Departamento de computación
Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑERÍA INFORMÁTICA
MENCIÓN EN TECNOLOXÍAS DA INFORMACIÓN

Arquitectura distribuída para a xeración dun DATASET semisintético de tráfico IoT

Estudante: Ares Montes Rey
Director/a/es/as: Víctor Manuel Carneiro Díaz

A Coruña, 4 de setembro de 2019.

Dedicado á miña familia, que estivo en todo momento pendente do progreso deste proxecto.

Agradecementos

Agradezo ó meu amigo Santiago por ser como é e a Stéfano e a Fernando polas risas ó asociar Contiki cun tipo de bebida tropical.

Resumo

O obxectivo principal deste traballo fin de grado consiste na xeración semi-sintética dun DATASET de tráfico IoT procedente dos sensores de temperatura dun Centro de Proceso de Datos (CPD). Para isto, lévase a cabo un estudo do estado do arte en materia dos diferentes tipos de sistemas operativos para IoT, simuladores dos mesmos, sensores dispoñibles e dos protocolos de comunicacións que se usarán. Tamén impleméntase en C o código a executar dos sensores para que reporten cada certo tempo unha temperatura o máis realista posible, mediante a modelización matemática do comportamento da mesma, e implémentase unha rede para que os sensores en varias instancias do simulador se poidan executar en paralelo e se comuniquen cun broker onde publicar os datos. Finalmente, execútase a simulación e realízase a captura das transmisións. Así, o DATASET resultante, é unha captura con datos de temperaturas que poderá ser usado en posteriores traballos para o estudo e investigación sobre ataques e vulnerabilidades ós protocolos de transmisión dos sensores.

Abstract

The main objective of this Final Degree Project is the semi-synthetic generation of a DATASET of IoT traffic from a datacenter's temperature motes. To achieve this, a state of the art study is carried out on different types of operating systems for IoT, IoT simulators and available motes and communication protocols used for this matter. Also, the code that the motes execute for temperature reporting it's implemented in C using mathematically modeled functions for realistic results. The network is implemented in a way that allow motes in multiple instances of the simulator to run in parallel and communicate with a broker where they publish their data. Finally, the simulation is run and the transmission is captured. This way, the resulting DATASET, it's a temperature data capture that can be used in subsequent work to study and research in matter of attacks and vulnerabilities of sensor transmission protocols.

Palabras chave:

- Cooja
- Contiki
- IoT
- Redes de sensores sen fíos
- Dataset
- Simulación de redes

Keywords:

- Cooja
- Contiki
- IoT
- Wireless sensor networks
- Dataset
- Network simulation

Índice Xeral

Índice de Figuras

Índice de Táboas

Capítulo 1

Introdución

1.1 Contexto

A finalidade deste proxecto é a creación semi-sintética dun dataset con temperaturas similares ás que tería un Centro de Procesamento de Datos (a partir de agora, CPD) real mediante un simulador de redes de Internet of Things (a partir de agora, IoT). Para isto, é necesario crear unha simulación con todos os sensores necesarios e unha rede que os conecte cun software para recibir os datos. Tamén é necesario preparar dita rede para poder capturar nela os paquetes transmitidos para xerar o dataset final.

Este dataset pode ser utilizado por parte doutros proxectos e, de feito, é unha das motivacións principais deste traballo, xa que será utilizado para adestrar unha intelixencia artificial que permita detectar ataques aos protocolos de comunicacións dos sensores dun CPD. Debido a que o produto final do traballo será utilizado noutro proxecto de investigación, este ten xa algúns requisitos establecidos, como o tipo de software de comunicacións utilizado ou a cantidade de sensores a simular.

Este traballo pode ser usado como base para a creación doutros proxectos similares, onde se necesite capturar o tráfico dunha gran cantidade de sensores que se estean a simular ou emular, xa que ten unha natureza mais similar á de proba de concepto que á dun proxecto que pretenda crear un produto final.

1.2 Obxectivos

Nesta sección establécense cales son os obxectivos principais do traballo:

- **Analizar os diferentes simuladores e sistemas operativos para sensores existentes**: Realízase un estudo sobre os principais sistemas operativos (a partir de agora, SO) para IoT, onde se analizan as súas vantaxes e desvantaxes, así como un estudo sobre

os simuladores de redes de sensores (a partir de agora, WSN polas súas siglas en inglés Wireless Sensor Networks), analizando as diferenzas entre eles.

- **Deseñar a arquitectura dunha simulación distribuída**: Debido ás características da simulación, será necesario o uso dunha arquitectura onde varias instancias do simulador se executen en paralelo e permitan que os sensores simulados se podan comunicar entre si e cun elemento software externo a elas, permitindo capturar todo o tráfico que se dirixa de e a este.

- **Escribir o código dos sensores de temperaturas**: Os sensores simulados executan un software que recolle a temperatura e a transmite por rede. Débese desenvolver este comportamento con axuda do propio sistema operativo e de librarías.

- **Executar a simulación e capturar o tráfico**: O obxectivo principal do proxecto é obter o dataset de temperaturas, polo que executar a simulación e capturar o tráfico da mesma para obtelo é primordial.

- **Analizar a calidade do dataset resultante**: Tras capturar o tráfico da simulación e obter o dataset de temperaturas, débese analizar a calidade e as características do mesmo.

1.3 Estrutura da memoria

Esta memoria contén unha serie de capítulos co traballo realizado durante o desenvolvemento deste proxecto:

1. No primeiro capítulo, o de **introdución**, trátase o motivo do traballo e os seus obxectivos.

2. No seguinte capítulo, o do **estudo do estado do arte**, trátanse os diferentes produtos que existen para resolver o problema e se estudan as súas diferenzas.

3. No terceiro capítulo trátase a **metodoloxía** usada, no cal se explica o motivo da mesma, as iteracións realizadas e as historias de usuario para cada iteración.

4. No cuarto capítulo realízase un **análise de requisitos** onde se indican as funcionalidades e requerimentos do simulador. Tamén se inclúe unha planificación do proxecto.

5. O quinto capítulo é o de **implementación, execución e probas** da simulación, onde se explica cal é o código das motas de forma superficial e como se realiza a captura dos datos. Tamén se indican unha serie de probas a realizar.

6. Para continuar, o sexto capítulo é o de **análise de resultados**, onde se indican diferentes estadísticas e características do dataset resultante.

7. E para finalizar, no sétimo capítulo quedan escritas as **conclusións** do traballo, as potenciais liñas futuras de traballo e as leccións aprendidas durante o mesmo.

A maiores dos capítulos, a memoria inclúe ó final unha serie de apéndices con información adicional, un glosario de acrónimos, un glosario de termos usados e a bibliografía que é citada polas referencias.

Capítulo 2

Estado do arte

Neste capítulo do estudo do estado do arte, analízanse os sistemas operativos para IoT mais usados e os principais simuladores de IoT existentes e estúdase con maior detalle o sistema operativo e o simulador escollido para usar neste proxecto, ademais dos principais tipos de motas e os diferentes protocolos de comunicacións que usan.

2.1 Sistemas operativos existentes

A seguinte selección de sistemas operativos existentes deseñados para ser utilizados en IoT non é exhaustiva [1] e está baseada na selección dos sete mais utilizados segundo unha enquisa realizada por *ITProToday* en 2018 [2].

- **Linux**: Existen multitude de SO que están baseados neste kernel como, por exemplo, Raspbian, o SO mais utilizado segundo a mesma enquisa [2]. É natural que este sexa o SO mais utilizado por ser open-source, moi maduro, por ter un desenvolvemento moi activo e por ser utilizado amplamente fóra do ámbito do IoT. Sen embargo, para a súa execución é necesario un mínimo de 1MB de RAM sendo o mínimo recomendado 8MB [3], algo incompatible cos microcontroladores dos sensores de temperaturas, onde a capacidade de RAM xira entorno as decenas de kilobytes.

- **Windows**: O sistema operativo de Microsoft na súa versión adicada para dispositivos IoT chamada Windows IoT. Aínda que é o único SO que permite programar dispositivos IoT dentro do ecosistema de Windows, é un SO de código pechado, ten un requerimento de RAM mínimo de 256 MB, 8 GB de almacenamento e require dun procesador moito mais rápido do que teñen a maioría de microcontroladores [4]. Este SO só pode executarse en arquitecturas ARM, x86 e x64.

- **FreeRTOS**: É o SO mais usado deseñado para ser executado nun microcontrolador, o que lle permite ser executado nun dispositivo con menos de 10 KB de RAM. Conta cunha

licencia MIT que lle permite ser utilizado ata en proxectos comerciais, e unha gran comunidade de soporte. Ten unha arquitectura de microkernel e está deseñado para ser un SO de execucións de aplicacións con requisitos de tempo real, tamén chamado RTOS, polas súas siglas en inglés de Real-Time Operating System [5]. Amazon mantén o Amazon FreeRTOS, un FreeRTOS con algunhas características extendidas [6].

- **Mbed OS**: É un SO cunha gran conectividade incluíndo NFC, Bluetooth, Wi-Fi, datos celulares, etc, capacidade para executar aplicacións en tempo real, un deseño modular e un deseño orientado á seguridade [7]. Usa un kernel monolítico e usa un paradigma baseado en eventos. O seu código está dispoñible baixo unha licencia Apache 2.0, o que permite usar o SO en proxectos comerciais. Sen embargo, é un SO só executable en plataformas ARM Cortex-M de 32 bits.

- **TinyOS**: Un SO para microcontroladores que pode chegar a requirir tan só 1 Kilobyte de memoria. Ten capacidades para controlar a enerxía incluídas no seu propio kernel, o cal tamén é capaz de realizar concorrencia usando un paradigma baseado en eventos. Este SO está orientado a microcontroladores IoT con recursos extremadamente baixos a costa dunha flexibilidade menor onde para substituir a aplicación que se está a executar, tamén é necesario substituir o propio kernel, mentres que outros SO non o requiren [8]. A linguaxe de programación das aplicacións para este SO é NesC, un dialecto de C ó contrario da maioría dos outros SO que usan C ou C++.

- **RIOT OS**: É un SO deseñado para cubrir o espazo entre os SO das redes de sensores e os SO tradicionais. Cunha arquitectura de microkernel e deseñado para a execución de tarefas en tempo real, RIOT OS ten capacidades similares as de FreeRTOS pero, a diferenza deste, xa conta cun stack de rede e drivers por defecto usando tan só 1.5 KB de memoria RAM [9].

- **Contiki**: Ó igual que FreeRTOS, é un SO deseñado para ser executado nun microcontrolador con poucos kilobytes de memoria, pero a diferenza do mesmo, Contiki ten un kernel monolítico e non ten como primeira prioridade a execución de tarefas en tempo real. Está deseñado para traballar coas redes WSN soportando o estándar IEEE 802.15.4 que define redes locais Low-Rate Wireless Personal Area Network (a partir de agora, LR-WPAN) usadas por dispositivos de baixo custo e baixa velocidade. Contiki tamén conta cun simulador oficial orientado a redes WSN incluído no propio repositorio de SO chamado Cooja [10].

2.2 Simuladores de WSN existentes

Para o desenvolvemento do proxecto é necesario contar cun simulador da rede de sensores de temperaturas. A seguinte lista está escrita en base a unha selección de sete dos simuladores de WSN mais utilizados segundo o artigo *Wireless Sensor Network Simulators: A Survey and Comparisons* [11].

- **Ns-2**: Un simulador de propósito xeral escrito na nunha combinación de C++ e OTcl, unha extensión de Tcl orientada a obxectos. Ten un gran potencial grazas a súa extensibilidade por módulos e, grazas ó seu deseño orientado a obxectos, permite crear novos protocolos facilmente. Ademais, conta cunha interface visual para ver o proceso de simulación. Sen embargo, non ten a capa de aplicación e, polo tanto, non se pode simular software personalizado e complexo nos nodos. Tamén conta cunha curva de aprendizaxe excesivamente ampla, o que pode chegar a ser un problema.

- **TOSSIM**: É un simulador deseñado para motas MICA executando TinyOS. Como simulador de TinyOS, usa nesC como linguaxe de programación. Ten a vantaxe de que a simulación dos protocolos de rede as fai a nivel de bit, o que permite personalizar todos os protocolos de rede. Un problema que ten é que TOSSIM asume que todos os nodos da rede executan o mesmo código facendo que moitos proxectos, como este, non o poidan utilizar pola súa natureza.

- **GloMoSim**: Realiza as simulacións agregando os nodos pola súa posición física na simulación, o que permite mellorar o seu rendemento de forma notable. O simulador é realmente unha serie de librarías escritas en PARSEC, un linguaxe baseado en C. A diferenza de outros simuladores, GloMoSim é capaz de executarse en entornos paralelos de forma nativa. O problema de GloMoSim é que está deseñado exclusivamente para comunicacións de rede de forma sen fíos e baseadas en IPv4.

- **Avrora**: A diferenza de outros simuladores, é independente do sistema operativo e linguaxe usados, aínda que a implementación do simulador en si mesmo é Java. É capaz de realizar simulacións cunha precisión temporal completa sobre redes de ata 10.000 nodos cun rendemento aceptable, sendo, aínda así, o dobre de lento que TOSSIM.

- **SENS**: É un simulador cunha arquitectura modular por capas -entorno físico, rede e aplicación- e permite simulacións realistas ó utilizar datos de sensores reais para modelar o seu comportamento. Está escrito en C++ pero é independente da plataforma a simular. Ten a capacidade de crear entornos como o de son, temperaturas ou sinal de forma personalizada, pero faino de forma non exacta.

- **J-Sim**: Similar a Ns-2, J-Sim é un simulador de propósito xeral escrito en dúas linguaxes: Java e Jacl, unha versión Java de Tcl. Usa unha arquitectura baseada en compoñentes, a cal escala mellor que a arquitectura baseada en obxectos usada por Ns-2, entre outros simuladores. Aínda que non é mais complicado que Ns-2, a súa curva de aprendizaxe segue sendo ampla pero, en comparación con este, ten unha comunidade menor de soporte. Tamén, por estar escrito en Java, ten un rendemento medio menor en comparación cos simuladores escritos en C ou C++.
- **Cooja**: Non é en realidade un simulador, senón que é un emulador do sistema operativo Contiki que permite executar a rede de sensores a nivel de hardware. Cooja está escrito en Java, pero os sensores prográmanse nun subconxunto de C. Permite a execución nunha mesma simulación de hardware e software diferentes e, incluso, permite emular un conxunto dos sensores a nivel hardware e emular outro a un nivel mais alto. O principal problema de Cooja é o seu rendemento, como se indica na sección 5.3.

