
uip_ip4addr_t ip4addr;
uip_ip6addr_t ip6addr;

//Socket HTTP
static struct http_socket s;
```

```
//Código dentro del proceso
  uip_ipaddr(&ip4addr, 8,8,8,8); //Se crea una dirección uIPv4
  ip64_addr_4to6(&ip4addr, &ip6addr); // Mediante el método NAT64
se convierte la dirección uIPv4 a una dirección uIPv6
  uip_nameserver_update(&ip6addr,
UIP_NAMESERVER_INFINITE_LIFETIME);
//Se establece el tiempo de vida de la dirección uIP
  http_socket_init(&s); //Se inicia el socket HTTP
  http_socket_post(&s, "https://a-garden.herokuapp.com/plants",
0, 0,
```

El método NAT64 utilizado para cambiar de IPv4 a IPv6 es un mecanismo de transmisión que facilita la comunicación entre IPv6 y IPv4 usando un formulario de conversión de direcciones de red. Este método esta implementado en Contiki para funcionar con la versión ligera de IP, uIP.
Con el código presentado anteriormente se consigue implementar peticiones HTTP en Contiki, demostrando así que Contiki OS es también compatible con el requisito de envío de muestras al backend.

Para crear el servidor CoaP Contiki provee la librería Erbium. En la carpeta donde se aloje el servidor, se necesita crear una carpeta con los recursos CoaP del servidor y después, vincularlos con el propio servidor.
Para crear el servidor se necesita la siguiente pieza de código:

```
// Inicializar el mecanismo REST del servidor CoaP
 rest_init_engine();
// Unir las los recursos del servidor con las rutas del servidor CoaP
rest_activate_resource(&res_sensor, “post/sensor”);
```

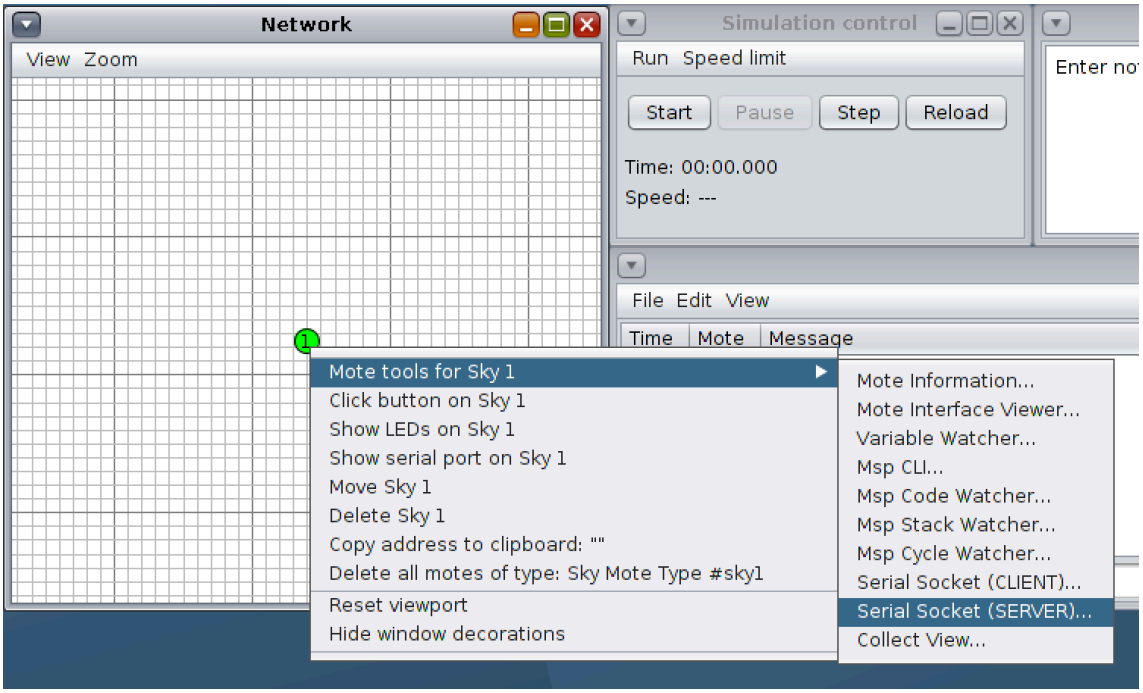
En la carpeta con los recursos, solo necesitamos tomar la muestra enviada por el dispositivo y usar el código http para enviarlo al servicio backend.