2.3 A elección: Contiki e Cooja

O sistema operativo finalmente escollido é Contiki, xa que ten capacidades multifío mediante protothreads [12], un stack TCP/IP xa incluído no mesmo e úsase habitualmente para sensores na vida real grazas a súa alta compatibilidade con estándares de Internet, o rápido desenvolvemento nel, a ampla comunidade, a súa madurez e que soporta soporte comercial [13]. Todo isto cunha licencia libre BSD [14] que permite o seu uso en proxectos de todo tipo, incluíndo os comerciais. Ten unha pegada en memoria de tan só uns kilobytes, o que permite a súa implementación nun amplo rango de dispositivos de baixa potencia [15].

Para realizar a simulación escolleuse o emulador Cooja [16], primeiramente por ser o oficial de Contiki, garantindo a máxima compatibilidade de base, e tamén porque evita algúns problemas que teñen outros simuladores tendo unha capa de aplicación onde executar aplicacións e software de comunicacións personalizados ou a capacidade de simular nodos non homoxéneos en hardware e en software. Cooja permite emular [17] as motas a nivel de hardware, o que engade precisión á hora de inspeccionar o sistema e mais realismo aos resultados producidos, ademais de que permite usar o mesmo código que se usaría nun sensor físico real. Isto resulta nunha execución mais lenta que a dun simulador, aínda que Cooja conta cun tipo de sensor chamado *Cooja Mote* que permite emular o sensor a un nivel mais alto, permitindo que o impacto no rendemento do sistema sexa menor.

2.3.1 Cooja

Tras compilar e arrancar Cooja por primeira vez, aparece a interface da figura 2.1, a cal ofrece a opción para crear unha simulación nova ou abrir unha anterior. Se se escollese crear

unha nova simulación, chegaríase a interface da figura 2.2, onde se poden distinguir o seguinte conxunto de ventás:

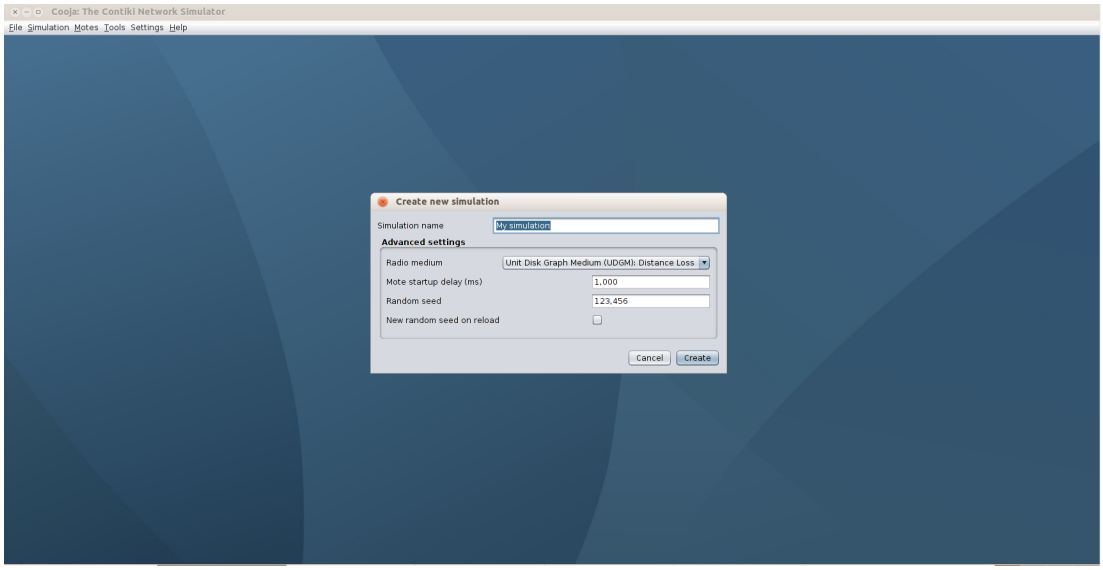


Figura 2.1: Cooja tras arrancar

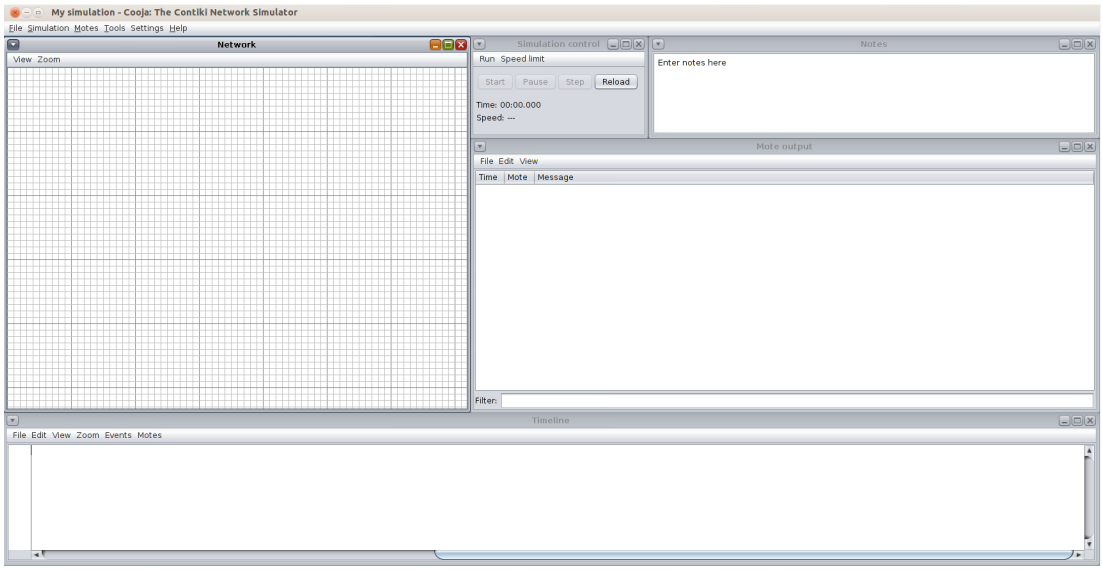


Figura 2.2: Interface principal de Cooja

- **Network**: Na parte esquerda superior está a zona de traballo onde se colocan os nodos no espazo físico simulado. É importante colocar correctamente os nodos, posto que só os que están preto entre si poden comunicarse.
- **Simulation Control**: A seguinte ventá xusto á dereita é o panel de control da simulación. Nel pódese modificar a velocidade de simulación entre os valores de 1%, 10%, 100%, 200%, 1000% e sen restrición, ademais de iniciar, pausar e reiniciar a simulación.
- **Notes**: O panel na parte superior dereita é unha ventá simple onde tomar notas relacionadas coa simulación.
- **Mote output**: Xusto debaixo do panel de notas está o panel de saída das motas. Neste panel aparecerán tódalas mensaxes de forma ordenada por tempo que impriman as motas durante o transcurso da simulación.
- **Timeline**: O panel que está abaixo de todo é o de transmisións. Indica mediante liñas grises se a mota está transmitindo datos e indica se algún dos LEDs da mota está acendido.

Ademais das ventás que ten por defecto, é posible abrir outras dende o menú de *Tools*. Tamén pódese facer clic dereito sobre as motas na ventá de *Network* para obter mais información delas e acceder ó visor de interfaces da mota (ver figura 2.3). Neste visor pódese consultar datos sobre a mota como a súa IP, premer o seu botón, subir arquivos ó seu sistema de ficheiros *Coffe Filesystem*, etc.

A execución dunha simulación en Cooja resulta nunha pantalla similar á da figura 2.4, onde se pode observar que o panel de *Network* contén liñas vermellas indicando cada unha das conexións que estableceron as motas e, no centro da figura, unha ventá extra chamada *Serial Socket*. Este panel utilízase para configurar o porto TCP polo que conectarse ó porto serial da mota e, desta forma, ter comunicación directa coa mesma dende fóra do simulador.

2.3.2 Tipos de motas

De entre as motas [18] que Cooja ten dispoñibles para emular, as mais comúns [19] son os microcontraladores baseados no Texas Instruments MSP430, os cales son emuladas co emulador MSPSim integrado dentro de Cooja e constan da familia Zolertia Z1 e TelosB/SkyMote.

- **Sky Mote**: A plataforma usada por defecto en Contiki para realizar emulacións. Representa a un hardware con capacidades de rede IEEE 802.15.4 de 2.4GHz a 250kbps, un procesador MSP430 a 8 MHz, 10 kilobytes de RAM e 48 kilobytes de memoria flash. A súa antena pode alcanzar ata a 125 metros en exteriores e ten integrado os sensores de temperatura, humidade e luminosidade [20].

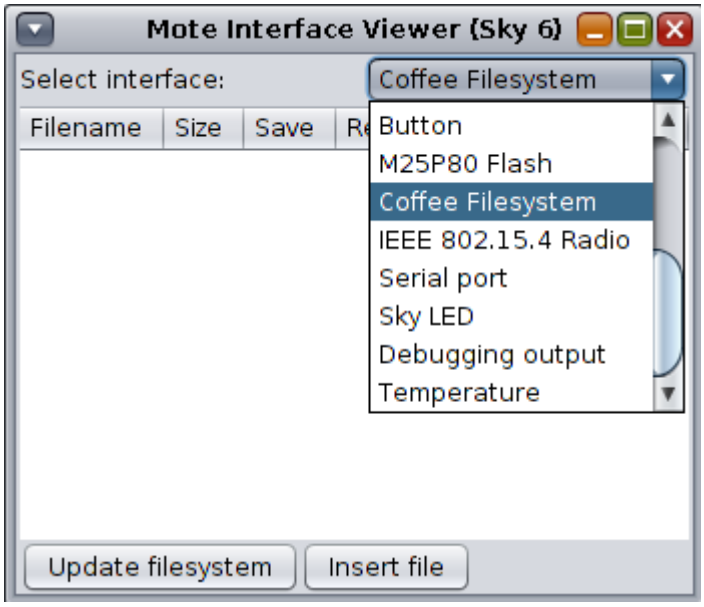


Figura 2.3: Visor de interfaces de Cooja

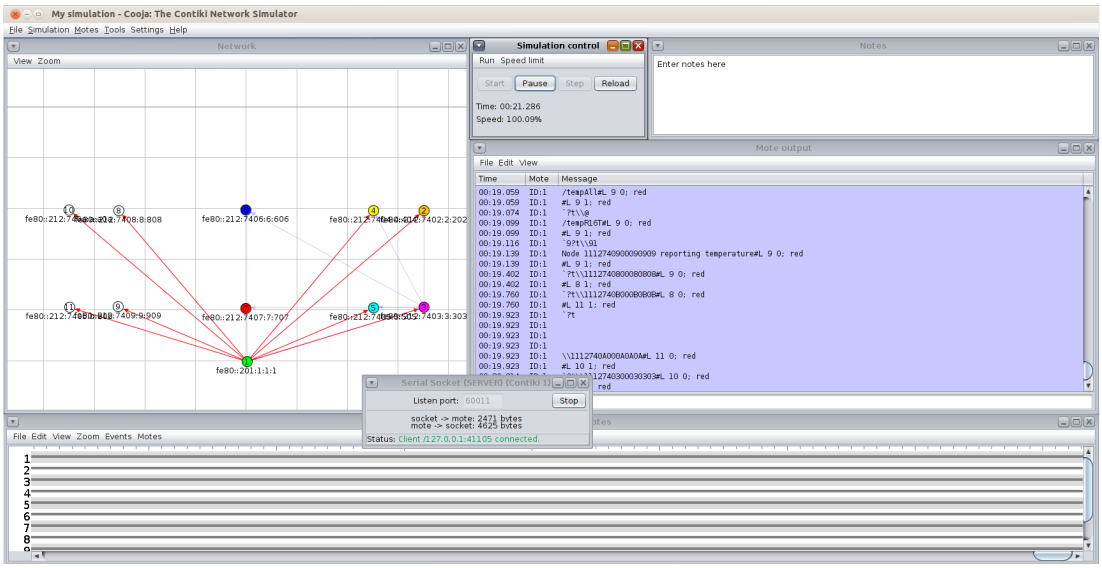


Figura 2.4: Cooja executando unha simulación

- **Z1 Mote**: Esta mota representa a un hardware que, ó igual que a Sky, tamén conta cun transmisor de rede con IEEE 802.15.4 de 2.4GHz a 250kbps e, aínda que ten 8 kilobytes de RAM, ten un procesador MSP430x de 16 bits a 16 MHz e 92 kilobytes de flash. [21] Tamén ten un sensor de temperatura e un acelerómetro integrado aínda que se lle poden engadir mais sensores externos mediante Phidgets [22].

- **Cooja Mote**: É a única mota nativa onde a emulación realízase a alto nivel, o que

provoca que o seu rendemento sexa o mellor de entre todas as motas a cambio de ter un comportamento non reproducible por ningún hardware real.

2.3.3 Protocolos de comunicacións

Contiki ten unha implementación do stack de rede micro IP (a partir de agora, uIP) [23], tanto na súa variante IPv4 como IPv6, e do stack Rime, una alternativa mais lixeira ó stack TCP/IP e orientada a un baixo consumo de enerxía [24].

Ademais, conta co protocolo de enrutado Routing Protocol for Low-Power and Lossy Networks (a partir de agora, RPL) na súa variante de IPv6 xunto coa especificación de IPv6 over Low -Power Wireless Personal Area Networks (a partir de agora, IPv6LoWPAN), deseñada para realizar transmisións cun consumo baixo de enerxía sobre IPv6 [25].

No caso deste proxecto optouse por utilizar o protocolo IPv6 por ser un estándar libre amplamente utilizado, polo que a súa integración noutros proxectos ou sistemas sería mais sinxela, xunto co protocolo de enrutamento nativo de Contiki, RPL.

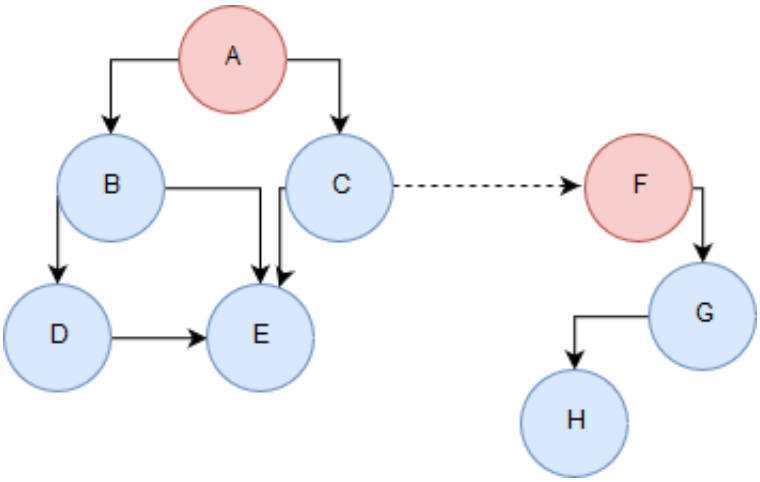


Figura 2.5: Topoloxía de RPL.

O protocolo RPL trata de crear unha topoloxía chamada Destination Oriented Directed Acyclic Graph (a partir de agora, DODAG) (ver figura 2.5) onde un nodo, normalmente o punto de acceso, é o nodo raíz e os demais colgan del. Isto conséguese mediante o envío de paquetes DODAG Information Solicitation (a partir de agora, DIS), para a detección de DODAGs próximos; DODAG Information Object (a partir de agora, DIO), para o envío de información cara os nodos fillos; e Destination Advertisement Object (a partir de agora, DAO), para transmitir información cara os nodos pais [26]. Segundo a figura 2.5, as mensaxes DIO van no sentido das frechas continuas, as mensaxes DAO van no sentido oposto das frechas continuas e as mensaxes DIS van no sentido oposto as frechas punteadas. Tras unha mensaxe DIS, o nodo raíz F pasaría a ser un nodo fillo de C.

O protocolo de aplicación usado para a transmisión dos datos é o de Message Queuing Telemetry Transport (a partir de agora, MQTT) [27], un protocolo baseado no paradigma publicación-subscritor (ver figura 2.6) lixeiro que funciona sobre TCP e que é usado na vida real para este tipo de tarefas.

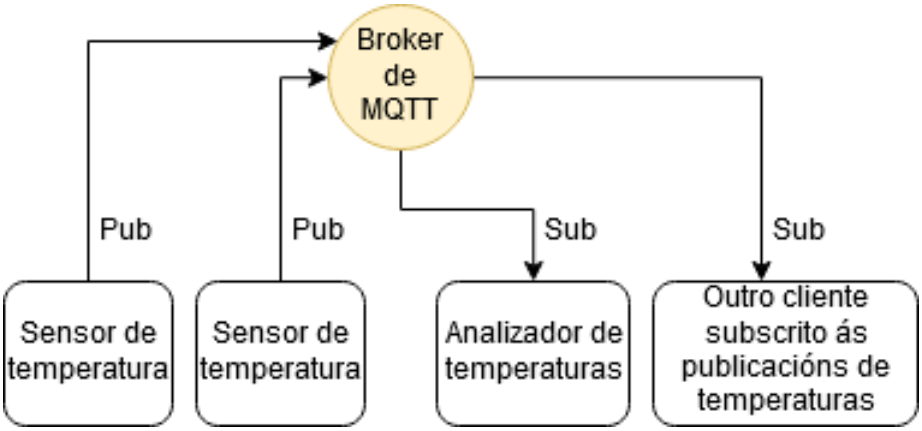


Figura 2.6: Diagrama de funcionamento de MQTT.

Capítulo 3

Metodoloxía do proxecto

A metodoloxía usada neste proxecto é a de PPDIOO (Preparar, Planificar, Deseñar, Implementar, Operar e Optimizar) xa que foi deseñada por Cisco [28] para a creación e mantemento de redes, parte central do proxecto. Está composta dun total de 6 iteracións, as cales teñen como nome as palabras indicadas anteriormente entre parénteses. Ó final deste capítulo tamén se inclúen as historias de usuario relacionadas coas iteraicóns.

3.1 Iteracións

1. **Preparar**. Na fase de preparación, a compañía que use esta metodoloxía debe marcar os seus requisitos de negocio e a visión e estratexia que ten sobre a tecnoloxía a usar. Esta fase considérase un caso de negocio.

   Neste proxecto, úsase esta fase para obter información sobre a simulación a realizar e as características do proxecto. Obtéñense requisitos e realízase un deseño de alto nivel.

2. **Planificar**. Nesta fase realízase un estudo das tecnoloxías actuais e analízanse os requisitos obtidos na anterior fase. Nesta fase tamén se realiza a planificación que o proxecto levará a cabo, onde se asignan responsables e se definen tarefas.

   Neste proxecto, úsase esta fase para realizar un análise do estado do arte sobre os simuladores, sistemas operativos, tipos de sensores, protocolos, etc, e para analizar os requisitos que se teñen que cumplir, como o número de sensores, a cantidade de tempo de execución da simulación, os protocolos de uso obrigatorio, etc.

3. **Deseñar**. Durante a fase de deseño, a compañía debe realizar os planos da rede a baixo nivel usando a información obtida das dúas fases anteriores.

   No proxecto, esta fase úsase para crear diagramas de como é a arquitectura da simulación tendo en conta os requisitos e as características de Cooja e Contiki. Tamén se deseña o código que executarán os sensores.

4. **Implementar**. Durante a iteración de implementación, a compañía realiza a instalación da rede segundo o deseño durante a fase anterior e de acordo ós plazos de execución definidos durante a fase de planificación.

   A fase de implementación neste proxecto realízase mediante a implantación da arquitectura anteriormente deseñada e mediante o desenvolvemento do código necesario para os sensores.

5. **Operar**. Na fase de operar, a compañía pon en funcionamento a rede recén implementada probándoa. É posible que sexa necesario volver a unha fase de deseño se a rede non funcionase de forma esperada. Durante esta fase tamén se realiza a documentación final da rede e se mantén un seguimento e mantemento continuo sobre a mesma.

   No proxecto, esta fase úsase para a execución e monitorización da simulación. Mentres que no caso xeral, a metodoloxía predí que esta fase é a mais longa en duración, no caso deste traballo é unha das fases mais curtas, por só ser necesaria executar a simulación durante unhas poucas horas. Sen embargo, se un requisito do proxecto fose a realización dunha simulación que durase varios meses, si que se axustaría correctamente a esta predicción.

6. **Optimizar**. A fase de optimizar pode iniciarse en calquer momento durante a fase de operar, sendo o normal que ocorra cando hai algún cambio nos requisitos da rede. Se o cambio de requisitos fose moi amplo, é recomendable volver a executar a metodoloxía dende a primeira fase.

   Durante a fase da realización da simulación buscaranse formas de mellorar o rendemento da mesma e de buscar erros que poidan aparecer.

3.2 Historias de usuario

1. **Preparación:**

   1.1. Obtención dos requisitos segundo indica o director.

   1.2. Obtención de manuais e recursos de aprendizaxe sobre IoT e redes de sensores.

2. **Planificación:**

   2.1. Estudo sobre os diferentes Sistemas Operativos para IoT.

   2.2. Estudo sobre os diferentes simuladores destes Sistemas Operativos.

   2.3. Estudo sobre os protocolos de comunicacións en IoT.

   2.4. Busca de información sobre sensores de temperaturas xa existentes.

2.5. Estudo sobre as características positivas e negativas de Cooja.

2.6. Estudo sobre como realizar a captura dun dataset nun entorno como o de Cooja.

3. **Deseño:**

3.1. Deseño dunha arquitectura de rede tendo en conta as características de Cooja e do entorno.

3.2. Deseño do código que usan os sensores tendo en conta as características de Cooja.

4. **Implementación:**

4.1. Instalación e configuración da máquina de Instant Contiki.

4.2. Desenvolvemento do código para simular temperaturas nas motas de Cooja.

4.3. Preparación das instancias de Cooja e das súas correspondentes motas.

5. **Operación:**

5.1. Posta en marcha da simulación e captura do dataset.

6. **Optimización:**

6.1. Detección de erros durante a simulación e solución acorde.

6.2. Realización do análise do dataset para comprobar a calidade do mesmo.

7. **Documentación:**

7.1. Realización da memoria do proxecto.

Capítulo 4

Análise de requisitos e planificación

Neste capítulo están escritas as necesidades que se deben cumprir para desenvolver o proxecto así como as necesidades e características que teñen que cumprir os elementos participantes no mesmo. Comézase cunha descrición dos requisitos que debe cumprir o proxecto por proporcionar o dataset resultante como unha entrada para outro proxecto, os requisitos hardware para soportar a execución da simulación, os requisitos que ten que cumprir o SO e os requisitos que ten que cumprir o simulador. Finalmente realízase unha planificación onde se indican os recursos necesarios a nivel hardware, software e humano, xunto cos seus custos.

4.1 Requisitos dados por proporcionar o dataset como produto para outro proxecto

- O CPD sobre o que se realiza o traballo é de tipo corredor quente, estando este aislado do exterior no centro e cunha fila de servidores por cada lado. Conta cun total de 9 racks para servidores, 4 inrows dedicados a refrixerar o corredor quente expulsando o aire enfriado cara a fóra e un SAI. Como se ve na figura 4.1, cada rack de servidores ten 2 sensores, un na zona quente e outro na zona fría; e cada inrow ten 4 sensores, un para a entrada de aire quente, outro para a saída de aire frío, outro para a entrada de agua fría e outro para a saída de auga quente. Estes son os sensores que deben ser simulados.

  No CPD real os inrows son de tipo APC ACRC103 [29] e os sensores dos racks están compostos polos sensores de temperatura APC AP9335T [30] e os sensores de temperatura e humidade APC AP9335TH [31]. Estes sensores conéctanse a monitores APC NBRK0551 [32] cunha capacidade de ata 6 sensores cada un. Tanto os monitores como as inrows conéctanse a un switch de rede APC AP9224110 [33] entregando os datos á aplicación supervisora APC AP9465 [34].

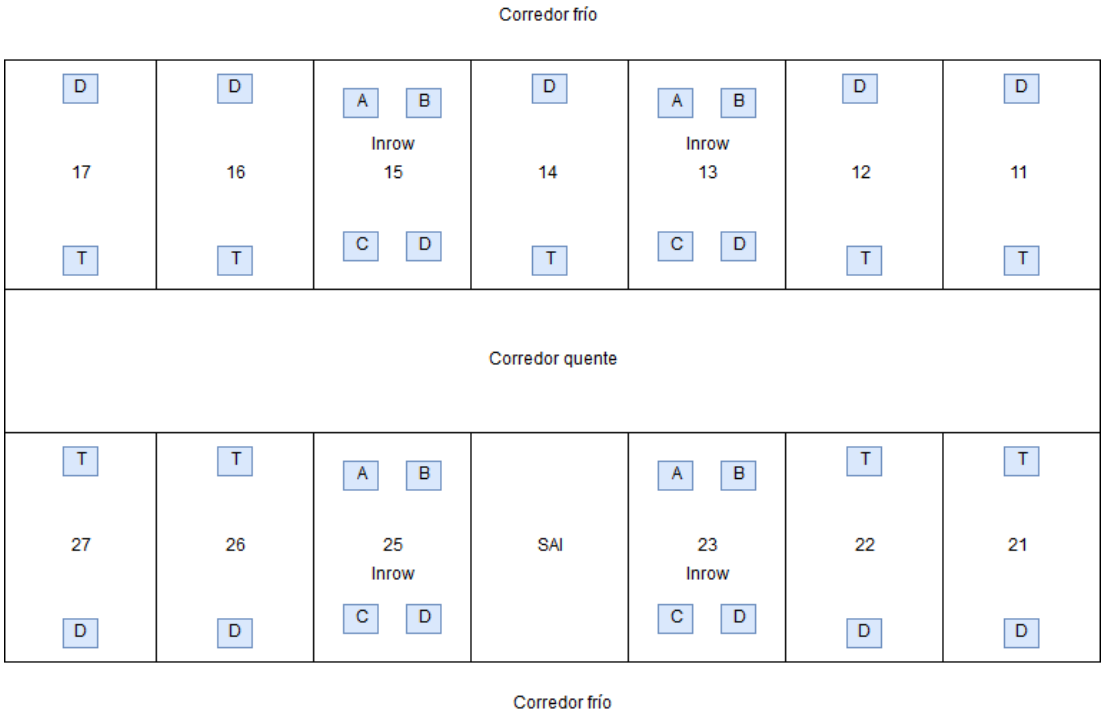


Figura 4.1: Esquema do CPD. As motas están representadas polos cadros azuis. Nos racks: D é a mota dianteira e T a traseira. Nos Inrows: A é a entrada de agua fría, B é a saída de agua quente, C é a saída de aire frío e D é a entrada de aire quente.

- O protocolo a que se está a investigar é MQTT, polo que os sensores, sistema operativo e simulador, deben ser compatibles co mesmo.
- As temperaturas que se deben xerar deben ser realistas e similares ás do CPD real.

4.2 Requisitos hardware

- A máquina sobre a cal se realiza a simulación debe ser capaz de estar acendida continuamente para poder realizar simulacións de longa duración.
- Debe ter capacidades de enrutamento interno do tráfico de rede e capacidade para executar o simulador.

4.3 Requisitos do sistema operativo dos sensores

- Debe poder traballar con redes de sensores.
- Debe ter capacidade para traballar con temperaturas e co recolector físico de datos de temperatura.

- Debe existir un simulador capaz de emular ou simular o seu comportamento.
- Debe ser capaz de usar o stack TCP/IP para a transmisión de datos por rede.
- Debe ser capaz de poder executar o protocolo MQTT.
- Debe ser un software que teña a suficiente madurez como para que sexa esperable un funcionamento estable.
- Debe ser un software cun soporte aceptable para poder resolver mellor potenciais adversidades.

4.4 Requisitos do simulador

- Debe ser capaz de simular nos nodos o stack TCP/IP e o protocolo de aplicación MQTT.
- Debe ter as funcionalidades necesarias para que se poda capturar e exportar o tráfico de rede ou debe permitir que o poda facer unha ferramenta externa.
- A captura de tráfico debe poder facerse sobre unha simulación de longa duración sen errores.
- Debe permitir executar diferentes sensores hardware executando código heteroxeneo á vez nunha mesma simulación.
- Debe permitir o escalado horizontal e permitir a comunicación entre instancias en paralelo.

4.5 Planificación do proxecto

4.5.1 Recursos hardware e software

En canto a hardware, úsase un servidor persoal Dell R710 con 2 procesadores Intel Xeon E5645 de 6 cores con HyperThreading (24 cores lóxicos en total) a 2.4 GHz, 12 sticks de 8 GB de RAM (96 GB en total) e 2 discos HDD de 2TB a 7200 rpm configurados en RAID1 con caché alimentada por batería. Ese hardware úsase de forma non exclusiva para manter a máquina virtual que usa o simulador en execución. O prezo do servidor estímase en 1500€ cun tempo de amortización de 4 anos. Isto significa que durante a duración do proxecto, 6 meses, o custo do servidor foi de 187.50€.