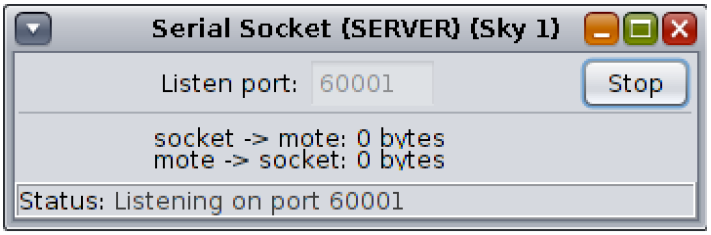
- **Cliente CoaP**
  Después de crear el servidor CoaP, solo se necesita comunicar los dispositivos con el coordinador mediante operaciones POST al servidor CoaP.
  La librería Erbium también ofrece soporte a este tipo de peticiones:

```
//Se inicia el mensaje, indicando la id del coordinador en la red (0), con el contenido del mensaje (request) y el tipo de mensaje (POST).
coap_init_message(request, COAP_TYPE_CON, COAP_POST, 0);
//Se introduce la ruta destino del mensaje
coap_set_header_uri_path(request, “post/sensor”);
//Se calcula el tamaño del mensaje para su envío
const char msg[] = get_measures();
coap_set_payload(request, (uint8_t *)msg, sizeof(msg) - 1);
//Se realiza el envío de forma bloqueante, es decir, hasta que no se transmita el mensaje el dispositivo no continuará ejecutando otras instrucciones
COAP_BLOCKING_REQUEST(&server_ipaddr, REMOTE_PORT, request,
                      client_chunk_handler);
```

Mediante el uso del código del servidor CoaP y del cliente CoaP, se puede realizar la comunicación entre el coordinador y los dispositivos encargados de recoger las muestras.
Este era el último requisito para implementar la red diseñada con ContikiOS, con lo cual se demuestra que la implementación de la aplicación Android y el backend son transparentes a la red IoT y esta es intercambiable. También se demuestra que se puede realizar la implementación de la estructura diseñada para ContikiOS.

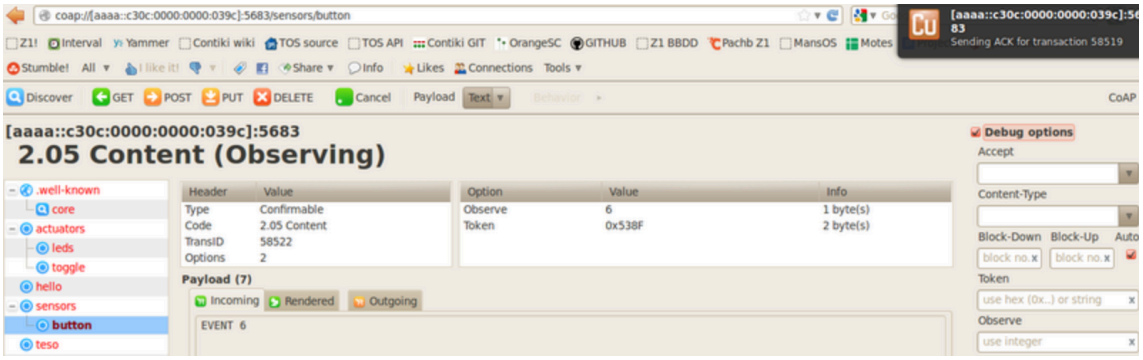
Una vez finalizada la implementación, se puede ejecutar el programa en el simulador de ContikiOS. Para que esta simulación funcione, se debe conectar el nodo coordinador a la red fuera del simulador Cooja, para así poder comunicarse con el servicio backend. Para realizar este proceso Contiki provee la herramienta *tunslip6*. Esta herramienta crea un puente del tráfico IP entre el host y el coordinador de la red que ejecuta las peticiones HTTP. *tunslip6* crea una interfaz de red virtual (tun) en el lado del host y usa SLIP (Linea serial IP) para encapsular y pasar el tráfico IP al otro lado de la linea serial.