Tamén se utilizou unha computadora persoal para acceder á máquina virtual, realizar as tarefas de investigación e redactar a memoria do traballo. O prezo da computadora foi de

1100€ cun tempo de amortización de 4 anos. Isto significa que durante a duración do proxecto, 6 meses, o custo da computadora foi de 91.67€.

En canto a software, todo o utilizado é gratuito. Algunhas pezas de software como Contiki 3.0, Cooja ou Ubuntu 14.04, son open-source, e outras como o software VMware ESXi, úsase coa licenza gratuita.

Desta forma, a táboa 4.1 indica o custo total do proxecto en recursos materiais.

Concepto	Custo
Servidor	187.50€
Computadora	91.67€
Total	279.17€

Táboa 4.1: Táboa de custos materiais.

4.5.2 Recursos humanos

Para a realización deste proxecto, é necesario contar cunha serie de perfís profesionais para a realización das tarefas.

Os perfís necesarios para a realización de tódalas tarefas son os da táboa 4.2.

ID	Perfil	Descrición
XP	Xefe de proxecto	Monitoriza o proxecto, establece as tarefas por perfil e xestiona os recursos asignados.
AN	Analista	Estudar e analiza os requisitos do proxecto.
PR	Programador	Estuda as librarías e desenvolve o código das motas.
TS	Técnico de sistemas	Instalar, configura e mantén os sistemas informáticos.

Táboa 4.2: Perfís profesionais e custo por perfil

Cada un destes roles realiza unha serie de tarefas durante a duración do proxecto. Estas están definidas na táboa 4.3 xunto coa súa duración.

Desta forma, o total de horas por perfil e o seu custo queda indicado na táboa 4.4, dando un custo humano total de 2783€.

Tarefa	Perfil	Horas
Obtención de requisitos	AN	4
Obtención de manuais e recursos de aprendizaxe sobre IoT e WSN	AN	3
Estudo dos diferentes SO para IoT	AN	4
Estudo dos diferentes simuladores destes SO	AN	4
Estudo dos protocolos de comunicacións en IoT	AN	6
Estudo dos sensores de temperaturas xa existentes	AN	3
Estudo das características de Cooja	AN	10
Estudo sobre a realización da captura dun dataset	AN	2
Deseño da arquitectura de rede	AN	12
Deseño dun plan de probas	AN	4
Deseño do código que usan os sensores	AN	15
Instalación e configuración da máquina de Instant Contiki	TS	7
Estudo da libraría de MQTT-SN	PR	12
Desenvolvemento do código dos sensores	PR	5
Preparación das instancias de Cooja para a simulación	TS	5
Realización das probas de implementación (I*)	TS	4
Execución da simulación e captura do dataset	TS	1
Realización das probas durante a execución da simulación (E*)	TS	4
Comprobación do dataset e análise das características do mesmo	AN	6
Realización da memoria do proxecto	XP	50
Total		161

Táboa 4.3: Tarefas a realizar e horas por tarefa

Perfil	Horas totais	Custo por hora	Custo total
XP	50h	20€/h	1000€
AN	73h	19€/h	1387€
PR	17h	12€/h	204€
TS	21h	12€/h	252€
Total			2843€

Táboa 4.4: Custo por perfís

Capítulo 5

Implementación, simulación e probas

Neste capítulo trátase a implementación da simulación dende a instalación ata a configuración dos elementos. Ademais, trátase o proceso de execución da simulación e defínense un conxunto de probas.

5.1 O entorno

Debido a que a natureza de este tipo de proxectos poden requirir da execucións de simulacións de longa duración, decídese que é recomendable instalar o entorno de traballo como unha máquina virtual nun servidor con capacidades de virtualización.

Nun primeiro paso, cumplindo coa historia de usuario 4.1., prepárase o servidor instalándolle un sistema operativo como Proxmox [35] ou VMware ESXi [36].

No momento de instalar o entorno de Contiki existen dúas aproximacións:

- Instalar un sistema operativo como Debian e nel descargar, compilar e preparar Contiki.
- Descargar a máquina virtual de Instant Contiki [37], a cal xa está case lista para o seu uso de base.

Probáronse ámbalas dúas opcións e escolleuse usar a máquina virtual de Instant Contiki debido a que simplifica a instalación e evita a realización de pasos adicionais de forma innecesaria.

Para a posta en marcha da máquina virtual pódense seguir os seguintes pasos por primeira vez:

1. Descargar Instant Contiki 3.0 en formato zip dende a páxina oficial. Descomprimir a máquina e importala no entorno de VMware.

2. Arrancar a máquina e acceder como *user* (usuario por defecto) co contrasinal *user*. É importante non instalar actualizacións, posto que a cantidade de disco duro co que conta a máquina é moi limitada. Pode ser necesario instalar a distribución do teclado español. É importante non executar ningunha destas instruccións como superusuario, só usando *sudo* cando se indique.

3. Instant Contiki contén dúas instalacións de Contiki, aínda que a importante é a de Contiki 3.0. Para quedarnos só co cartafol de Contiki 3.0, pódense executar os seguintes comandos:

```
cd ~
rm -rf contiki
mv contiki-3.0/ contiki/
```

4. Agora é necesario instalar o simulador MSPSIM para poder emular as motas Sky. Para conseguilo, pódese executar a seguinte lista de comandos:

```
cd ~/contiki/tools/
rmdir mspsim
wget https://github.com/contiki-os/mspsim/archive/master.zip
unzip master.zip
rm master.zip
mv mspsim-master/ mspsim/
```

5. Tamén é necesario compilar a ferramenta necesaria para poder realizar comunicacións dende dentro da simulación á máquina host. Este programa chámase tunslip e pódese compilar con estes comandos:

```
cd ~/contiki/tools/
make
make tunslip6
```

6. Para rematar, é necesario habilitar o enrutamento de tráfico IPv6 no SO co seguinte comando:

```
sudo sysctl -w net.ipv6.conf.all.forwarding=1
```

Unha vez executados estes pasos, xa só é necesario compilar e arrancar o simulador de Cooja cos seguintes comandos. É importante non facelo como superusuario.

```
cd ~/contiki/tools/cooja
ant run
```

Agora a interface de Cooja ten que aparecer en pantalla. É importante non pechar a liña de comandos dende onde se lanzou Cooja.

5.2 Simulación de temperaturas

A finalidade deste proxecto é simular unha serie de temperaturas nas motas dentro de Cooja e, por isto, é necesario estudar e entender como traballa o simulador coas temperaturas.

Aínda que o sistema operativo Contiki é compatible coas redes de sensores de temperatura e as motas de Sky e Z1 teñen sensores de temperatura integrados, Cooja non ten incluído ningún módulo de simulación de temperaturas [38].

Sen embargo, tras estudar o caso, encontráronse dúas solucións posibles:

- **Editar o código fonte de Cooja**, en Java, para engadir a funcionalidade ó simulador. Isto pódese facer indo a */contiki/tools/cooja/apps/mspsim/src/org/contikios/cooja/mspmote/plugins*, editando o arquivo *interfaces/SkyTemperature.java* e engadindo a clase no *getAllMoteInterfaceClasses* do *SkyMoteType.java* [39]. A vantaxe é que o código das motas é o mesmo que se usa nas motas reais, pero ten o inconveniente de que se está editando o código do simulador de Cooja, facendo a resolución de erros mais difícil e incrementando a complexidade do proxecto. Tamén tería o problema de ter que identificar dende o simulador ás motas para poder entregarlle a cada unha o seu correspondente conxunto de temperaturas.

- **Modificar o código fonte das motas**, en C, para engadir a funcionalidade á mota. A vantaxe é que o proxecto mantense sinxelo, non hai que modificar o simulador e cada mota xa ten os seus datos de temperatura no seu código. Ten o inconveniente de que o código das motas é diferente do código que usan as motas reais.

Neste caso, por simplicidade e para evitar problemas, óptase por editar o código das motas. A única desvantaxe que ten esta elección é que o código das motas xa non é o que se usaría nas motas reais. Sen embargo, isto non se considera un problema xa que, unha vez se porte o código ás motas reais, pódese editar a orixe dos datos na función que obtén a temperatura, pasando de obtelos dunha función matemática a unha chamada da API do sensor.

5.2.1 As funcións de temperaturas

Para cada mota existe unha función que xera un dato de temperatura en base a uns valores establecidos e cedidos polo director do proxecto, os cales foron obtidos dun estudo realizado no CPD do CITIC. As funcións son distintas entre os inrows e os racks:

- Para os racks úsase unha función que dados 2 ou 3 valores con certa probabilidade cada un, xera un valor aleatorio.

- Para os Inrows úsase unha función que suma un valor base, un valor dunha táboa segundo o momento da simulación e un valor dunha función aleatoria (ver figura 5.1).

Os valores da táboa fan que o patrón de temperaturas se repita a cada hora, como se observou no CPD real e segundo os datos proporcionados polo director.

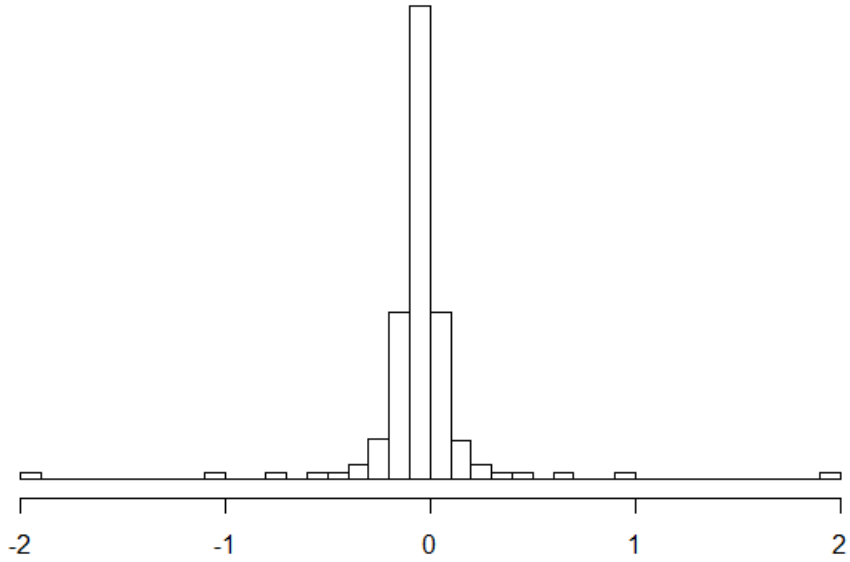


Figura 5.1: Distribución redondeada a 1 decimal dos valores da función aleatoria para 100.000 de mostras.

5.3 Rendemento do simulador

Cooja é un emulador monothread [40](ver figura 5.2) polo que executa nun só fío a emulación do código de todas as motas da súa instancia. Isto fai que o rendemento do simulador dependa da velocidade dun só núcleo da CPU, non sendo útil asignarlle mais. Unha simulación cun tamaño o suficientemente grande, aínda que se execute sobre unha CPU moderna e a unha frecuencia elevada, segue tendo limitada a cantidade de motas que o núcleo é capaz de emular. Así que, se se quixese aumentar o tamaño da simulación por enriba deste límite, quedarían dúas posibles opcións:

- Engadir mais motas á simulación e executala a unha velocidade menor á real como, por exemplo, executala a unha velocidade do 1%. Isto é posible se a simulación só se realizase dentro de Cooja e se a precisión temporal para o proxecto non fose crítica.
- Executar múltiples instancias de Cooja e en cada unha delas simular un subconxunto das motas. Isto reparte a carga entre as instancias, as cales se executarán en cores diferentes, conseguindo unha experiencia escalable e multithreading.

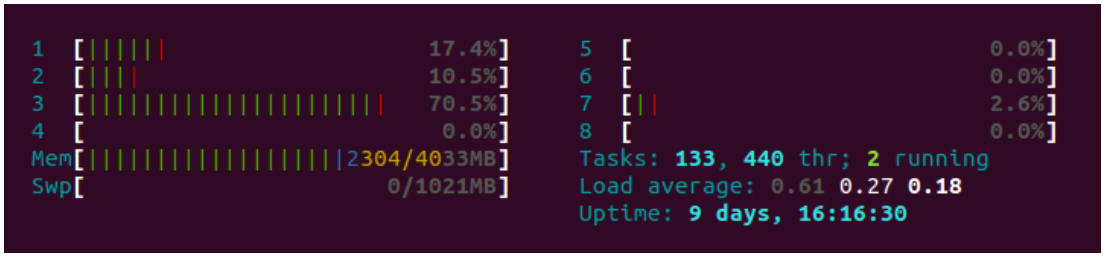


Figura 5.2: Execución dunha instancia de Cooja sen límite de velocidade. Só un dos núcleos estase a utilizar plenamente.

Para este proxecto é necesario usar a segunda solución posto que é unha simulación sensible ó tempo. Ademais, esta opción permite tamén escalar fóra da máquina virtual, podendo engadir mais máquinas virtuais con varias instancia de Cooja e deixar que un router externo reenvíe os paquetes ó broker MQTT, o que permitiría escalar de forma horizontal.

5.4 Arquitectura da rede

Para conseguir o dataset do resultado da simulación é necesario crear unha rede pola que podan fluír os paquetes cos datos e é necesaria unha ferramenta para poder capturalos.

O primeiro a ter en conta e que o broker de MQTT é unha peza de software que pode ser executada na máquina virtual de forma nativa, mentres que as motas colectoras dos datos de temperatura teñen que estar dentro da simulación de Cooja.

Para poder intercomunicar ambos elementos, pódese usar un RPL Border Router.

5.4.1 RPL Border Router

Un Border Router é unha mota que executa un software encargado de enrutar paquetes entre a rede RPL do simulador e unha rede externa. Para cumprir este cometido, a mota de Border Router é a raíz da rede RPL na simulación na que se encontra. Para conectala cunha rede externa é necesario facer un túnel aproveitando o seu porto serie. Isto é posible facelo abrindo a ventá de *Serial Socket* da mota facendo clic dereito sobre a mota → *Mote tools for [...]* → *Serial Socket (SERVER)* (ver figura 5.3).

Nesta ventá seleccionaríase un porto que estivese aberto en TCP e pulsaríase en Start para abrir ese porto a escoitas dende fóra do simulador (ver figura 5.4).

Na máquina virtual na que se está executando Cooja é necesario crear unha interface virtual asociada a ese socket. Para elo Contiki ten a ferramenta chamada *tunslip6* no cartafol *tools* que se encarga de todo. Un exemplo do comando que crea unha interface chamada tun1 conectada ó porto 60001 e asignándolle o prefixo de rede IPv6 a001 é o seguinte:

```
sudo ./tunslip6 -a 127.0.0.1 -p 60001 a001::1/64 -t tun1
```

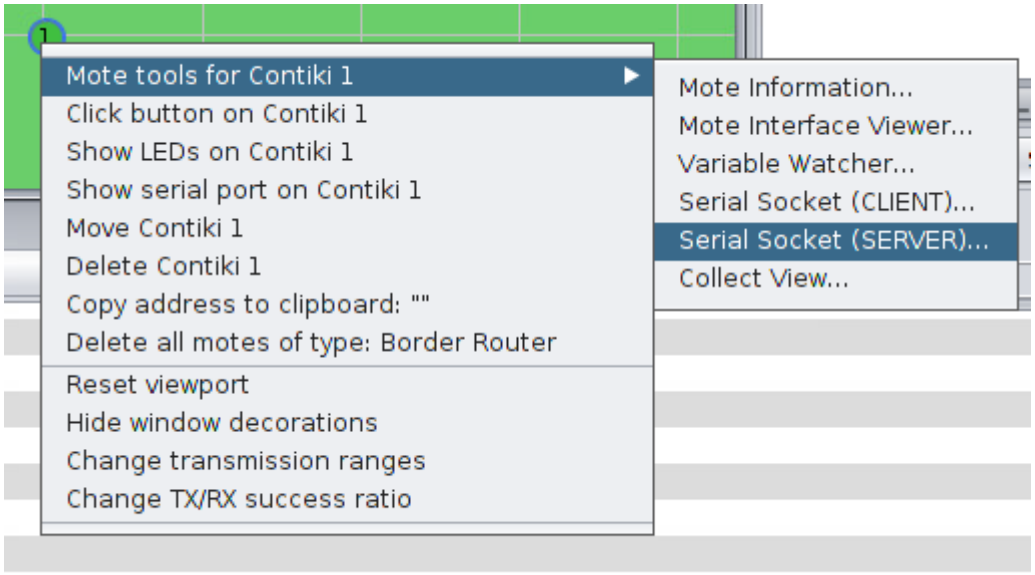


Figura 5.3: Como abrir a ventá de Serial Socket.

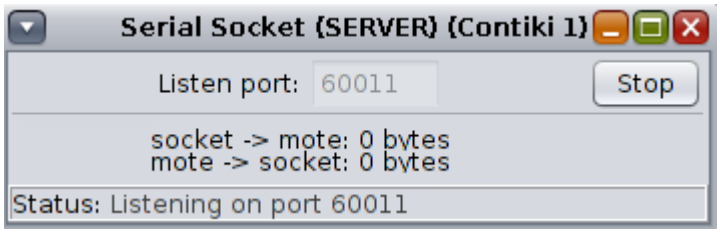


Figura 5.4: Serial Socket esperando por unha conexión.

Desta forma (ver figura 5.5), se o sistema operativo ten activado o enrutado de paquetes de IPv6, é posible contactar con tódalas motas dentro dunha simulación coa dirección da mota correspondente pero substituíndo o prefixo local polo indicado.

Se nalgún momento se recargase a simulación, romperíase o túnel creado e teríase que volver a executar o comando de *tunslip6*.

5.4.2 Captura de paquetes

Durante a execución da simulación, é necesario realizar a captura dos paquetes para poder obter o dataset final. Neste aspecto existen varias opcións, das cales se consideraron as dúas seguintes:

- Usar a ferramenta *Radio Messages* (ver figura 5.6) da propia instancia de Cooja, que se atopa no menú *Tools*. Ten a vantaxe de estar xa integrada dentro da instancia do simulador, proporciona un overhead menor, é sinxela de usar e non require de ferra-

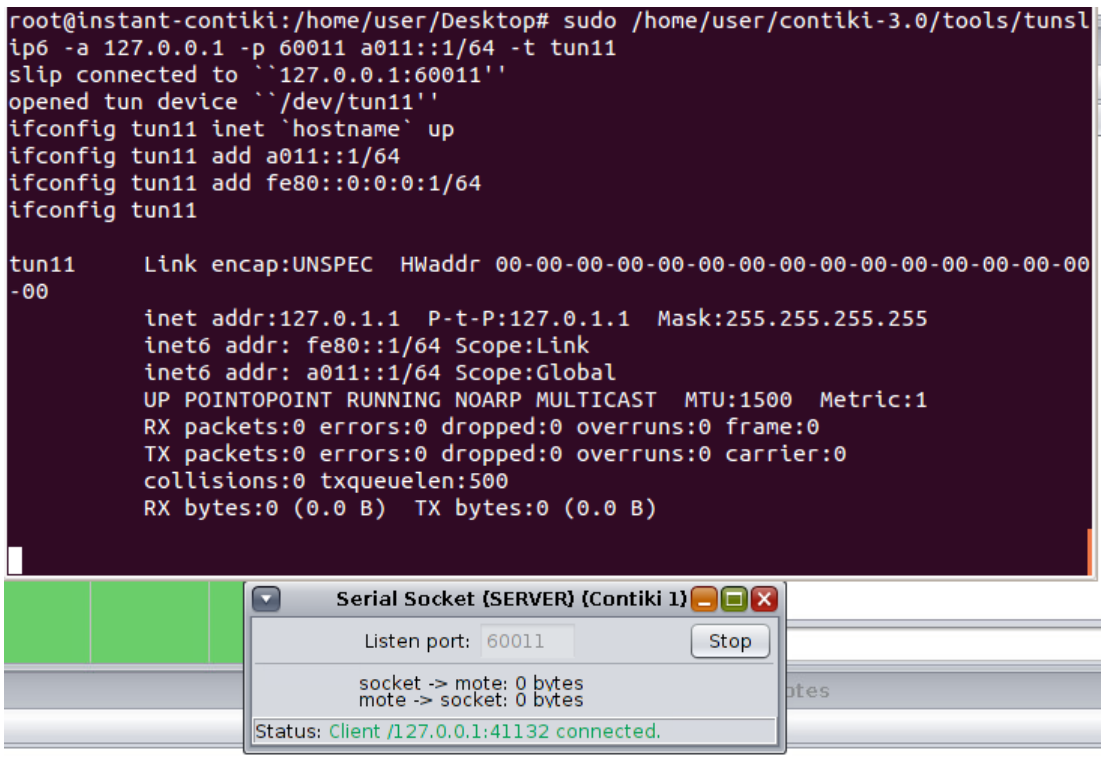


Figura 5.5: Serial Socket conectado por tunslip.

mentas externas. Ademais, pode capturar o tráfico interno entre as motas que non sae ó exterior. Ten o inconveniente de que se rompese a instancia durante a execución da simulación por algún motivo, a captura podería corromperse ou perdese e de que non pode capturar o tráfico do exterior da instancia.

- Usar unha ferramenta externa para realizar a captura de paquetes, como *WireShark*, na máquina host contra a interface creada. Ten a vantaxe de non depender do simulador e non perder a captura se a instancia rompese, ademais de poder capturar varias interfaces á vez nun único dataset. Ten o inconveniente de que non pode capturar o tráfico interno dentro da instancia, podendo só capturar aquel tráfico que saia ó exterior.

5.5 Preparación das instacias

Nesta sección cúmplese a historia de usuario 4.3.. Tendo en conta que a simulación usa como referencia ós sensores do CPD do CITIC (ver figura 4.1), é necesario simular un total de 34 motas. Aínda que o rendemento do simulador varía dependendo do software de cada mota, pois non é o mesmo emular unha mota que xera datos e os transmita a cada segundo

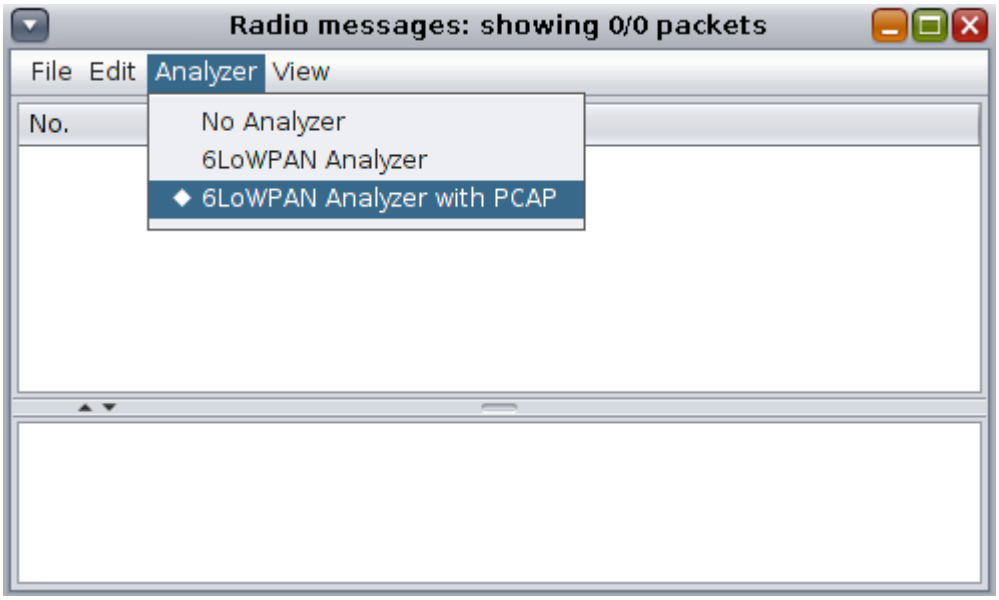


Figura 5.6: Interface de Radio Messages. É necesario habilitar a captura nun PCAP.

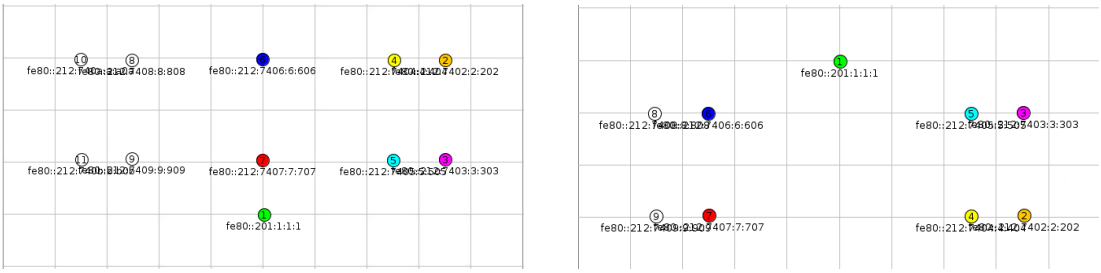
que unha que o fai cada 5 minutos, é recomendable non sobrecargar o simulador para evitar caídas do 100% da velocidade de simulación.

Para este proxecto é necesario distribuir a carga de motas en, a lo menos, dúas instancias. Sen embargo, para manter unha mellor organización, exemplificar mais a capacidade de escalado horizontal da arquitectura e mellorar o rendemento e estabilidade xeral, decídese separar as motas en 4 instancias con entre 8 e 10 motas por instancia, sen contar o Border Router, seguindo a seguinte distribución:

- Motas dos racks superiores (rackA) (ver figura 5.7a)
- Motas dos racks inferiores (rackB) (ver figura 5.7b)
- Motas dos inrows superiores (inrowA) (ver figura 5.7c)
- Motas dos inrows inferiores (inrowB) (ver figura 5.7d)

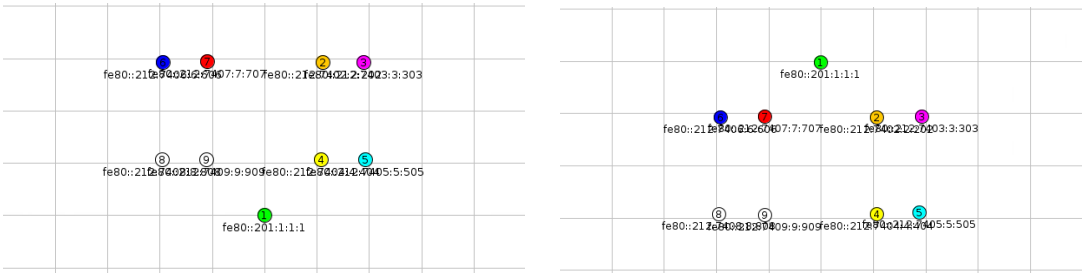
Todas as motas son de tipo sky excepto o border router que é de tipo Cooja Mote. Isto débese a que non é necesario ter emulación hardware de alta precisión nun compoñente auxiliar como o é un border router que só se usa para permitir a comunicación entre o simulador e o exterior. O uso da mota sky en lugar de outras como a Z1 débese a que en probas realizadas no laboratorio, só a sky mostrou estabilidade á hora de crear rutas con RPL, mentres que Z1 volvíase imprevisible, non creando rutas esenciais entre motas en moitos casos.

Desta forma, o resultado é unha máquina virtual que executa catro instancias de Cooja, un broker MQTT e unha instancia de WireShark (ver figura 5.8).



(a) Racks superiores (rackA)

(b) Racks inferiores (rackB)



(c) Inrows superiores (inrowA)

(d) Inrows inferiores (inrowB)

Figura 5.7: Colocacións das motas nas 4 instancias de Cooja. A mota 1 e de cor verde, é o Border Router en cada subfigura.