Para conectar Cooja a tunslip, primero se debe seleccionar el coordinador y empezar a escuchar en un puerto (un extremo de la linea serial):





Después, se debe conectar el simulador con el exterior para cerrar la linea serial IP. Esto se realiza mediante este comando en una terminal:

Tras este proceso, Cooja puede comunicarse con el exterior del simulador. Sin embargo, al realizar pruebas con el backend, se ha comprobado que la librería no funciona. El mecanismo de resolución de nombres de máquina (DNS) no consigue resolver ninguna URL que se encuentra fuera del simulador Contiki. El servidor CoaP sí es visible fuera del simulador, cómo se ha comprobado usando la extensión Cooper (Cu) en el navegador Firefox:

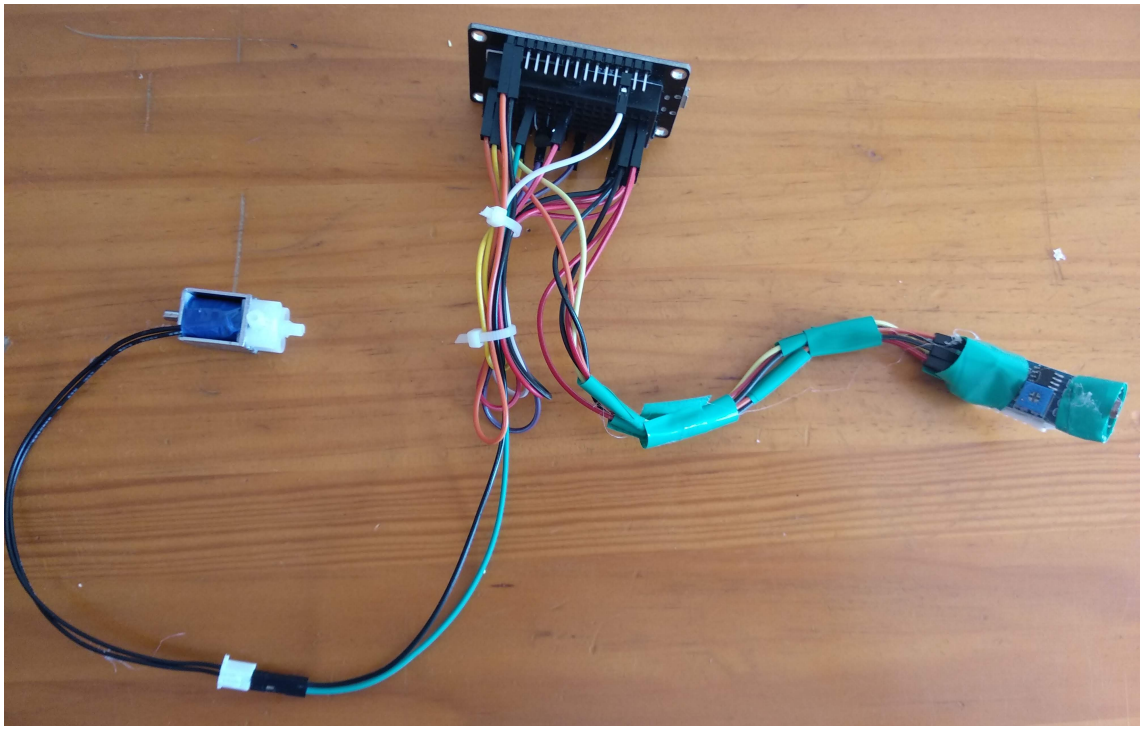


Esta extensión permite realizar peticiones mediante el protocolo CoaP. Sin embargo, su instalación es molesta, ya que no se soporta desde la versión 56 de Firefox, lo que obliga a instalar una versión obsoleta del navegador.