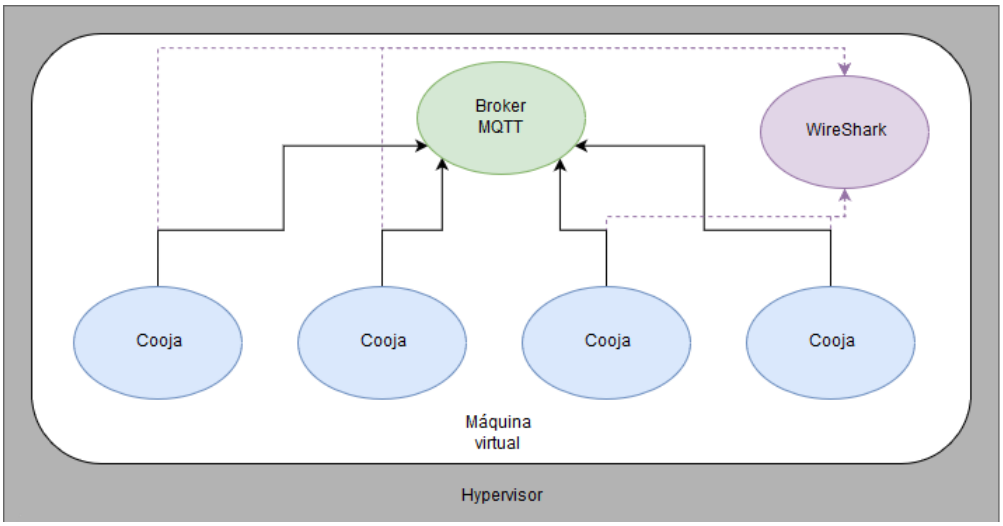


Figura 5.8: Esquema da máquina virtual.

Tamén é necesario dimensionar a máquina virtual acorde á carga de traballo que se espera simular. Cooja utiliza 150 MB de RAM ó arranque, que aumentan ata 370 MB ó executar a simulación dunha das instancias definidas. Cooja tamén é monothread [40], polo que cada unha das instancias usa un core.

Asignándolle un total de 8 cores e 4 GB de RAM (ver figura 5.9) á máquina virtual pódese

dedicarlle 1 core a cada unha das instancias e os outros 4 para o WireShark, para o broker de MQTT e para o sistema operativo. Así, tamén se lle pode dedicar 2 GB de RAM ás instancias, usando 500 MB por instancia, e 2 GB para WireShark, para o broker de MQTT e para o sistema operativo. Se ben é certo que se podería chegar a asignar os recursos de forma mais axustada, é recomendable sobreasignalos lixeiramente para manter a estabilidade e rendemento no sistema.

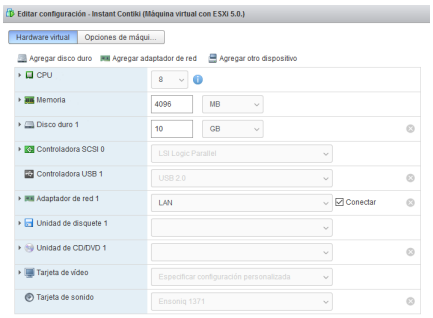


Figura 5.9: Interface de ESXi mostrando os recursos proporcionados.

Isto repercute na elección da solución na ferramenta de captura de tráfico de Cooja, xa que a única forma de ter un dataset unificado con tódalas motas é usando unha captura de varias interfaces dende *WireShark* ou outra ferramenta externa.

5.6 A libraría de MQTT

Para realizar a comunicación co protocolo MQTT, úsase unha versión simplificada chamada MQTT-SN [41]. Esta elección faise tendo en conta que son o suficientemente similares como para non producir problemas no proxecto de investigación.

Algunhas diferencias principais de MQTT-SN con MQTT son:

- Utiliza UDP en lugar de TCP para reducir o overhead da conexión.
- Non é obrigatorio identificar as publicacións co topic, senón que se pode realizar cun número, reducindo a cantidade de memoria e ancho de banda durante a execución.
- Existe unha modificación nos mensaxes *keep-alive* para mellorar a duración da batería nas motas.
- Existe un nivel de QoS de -1 chamado *fire and forget* no que non se ten ningún control na transmisión dos mensaxes, algo común en redes de sensores.
- Non require o uso do stack TCP/IP, podendo realizarse a comunicación mediante porto serie [42].

O broker de MQTT-SN a utilizar é rsmb [43] por ser lixeiro e compatible con Linux.

En canto ó código das motas, úsase a libraría do repositorio de github *aignacio/mqtt-sn-contiki_example* [44] xa deseñada para Contiki e con licencia Apache 2.0 [45].

A libraría tamén inclúe un exemplo sobre como se usa a mesma. Este exemplo é usado de base para o desenvolvemento do código das motas usadas polo proxecto.

Para instalar a libraría, pódese executar os seguintes comandos:

```
cd ~/contiki/
git clone \
https://github.com/aignacio/mqtt-sn-contiki_example \
mqtt-sn-contiki
cd mqtt-sn-contiki
make all
cd tools
unzip mosquitto.rsmb.zip
cd mosquitto.rsmb/rsmb/src
make clean
make
```

Os dous últimos comandos son necesarios se se usa unha máquina de 32 bits como o é a de Instant Contiki pois, o executable precompilado, tal e como indica o autor da libraría, o está para sistemas de 64 bits, non podendo executalo sen facer a recompilación.

Dentro do novo cartafol, existe unha serie de ficheiros de código que deben ser copiados ó cartafol onde está o código do proxecto. Os ficheiros que deben ser copiados son: *main_core.c*, *mqtt_sn.c*, *mqtt_sn.h*, *project-conf.h* e *Makefile*. É importante editar o arquivo *Makefile* do proxecto para que a variable *CONTIKI* referencie ó cartafol raíz de Contiki.

5.7 O código das motas

Tras estudar o código de exemplo da libraría de MQTT-SN, decídese realizar unha adaptación do mesmo para ser utilizado como código base das motas do proxecto. Desta forma, cúmplese coa historia de usuario 4.2..

Por unha banda, o código que utiliza o border router é o que trae por defecto a propia instalación de Contiki, por realizar exactamente a tarefa que se espera del. Pola outra banda, o código das demais motas é similar entre si e só varía na función xeradora de temperaturas e nas súas variables.

O código ten a seguinte estrutura:

1. Iniciar a libraría de MQTT, realizando as conexións necesarias e rexistrando os topics da mota.

2. Obter o dato de temperatura actual usando, neste caso, a función xeradora de temperaturas.

3. Envialo por MQTT ao topic xeral /tempall e a un topic para cada mota, como por exemplo /tempI13A para a mota I13A.

4. Esperar 5 minutos e volver a comezar no punto 2.

Decídese enviar o dato duplicado a dous topics para, por unha banda, obter redundancia e, pola outra, ter unha forma nativa de MQTT de agregar todo o tráfico e separar o tráfico por mota.

A periodicidade é de 5 minutos porque, segundo o director do proxecto, observouse que o tempo de transmisión de temperaturas dos sensores do CPD era tamén de 5 minutos. As temperaturas están modeladas para este intervalo de tempo e a función que xera temperaturas ten en conta este dato.

Debido a que non existe unha Unidade de Coma Flotante, ou FPU polas siglas en inglés, na maioría dos sensores físicos, Contiki non soporta traballar con números flotantes [46]. Por esta razón, á hora de traballar coas funcións de temperatura, débense convertir os datos en enteiros mediante multiplicacións e realizar con eles as operacións. Unha vez feitas, queda volver a reconverter de novo os datos mediante divisións.

Tras escribir o código de tódalas motas, é necesario introducilo e compilalo en 4 proxectos de Cooja, un para cada instancia, seguindo o patrón definido na sección 5.5. Para facelo, débese ir ó menú de Cooja *Motes* → *Add motes* → *Create new mote type* → *Sky mote...*, seleccionar o arquivo *main_core.c* e facer clic en *Compile*. Agora pódese engadirlle un nome á mota e facer clic en *Create* (ver figura 5.10).

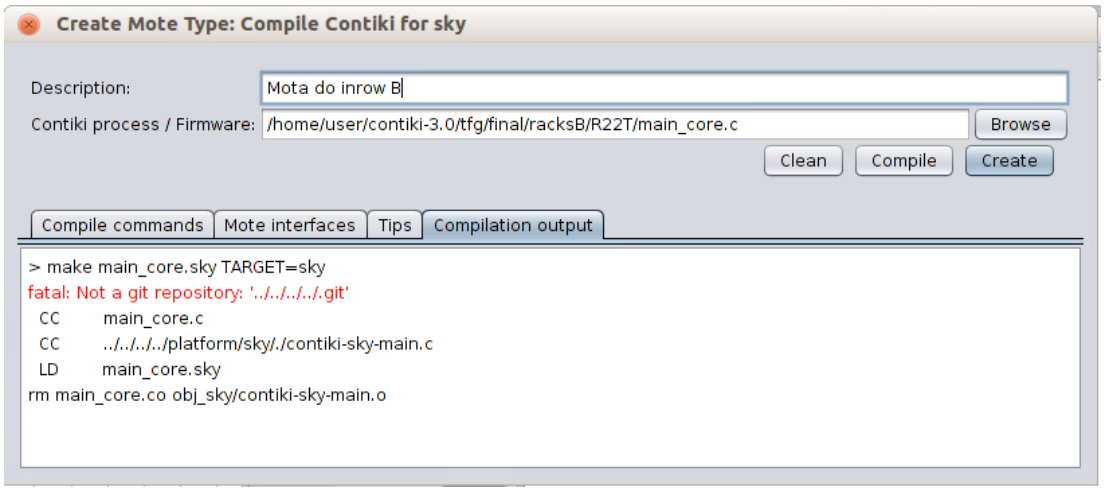


Figura 5.10: Ventá de compilación das motas.

Na seguinte pantalla débese indicar que só se engade 1 mota, o que colocará a nova mota nun lugar aleatorio do mapa. Finalmente pódese colocar a mota no lugar mais axeitado de forma manual (ver figura 5.11).

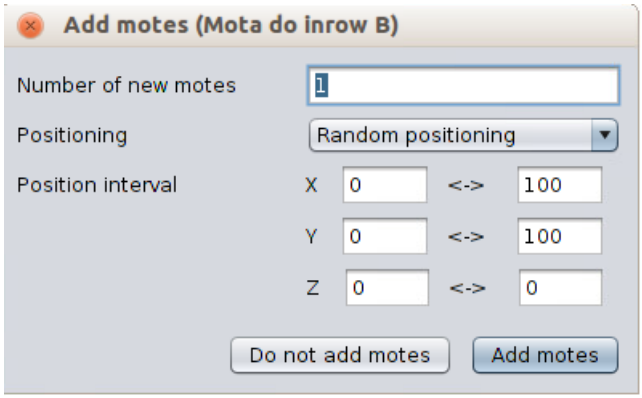


Figura 5.11: Ventá para engadir motas á simulación.

Unha vez engadidas todas as motas da instancia, é necesario gardar o proxecto, para o cal pódese ir ó menú *File* → *Save simulation as...* para escoller por primeira vez onde gardar o proxecto.

5.8 Execución da simulación

Unha vez está o código a punto para a simulación e cada unha das instancias gardadas como proxectos de Cooja, pódese comezar coa simulación, cumplindo a historia de usuario 5.1..

O primeiro paso consiste en abrir as 4 instancias de Cooja. Para evitar ter que abrir moitas ventás de terminal, pódese executar os seguintes comandos para que se abran as 4 instancias de Cooja nunha soa ventá de terminal:

```
cd ~/contiki/tools
ant run &
ant run &
ant run &
ant run &
```

En canto abran as 4 instancias, pódese abrir os 4 proxectos de Cooja nelas (ver figura 5.12), un en cada unha. É recomendable asociarlle a cada instancia un número e mantelo anotado para poder ser consistentes coa numeración ip, a numeración de TCP e a numeración da interface de rede. No caso deste proxecto os números por proxecto son: inrowsA o 1, inrowsB o 2, racksA o 11 e racksB o 12.

Unha vez aberto o proxecto, é necesario abrir a ventá de Serial Socket se non estivese xa aberta, tal e como se indica na sección de arquitectura da rede (5.4). No número de porto TCP é recomendable usar un porto alto rematado no número do proxecto correspondente. Así, decidiuse que o porto de inrowsA é o 60001, o de inrowsB é o 60002, o de racksA é o 60011 e o de racksB o 60012 (ver figura 5.13)

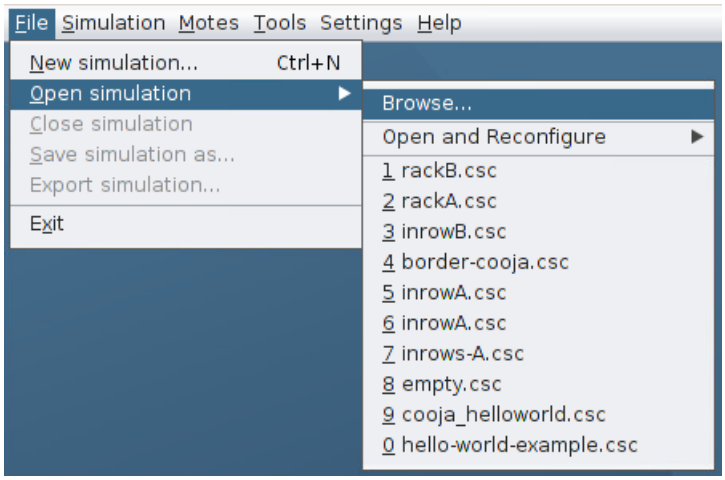


Figura 5.12: Ventá para abrir un proxecto en Cooja.

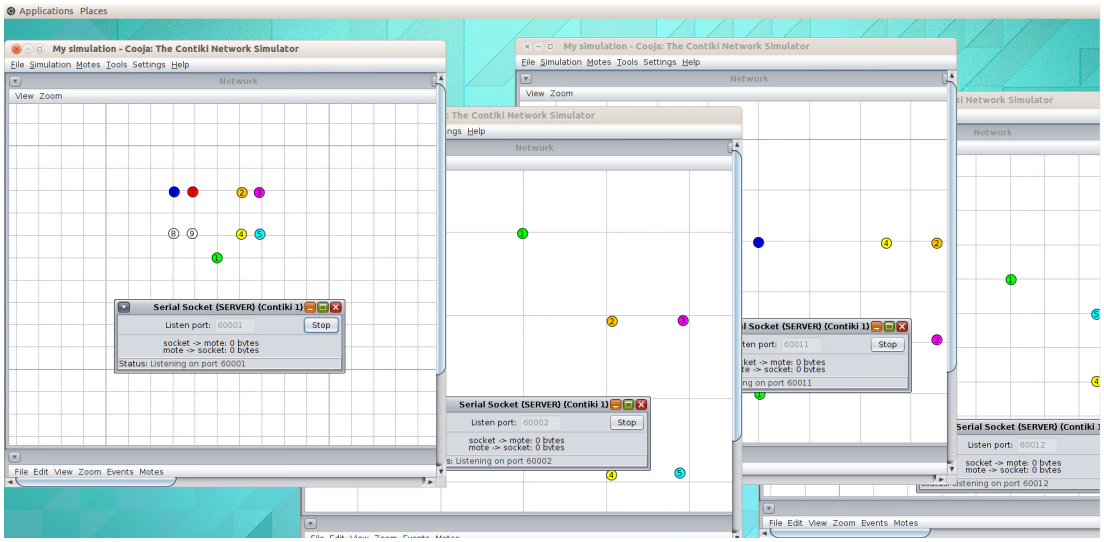


Figura 5.13: As 4 instancias de Cooja cos seus respectivos portos preparados.

Unha vez abertos os portos, é necesario crear as 4 interfaces de rede e asocialas con cada instancia mediante o comando tunslip6. Isto pódese facer mediante os seguintes comandos nunha soa ventá de terminal:

```
cd ~/contiki/tools/
sudo ./tunslip6 -a 127.0.0.1 -p 60001 a001::1/64 -t tun1 &
sudo ./tunslip6 -a 127.0.0.1 -p 60002 a002::1/64 -t tun2 &
sudo ./tunslip6 -a 127.0.0.1 -p 60011 a011::1/64 -t tun11 &
sudo ./tunslip6 -a 127.0.0.1 -p 60012 a012::1/64 -t tun12 &
```

O seguinte paso e abrir o broker de MQTT (ver figura 5.14) na máquina host para que os sensores poidan conectarse a el. Isto pódese facer mediante os seguintes comandos nunha nova ventá de terminal:

```
cd ~/contiki/mqtt-sn-contiki/tools/mosquitto.rsmb/rsmb/src/
./broker_mqtts config.mqtt
```

```
20190731 173849.430 CWNAN9999I Really Small Message Broker
20190731 173849.430 CWNAN9998I Part of Project Mosquitto in Eclipse
(http://projects.eclipse.org/projects/technology.mosquitto)
20190731 173849.431 CWNAN0049I Configuration file name is config.mqtt
20190731 173849.431 CWNAN0053I Version 1.3.0.2, May 20 2019 20:39:33
20190731 173849.431 CWNAN0054I Features included: bridge MQTTS
20190731 173849.431 CWNAN9993I Authors: Ian Craggs (icraggs@uk.ibm.com), Nichola
s O'Leary
20190731 173849.431 CWNAN0014I MQTT protocol starting, listening on port 1883
20190731 173849.431 CWNAN0300I MQTT-S protocol starting, listening on port 1884
```

Figura 5.14: Broker de MQTT-SN.

É posible subscribirse ó topic do broker que se queira se se instala un cliente de mosquitto.

```
sudo apt-get install mosquitto mosquitto-clients
mosquitto_sub -t /tempAll
```

A continuación, é necesario abrir a ferramenta de captura de paquetes e preparala para o traballo. Neste caso, consiste en abrir wireshark e seleccionar as catro interfaces *tun* para capturar o seu tráfico (ver figura 5.15). Unha vez seleccionadas, pódese facer clic en *Start* para comezar a captura.

Para rematar, só queda executar a simulación. Unha vez WireShark esté capturando as interfaces, compróbase que tódalas instancias teñan a velocidade de simulación no 100% e pódese darlle a *Start* (ver figura 5.16).

Tras agardar o tempo que se considere necesario, pódese rematar a simulación parando a captura de WireShark e parando a simulación das instancias en Cooja. Neste proxecto o tempo de simulación é de 3 horas, pois é o suficiente para crear o dataset de temperatura que se buscaba.

5.9 Probas

É importante que cando se estea facendo a implementación e a simulación do proxecto, exista unha forma de validar e comprobar que todo está funcionando correctamente e sen erros para así poder cumplir a historia de usuario 6.1..

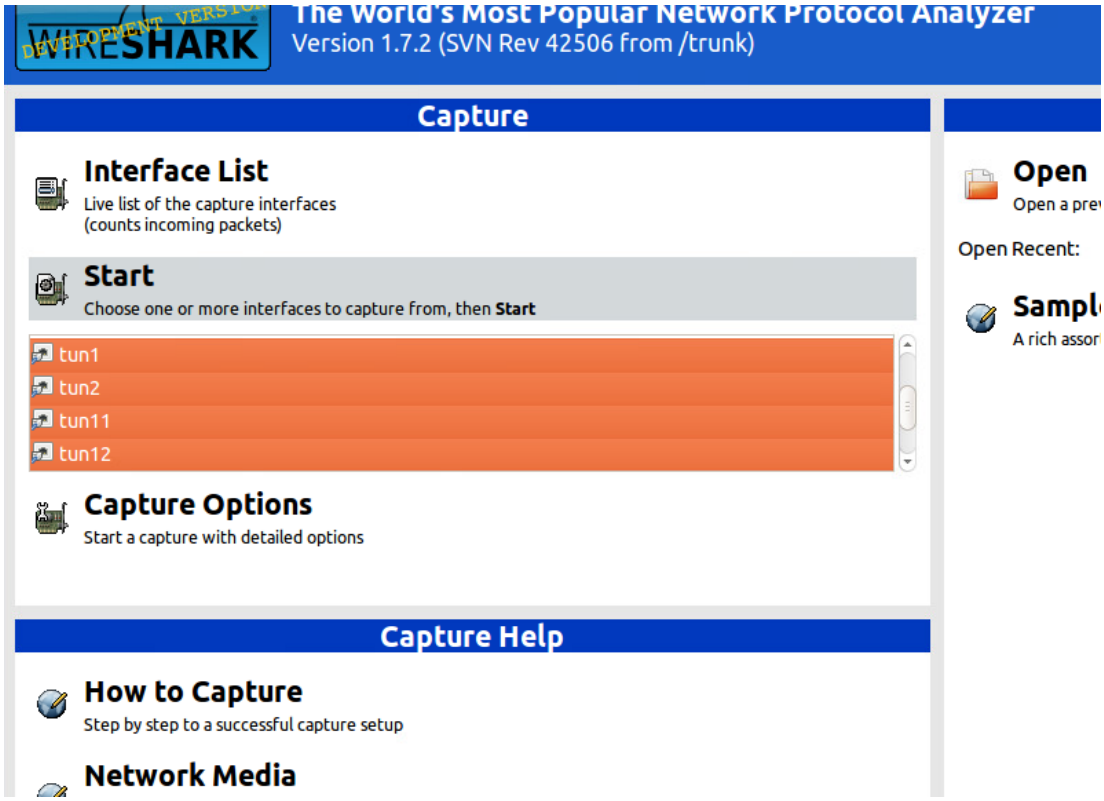


Figura 5.15: WireShark coas 4 interfaces seleccionadas.

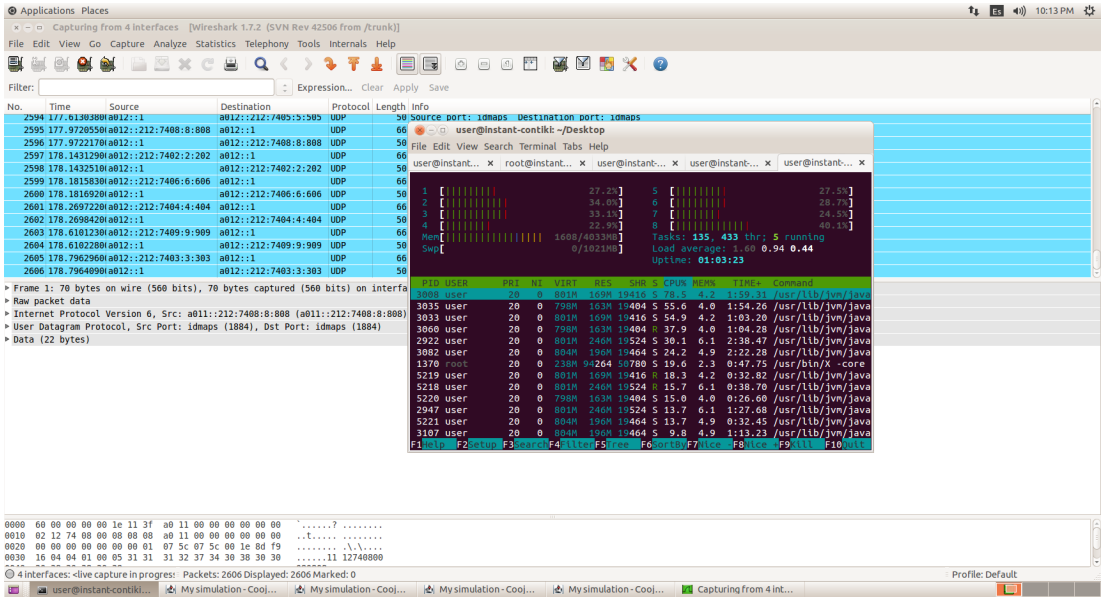


Figura 5.16: WireShark capturando paquetes da simulación cunha ventá de terminal mostrando a carga do sistema mediante a ferramente htop.

5.9.1 Implementación

I1 No apartado da instalación, pódense comprobar que se instalou correctamente Cooja se ao abrila non da ningún erro. Se non fose así, é posible que algún paso se executase como superusuario, tendo que reparar os permisos dos cartafoles. Unha vez aberto, pódese comprobar se MSPSIM foi correctamente instalado ao intentar compilar unha mota Sky cun software de exemplo como o *Hello World*. Se ocurrise algún error, é necesario comprobar o cartafol de */contiki/tools/mspsim* e os seus permisos.

I2 No apartado do código das motas, comprobar que o código compila sen problemas é doado xa que, á hora de compilar a mota para incluíla na simulación, pódense ver os erros se os houbese. Se houbese erros durante a execución, a simulación tamén fallaría mostrando unha mensaxe de erro en pantalla.

I3 No apartado de rendemento do simulador, se a instancia non é capaz de soportar toda a carga que se lle require, pódese observar como a velocidade de simulación baixa do 100% nalgún momento da simulación. A solución pasa por cargar menos a instancia e dividir mais a carga. É importante que a máquina host que está a soportar a carga de todas as instancias teña os recursos suficientes para elas e mais para si mesma para evitar que o sistema se conxele ou se perda información.

I4 A parte de arquitectura de rede pode ser mais complexa de probar. Tras preparar unha instancia cun *Border Router*, se a ventá de *Serial Socket (SERVER)* só indica *Listening...* e non *Connected...*, é que falta executar a ferramenta *tunslip6*. É importante comprobar que se usa o mesmo porto TCP e se está a usar tunslip6 e non tunslip, xa que esta última é para IPv4. Se a conexión faise de forma correcta pero non existe tráfico entre o exterior e a instancia, é recomendable comprobar que se executou o comando para permitir o enrutamento IPv6 por parte do Kernel. Unha forma de comprobar se existe tráfico cara o exterior e intentar capturar as interfaces *tunX* mediante unha ferramenta como *WireShark*.

5.9.2 Execución

Durante a execución da simulación é recomendable comprobar que todo se está executando perfectamente e, polo tanto, é recomendable facer o seguinte:

E1 Comprobar no terminal onde se está executando o broker que ten tráfico coas motas.

E2 Comprobar visualmente en Cooja que existe un camiño de todos os sensores co seu Border Router.

E3 Como Cooja non representa a veces tódalas conexións mediante liñas de colores, é recomendable comprobar mediante unha ferramenta como as do paquete mosquitto-clients que as motas das que se sospeite que non están a funcionar estean xerando tráfico mediante a suscripción aos topics propios de cada mota como o de /tempI13A. Tamén se pode comprobar se existe unha mensaxe de cada mota en /tempAll. Pódese usar o seguinte comando para facelo:
`mosquitto_sub -t /tempAll`

E4 Comprobar que WireShark está capturando o tráfico das motas. Se se sospeitase que algunha mota non está emitindo tráfico, pódese usar os filtros de WireShark para comprobalo. Por exemplo, con *ipv6.src == a012::212:7403:3:303*, compróbase se existe tráfico capturado da mota 3 da instancia 12.

E5 Existe a posibilidade de que durante a execución da simulación, algunha instancia rompa e se peche. Para evitalo, é recomendable supervisar as instancias de forma periódica. Tamén se pode comprobar ó final da simulación se ocurriu algún imprevisto ó analizar o dataset resultante comprobando que existe tráfico de todas as instancias dende o comezo ata o final.

Capítulo 6

Análise de resultados

Cumprindo coa historia de usuario 6.2., analizouse a captura de tráfico xerada por WireShark durante un total de 11021.75 segundos (3 horas, 3 minutos e 41 segundos). Esta captura conta cunha serie de estadísticas xerais indicadas na táboa 6.1.

Dato	Valor
Paquetes	152031
Tempo de simulación	11021.752
Paquetes por segundo	13.8
Tamaño do paquete medio	59 B
Tamaño total	8906128 B
Velocidade media	808 B/s

Táboa 6.1: Estadísticas xerais da captura do dataset

Se afondamos no uso de protocolos nos paquetes, os datos da táboa 6.2 indican que todos os paquetes usan o protocolo de rede IPv6 e o de transporte UDP, xa que a libraría MQTT-SN usa UDP en lugar de TCP, ó contrario que o MQTT tradicional. Tamén se ve que a parte pesada da transmisión é IPv6, consumindo un total do 68.3% do peso da transmisión, mentres que os datos finais, xunto co protocolo MQTT, usan tan só o 18.06%. Isto significa que dun paquete medio de 59 Bytes, aproximadamente 40.29 son de IPv6, 8.06 son de UDP e uns 10.66 de MQTT e datos, o cal se aproxima os mínimos de cada protocolo, sendo 40 no caso de IPv6 e 8 no caso de UDP.