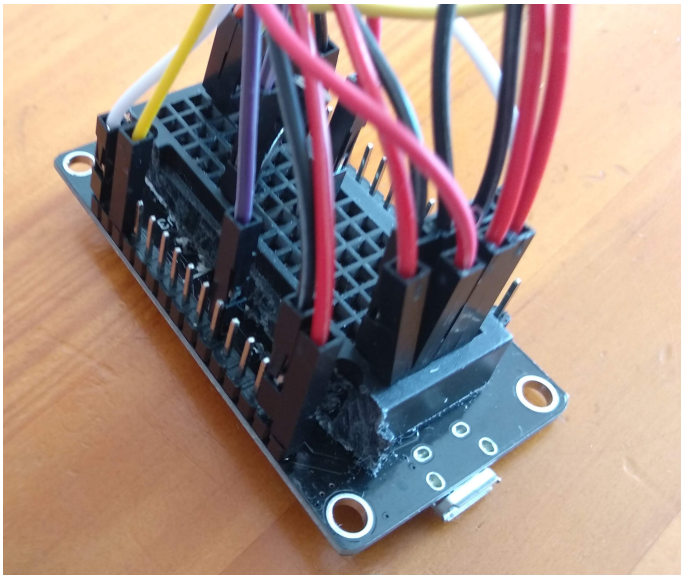
No se ha realizado la prueba del funcionamiento del sistema fuera del simulador.
Tras una búsqueda de soluciones para el problema del DNS se ha determinado que el sistema Contiki no es adecuado para la implementación del sistema, ya que esta búsqueda no ha dado resultados y las soluciones de problemas de la versión 3 de Contiki es prácticamente inexistente en la web.
ContikiNG aún no es compatible con la comunicación de Cooja con el exterior, pero está planificada la inclusión de esta característica en el futuro.
Con los datos obtenidos tras la implementación del sistema en ContikiOS y el análisis de la evolución de Contiki-NG se estima que en un rango de 3 a 5 años Contiki NG será lo suficientemente estable y tendrá las características necesarias para la implantación de este sistema. Sin embargo, actualmente la solución con la plataforma NodeMCU ha resultado mucha más sencilla y fiable, con lo cual será la que será implantada.

Capítulo 6: Implantación

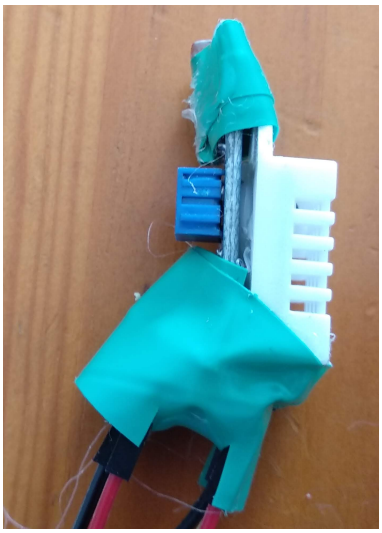
Tal como se ha explicado antes, se ha realizado la implementación del sistema con la plataforma de desarrollo NodeMCU.



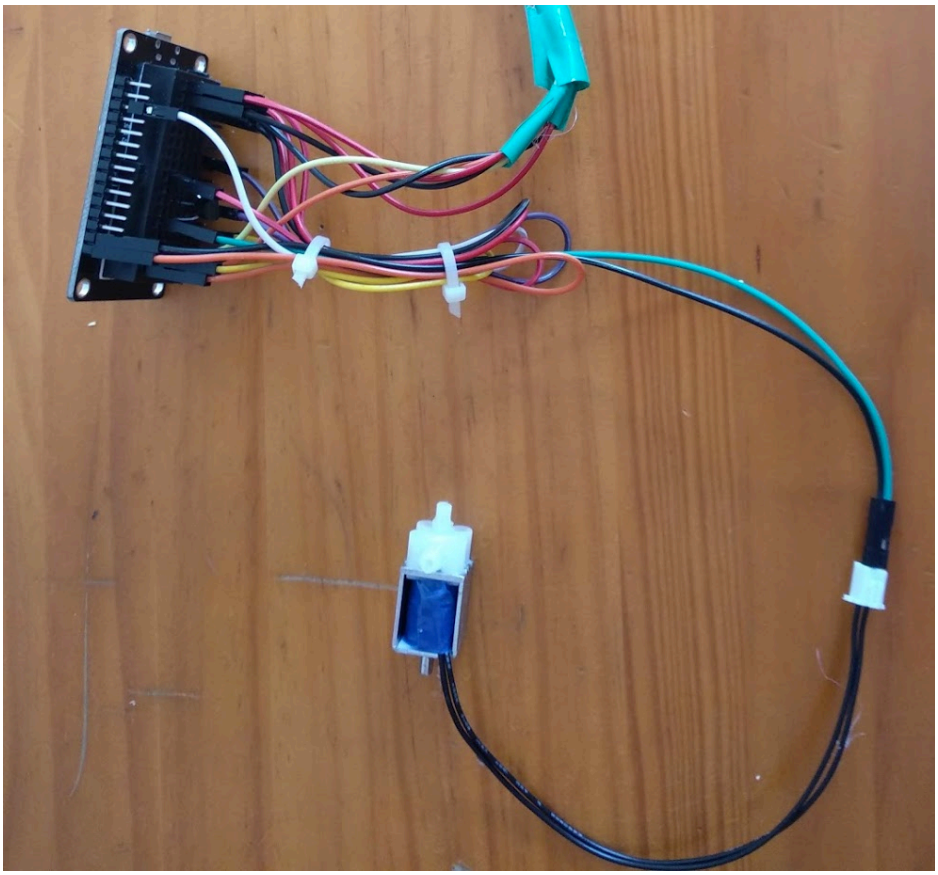
Para conectar todos los cables y que el diseño no abulte mucho se ha pegado una protoboard (placa de pruebas) a la parte trasera de la placa:



Los sensores han sido pegados y fijados en una sola pieza para que se puedan colocar en un extremo de la maceta:



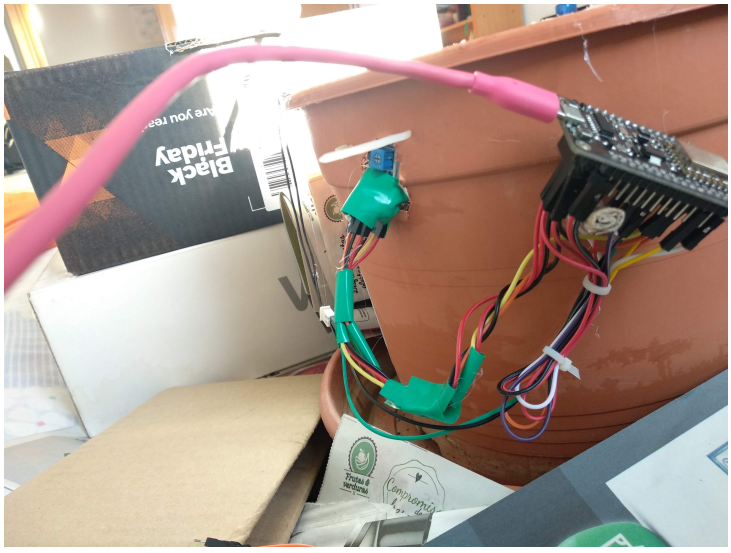
La salida del transistor conectado al pin D6 y una conexión a tierra se han conectado a la válvula:



Capítulo 7: Pruebas

Las pruebas del sistema se han realizado empíricamente. Se ha realizado el montaje del sistema implantado en una maceta. Como deposito de agua se ha utilizado una botella de agua de 1,5 L y para proveer de energía al dispositivo se ha conectado a un cargador de móvil.
El sistema se puede ver en estas imágenes:

Visión general del sistema.
La botella ha sido conectada mediante un tubo a la válvula, la cual ha sido fijada en la maceta.
Del otro extremo de la válvula sale otro tubo que reparte el agua por la maceta.



Dispositivo NodeMCU.
La placa ha sido conectada a un cargador de un móvil (cable rosa).

Sensores.
Se ha realizado un agujero en la maceta para colocar los sensores. Este agujero se podría fijar para evitar que se cayeran los sensores.

La colocación de los sensores próxima al tubo de riego es causada para la correcta medición de la humedad ganada con el riego. En las pruebas se ha apreciado que cuando el sensor de humedad está lejos de la fuente de agua, el riego es excesivo. Esta estructura ha resultado ser la más efectiva para controlar el riego.

Sistema en el momento de riego

Tras las pruebas se reconoce que el funcionamiento es correcto, los distintos modos de riego funcionan tal y como han sido configurados. El sistema podría ser perfeccionado si se tuviese el conocimiento exacto de las formas de riego que las plantas necesitan, pero es un campo totalmente nuevo que no se ha investigado dado que no era necesario para los requisitos del sistema.

Capítulo 8: Conclusiones

Tras la realización de este proyecto ha sido posible comprobar que los medios actuales permiten realizar sistemas IoT útiles y a un precio bajo. El coste de producción del sistema producido ha resultado mucho menor que las opciones disponibles en el mercado y su funcionalidad es aceptable, se puede adaptar a cualquier tamaño de maceta, se puede alimentar con diferentes fuentes de energía y es independiente de la aplicación Android.

El análisis del funcionamiento de la red IoT, sus protocolos y sus sistemas operativos ha permitido la comprensión de la magnitud y la complejidad de esta red. Respecto a los sistemas operativos, ha sido realmente edificante el poder ver cómo un grupo de desarrolladores ha revitalizado el desarrollo de Contiki con ContikiNG, además de tener la oportunidad de hablar con ellos directamente en el canal de Contiki.
Ha sido necesario el aprendizaje del funcionamiento y programación de este sistema operativo. Esta parte ha sido con diferencia la parte más difícil y costosa de todo el proyecto, ya que no había experiencia alguna en el sector. Los conocimientos iniciales antes de abarcar el proyecto eran únicamente de Android y Kotlin, con lo cual toda la parte realizada con la red IoT ha sido implementada tras el aprendizaje de todo el funcionamiento.
Esta falta de experiencia en el sector fue un gran error al realizar el proyecto, ya que consumió una gran cantidad de tiempo en un sistema operativo que no era todavía estable. Además, la curva de aprendizaje era tan alta que era frustrante. Este error podría haber sido evitado reservando más tiempo para la implementación decidida en el capítulo de diseño de la solución. Una mejor gestión del proceso podría haber aportado mucho al desarrollo del proyecto.
A pesar de todo, una vez aprendidas las bases de Contiki es posible darse cuenta del gran futuro que tiene este sistema en el IoT.
Respecto a la implementación con la placa de desarrollo NodeMCU, ha sido necesario aprender el funcionamiento de esta placa y de los sensores a los que se ha conectado. Esta parte ha sido mucho más liviana que el aprendizaje de Contiki, gracias a la gran cantidad de documentación y la sencillez con la que la placa debe ser programada.

Tras este proyecto se ha comprendido de forma más profunda el funcionamiento de las redes en general, se ha aprendido el montaje de plataformas de desarrollo con sensores y actuadores y se han mejorado los conocimientos en Android y Kotlin.
También es notable la satisfacción recibida al conseguir la implementación de un sistema listo para utilizar en los hogares a un precio realmente asequible.
Los objetivos del proyecto han sido conseguidos, aunque a un grado menor del deseado. En el capítulo de trabajos futuros se analizarán las cosas que se habrían querido también conseguir en el proyecto.

Relación del trabajo desarrollado con los estudios cursados

A nivel tecnológico, el desarrollo de la aplicación Android y del backend han sido basados en los conocimientos que se han aprendido en la carrera. Además, la asignatura Internet of Things cursada en el periodo de estancia en el Politécnico de Milano ha ayudado a conocer los sistemas operativos analizados y las cuestiones principales del Internet of Things, aunque no se haya profundizado al nivel que se ha hecho en el proyecto.
Se han utilizado también muchas herramientas que no se han visto en el desarrollo del proyecto y que fueron aprendidas a lo largo de la carrera, como por ejemplo el control de versiones (se ha utilizado git y la plataforma GitHub) o el uso de Docker (necesario para la instalación de Contiki-NG)
El trabajo ha realizado una gran cantidad de tecnologías, siendo conocidas en la parte de la aplicación Android y el backend. En cuanto a las del sistema de control y riego, todas han sido descubiertas en el desarrollo del proyecto.
En mi opinión, en el trabajo se han querido abarcar demasiadas tecnologías, por lo cual algunas han quedado con una explicación demasiado escueta.

Respecto a las competencias transversales que han resultado útiles para la elaboración del proyecto cabe destacar las siguientes:

- **Innovación, creatividad y emprendimiento / Conocimiento de problemas contemporáneos**
  La idea del sistema de riego y control de plantas viene motivada por el deseo de conseguir un sistema que pueda ayudar en los hogares y que permita ser asequible a todas las personas.

- **Aprendizaje permanente**
  El aprendizaje de conceptos como el control de versiones, las arquitecturas de software y simulación de sistemas operativos con VirtualBox ha sido muy útil, ya que estos han sido utilizados durante el desarrollo del proyecto y no ha sido necesario volver a estudiar para su uso.

Capítulo 9: Trabajos futuros

Aunque se han conseguido los objetivos planteados inicialmente, han quedado varios flecos que no han sido abordados por falta de tiempo:

- Sistema para añadir plantas y actualizaciones de firmware
  En el proyecto no se ha podido abordar el tema de cómo se puede añadir una planta a la lista una vez el sistema está implementado o cómo se puede actualizar en firmware del sistema. Este problema se podría solucionar mediante un programa para ordenador que pueda acceder al dispositivo mediante una conexión USB. De esta forma, se podría añadir la planta al sistema y actualizar el dispositivo.
  Otra opción que se podría valorar sería el uso de NFC o Bluetooth para añadir la planta al sistema
- Implementación del sistema funcional con el simulador Cooja
  Los errores y la poca ayuda que se encontraba en la web para el desarrollo del sistema en Cooja han hecho imposible que el sistema funcionara en el simulador. En los próximos años con las mejoras que va incluyendo Contiki-NG esta opción podría ser completada.

También hay muchas funcionalidades interesantes que podría tener el sistema de control y riego. En mi opinión, este sistema podría tener potencial para ser un producto que se pueda vender al público. Algunas de las funcionalidades a añadir pensadas para la venta del producto serían:

- Diseño para acoplar el dispositivo a las macetas mediante algún tipo de enganche
- Cuidado más intensivo de las plantas, clasificación de plantas por tipo y requerimientos de sol. La aplicación podría notificar al usuario cuando la planta está recibiendo demasiada o insuficiente luz. También podría incorporarse un rango de temperatura adecuada para la planta.
- Disminución máxima del consumo de la placa. Es posible modificar manualmente la placa para reducir el gasto de energía, quitando componentes no utilizados. [26]

Además de su bajo precio, la gran ventaja que este producto presenta es su adaptabilidad. En vez de crear macetas propias, es más útil el desarrollo de este producto como una pieza adaptable que añadir a las macetas ya existentes. Esto permite mantener un coste bajo de producción y permite ganar comodidad al usuario, ya que no debe mover una planta de una maceta a otra.

Capítulo 10: Referencias

Bibliografía

Libros

Practical Contiki-NG: Programming for Wireless Sensor Networks (disponible en PoliBuscador)
Cisco self-study : implementing IPv6 networks (IPV6) (disponible en PoliBuscador)
Learning Internet of Things | Waher, Peter (disponible en PoliBuscador)
IoT fundamentals: networking technologies, protocols, and use cases for the Internet of Things (disponible en PoliBuscador)
TinyOS Programming (http://csl.stanford.edu/~pal/pubs/tos-programming-web.pdf)
Manual de simulación de Cooja (https://www.researchgate.net/publication/304572240_Cooja_Simulator_Manual)

Fuentes de información online

Página oficial de Contiki:
http://www.contiki-os.org/

Página oficial de Contiki OS en GitHub:
https://github.com/contiki-os/contiki/wiki/

Información de Adam Dunkels sobre el funcionamiento de Contiki:
http://dunkels.com/adam/pt/
http://www.dunkels.com/adam/download/uip-doc-0.5.pdf

Información sobre los protocolos de Internet of Things
https://www.postscapes.com/internet-of-things-protocols/
https://www.link-labs.com/blog/zigbee-vs-wifi-802-11ah
https://iot6.eu/ipv6_for_iot
https://tools.ietf.org/html/rfc6550
https://www.ibm.com/developerworks/library/iot-lp101-connectivity-network-protocols/index.html
https://www.researchgate.net/publication/303192188_A_survey_on_application_layer_protocols_for_the_Internet_of_Things

Página oficial de TinyOS
http://tinyos.stanford.edu/tinyos-wiki/index.php/Main_Page

Datasheet DHT22
https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf

Datasheet DHT11
https://www.mouser.com/ds/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf

Referencias en el texto

[1] https://www.amazon.es/
[2] https://es.aliexpress.com/
[3] https://www.parrot.com/global/connected-garden/parrot-pot#parrot-pot
[4] http://www.fliwer.com/
[5] https://tuxiaomi.es/tienda-xiaomi/xiaomi-mi-plant-monitor/
[6] http://www.ti.com/product/CC2538/samplebuy
[7] http://www.ti.com/ww/en/wireless_connectivity/sensortag/devPacks.html
[8] https://store.ti.com/CC2650F128RGZR.aspx
[9] https://github.com/PIlin/contiki-arduino
[10] https://sourceforge.net/p/contiki/mailman/message/29876684/
[11] http://www.eistec.se/mulle/ordering/
[12] https://www.advanticsys.com/shop/mtmcm3300msp-p-7.html
[13] https://sourceforge.net/projects/libcoap/
[14] https://www.amazon.com/TinyOS-Programming-Philip-Levis/dp/0521896061
[15] https://www.wemos.cc/
[16] https://nodemcu.readthedocs.io/
[17] https://nodemcu.readthedocs.io/en/master/en/modules/coap/
[18] https://www.losant.com/blog/making-the-esp8266-low-powered-with-deep-sleep
[19] https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf
https://www.mouser.com/ds/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf
[20] http://www.home-automation-community.com/temperature-and-humidity-from-am2302-dht22-sensor-displayed-as-chart/

[21] https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html
[22] https://github.com/kittinunf/Fuel
[23] https://github.com/PhilJay/MPAndroidChart
[24] http://ktor.io/
[25] https://sunrise-sunset.org/api
[26] https://tinker.yeoman.com.au/2016/05/29/running-nodemcu-on-a-battery-esp8266-low-power-consumption-revisited/