Protocolo	% paquetes	% Bytes	Bytes
Paquete en bruto	100	100	8906128
IPv6	100	68.28	6081240
UDP	100	13.66	1216248
MQTT-SN e datos	100	18.06	1608640

Táboa 6.2: Estadísticas por protocolo da captura do dataset

Instancia	Dirección sensor	Paquetes	Bytes	P. In	B. In	P. Out	B. Out
1	a001::212:7402:2:202	4482	262880	2205	110266	2277	152614
1	a001::212:7403:3:303	4484	262996	2206	110316	2278	152680
1	a001::212:7404:4:404	4486	263112	2207	110366	2279	152746
1	a001::212:7405:5:505	4478	262648	2203	110166	2275	152482
1	a001::212:7406:6:606	4486	263110	2207	110366	2279	152744
1	a001::212:7407:7:707	4478	262648	2203	110166	2275	152482
1	a001::212:7408:8:808	4486	263112	2207	110366	2279	152746
1	a001::212:7409:9:909	4448	259468	2206	110316	2242	149152
2	a002::212:7402:2:202	4482	262880	2205	110266	2277	152614
2	a002::212:7403:3:303	4484	262996	2206	110316	2278	152680
2	a002::212:7404:4:404	4470	262184	2199	109966	2271	152218
2	a002::212:7405:5:505	4475	262434	2202	110116	2273	152318
2	a002::212:7406:6:606	4482	262880	2205	110266	2277	152614
2	a002::212:7407:7:707	4472	262300	2200	110016	2272	152284
2	a002::212:7408:8:808	4482	262880	2205	110266	2277	152614
2	a002::212:7409:9:909	4448	259468	2206	110316	2242	149152
11	a011::212:7402:2:202	4468	261924	2198	109916	2270	152008
11	a011::212:7403:3:303	4476	262393	2202	110117	2274	152276
11	a011::212:7404:4:404	4470	262040	2199	109966	2271	152074
11	a011::212:7405:5:505	4438	258821	2201	110067	2237	148754
11	a011::212:7406:6:606	4480	262620	2204	110216	2276	152404
11	a011::212:7407:7:707	4477	262374	2203	110166	2274	152208
11	a011::212:7408:8:808	4478	262504	2203	110166	2275	152338
11	a011::212:7409:9:909	4440	258932	2202	110116	2238	148816
11	a011::212:740a:a:a0a	4468	261924	2198	109916	2270	152008
11	a011::212:740b:b:b0b	4474	262272	2201	110066	2273	152206
12	a012::212:7402:2:202	4473	262176	2201	110066	2272	152110
12	a012::212:7403:3:303	4482	262736	2205	110266	2277	152470
12	a012::212:7404:4:404	4474	262272	2201	110066	2273	152206
12	a012::212:7405:5:505	4436	258700	2200	110016	2236	148684
12	a012::212:7406:6:606	4476	262388	2202	110116	2274	152272
12	a012::212:7407:7:707	4482	262736	2205	110266	2277	152470
12	a012::212:7408:8:808	4470	262040	2199	109966	2271	152074
12	a012::212:7409:9:909	4446	259280	2205	110266	2241	149014

Táboa 6.3: Estadísticas por sensor da captura do dataset. P(aquetes) In e B(ytes) In, indican a cantidade de paquetes e bytes recibidos polo sensor mentres que P(aquetes) Out e B(ytes) Out, indican a cantidade de paquetes e bytes enviados.

Na táboa 6.3, vense os datos por conexión entre o sensor e o broker de MQTT. Nela, aparecen todos os sensores agrupados pola instancia na que se están simulando. A media de paquetes por sensor é de 4471.5, sendo 2202.97 recibidos polo sensor e 2268.53 enviados polo sensor, e a media do total transferido por sensor é de 261944.94 Bytes, sendo 110164.59

Bytes recibidos polo sensor e 151780.35 Bytes enviados polo sensor. A proporción de paquetes recibidos:enviados é de 1:1.03, o que fai que case por cada paquete enviado exista un recibido. A proporción de Bytes recibidos:enviados é de 1:1.38, o que significa que a cantidade de datos que envía e próxima, aínda que lixeiramente maior, da cantidade de datos que recibe. Isto deixa claro que o broker MQTT ten que ter un ancho de banda similar tanto en transmisión como en recepción de datos.

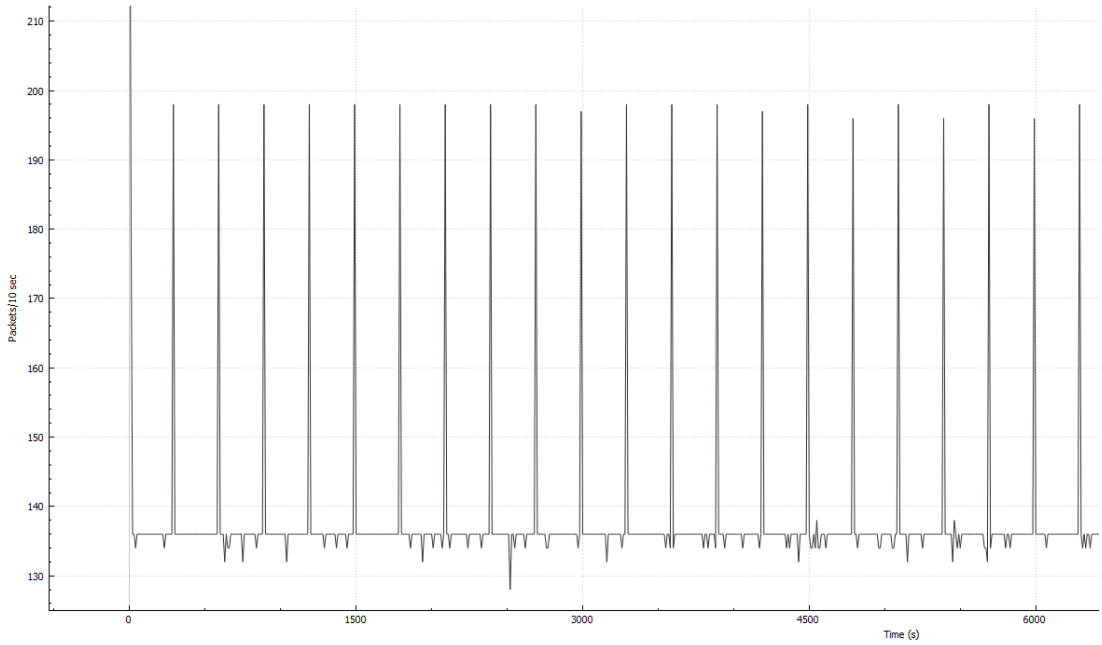


Figura 6.1: Tráfico da captura de WireShark. Os datos están agrupados en intervalos de 10 segundos.

Vendo a figura 6.1, pódese observar o tráfico en forma de paquetes por segundo que capturou WireShark durante a simulación. Ó inicio da simulación existe un pico de 260 paquetes en 10 segundos debido a que as motas están rexistrándose no broker MQTT. Despois, cada 5 minutos existe un pico de 195-198 paquetes debido a que a frecuencia de reporte de cada sensor é de 5 minutos e comezaron todos á vez. Existe un tráfico de 136 paquetes de fondo asociados aos pings do protocolo MQTT.

6.1 Análise de MQTT-SN

MQTT-SN comeza as comunicacións cun comando de tipo *Connect* (Tipo de mensaxe MQTT-SN: 0x4). Esta mensaxe foi capturada 36 veces, unha de cada un dos 34 sensores e 2 mais para os sensores a011::212:7403:3:303 e a011::212:7405:5:505 que, por algunha razón,

pediron conectarse dúas veces usando o mesmo paquete MQTT-SN. Esta mensaxe é acompañada por unha de resposta de tipo *Connect Ack* (0x5), a cal tamén se capturou 36 veces por parte do broker e dirixida a cada un dos sensores.

O seguinte paso e rexistrar os topics por parte dos sensores, o cal faise cunha mensaxe de tipo *Register* (0xA). Enviáronse un total de 102, 3 por cada un dos 34 sensores, sendo un o topic */tempAll*, un topic propio do sensor como */tempR26D* e outro topic tamén por sensor como *Node 1112740A000A0A0A reporting temperature* que indica a ID do sensor. Para estas mensaxes tamén existe o seu correspondente *Register Ack* (0xB), enviado 102 veces polo broker.

A partir deste momento comezan as comunicacións. Por unha banda, os sensores comezan a enviar publicacións con mensaxes de tipo *Publish* (0xC). Enviáronse un total de 2229, o 1.5% do total dos paquetes. Cada sensor envía 2 mensaxes *Publish* (0xC) á vez, unha para */tempAll* e outra para o seu propio topic, repetindo o envío cada 5 minutos. Esta mensaxe non ten o seu correspondente Ack por parte do broker.

Pola outra banda, os sensores envían unha mensaxe de tipo *Ping request* (0x16) cada 5 segundos ó broker, o cal responde á mesma mediante unha mensaxe de tipo *Ping response* (0x17). Capturánronse un total de 74763, o 49.2% do tráfico, de *Ping request* (0x16) e a mesma cifra de *Ping response* (0x17), o que significa que un 98.4% do tráfico son mensaxes de ping para comprobar a conectividade do sensor co broker (ver figura 6.2).

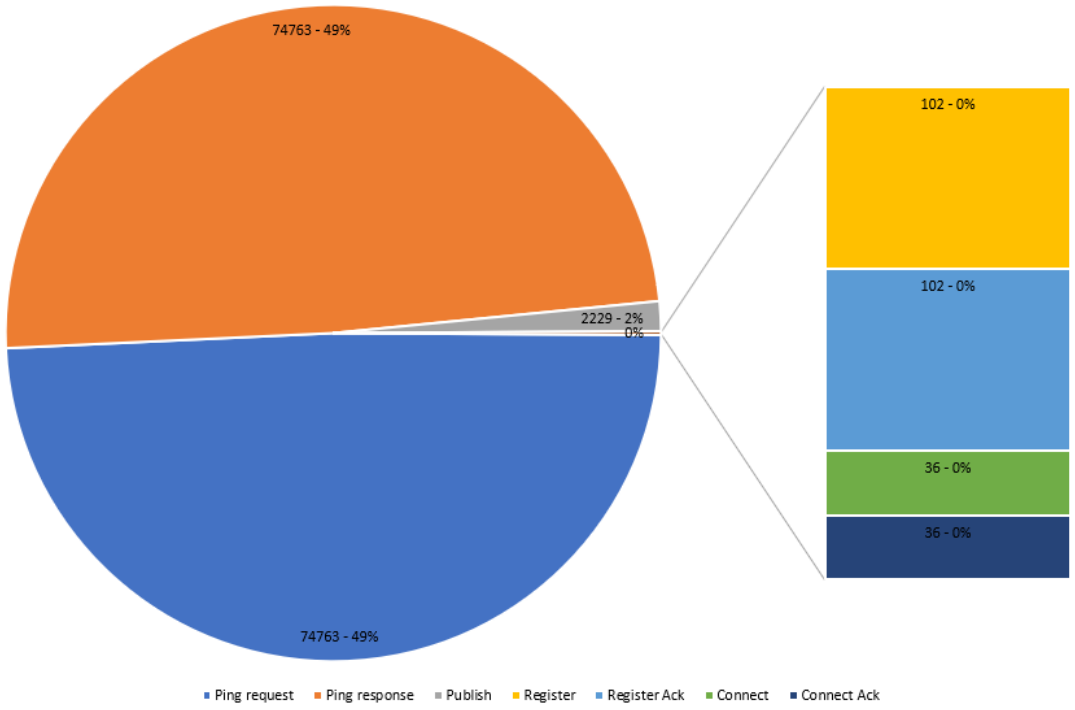


Figura 6.2: Visual das proporcións en cantidade de mensaxes por tipo.

Ó observar esta gráfica chama a atención que a maioría do tráfico que existe en MQTT é sen duda o dos pings e a súa resposta por parte do broker. Un ping xerado por un sensor ten un tamaño de só 66 bytes e a súa resposta é de 50 bytes, ó contrario que unha mensaxe publish, que te un tamaño de entre 96 bytes por parte dos racks e 98 bytes por parte das inrows, debido a precisión decimal destes últimos, sendo case o dobre que os pings. Aínda así, o maior uso do ancho de banda segue sendo por parte dos pings debido ó seu maior número.

Capítulo 7

Conclusións, leccións aprendidas e liñas futuras

7.1 Conclusións e leccións aprendidas

Este proxecto tivo como obxectivo capturar un dataset de temperaturas dunha simulación dun CPD e usando unha arquitectura distribuída de rede para poder escalala se fose necesario. Durante o desenvolvemento do mesmo, traballouse con tecnoloxías como MQTT e a súa variante MQTT-SN, traballaouse cun sistema operativo deseñado para redes de sensores de IoT e cun simulador desas redes, solucionáronse multitude de problemas que se foron atopando polo camiño como a incapacidade por parte de Cooja de traballar con temperaturas ou a característica de ser monothread, o que provocaba que a súas instancias estivesen limitadas, probando e comprobando varias solucións potenciais para os problemas de captura de paquetes de forma unificada por parte de WireShark ou como simular as temperaturas de forma realista dentro dun simulador que non o soportase, etc.

Este proxecto é unha primeira toma de contacto co mundo de IoT e os seus paradigmas e protocolos, ademais de outros protocolos como IPv6 que, aínda que non é exclusivo do mundo de IoT, si que se usa de forma masiva neste ó contrario coas redes de comunicacións xerais que seguen dependendo en gran medida de IPv4. Todo isto resulta nunha experiencia realmente enriquecedora.

Este traballo tamén anima a ser reutilizado para axudar con outros proxectos de características similares como, por exemplo, a simulación dunha rede de comunicacións baseada en motas que actúan como puntos de acceso e enrutadores distribuídos por un monte ou a simulación dunha rede de sensores que reporten múltiples variables dun sistema complexo.

En canto a obxectivos cumplidos, puideronse cumplir todos, aínda que é importante engadir unha serie de notas.

- Aínda que non está escrito como obxectivo, existía un interés en intentar usar exac-

tamente o mesmo código nas motas simuladas que se usaría nas motas reais, algo que acabou sendo imposible debido á incapacidade de Cooja de simular temperaturas, o que provocaba que estas sempre reportasen a mesma temperatura e non existise ningunha forma de alterala. Tampouco foi posible modificar o simulador de forma que o código das motas non tivese que ser distinto do código que se usaría nunha mota real debido a que non se conseguiu que Cooja detectase e entregase a cada mota individual o seu correspondente valor.

- Outro problema que non afectou aos obxectivos principais, pero que non permitiu realizar o traballo como se esperaba é a inestabilidade que teñen os sensores Z1 co protocolo RPL. Isto provocou que non se puidesen chegar a usar na simulació final, como orixinalmente se tiña pensado, tendo que terminar por usar as motas Cooja e Sky. Sen embargo, este contratempo non afectou á calidade do dataset.

7.2 Liñas futuras de traballo

Tras rematar o proxecto, quedaron certas liñas de traballo que poderían ser interesantes para completar nun futuro:

- **Uso de outros sistemas operativos e simuladores**. Aínda que neste traballo usouse o sistema operativo Contiki xunto co seu simulador Cooja, podería ser interesante chegar a facer a simulación noutro sistema operativo como TinyOS e noutro simulador como Avrora. Tamén existe un sucesor do sistema operativo Contiki chamado Contiki-NG, que non foi usado neste proxecto por considerarse aínda inmaduro e sen aínda un importante soporte comunitario.

- **Automatización mediante scripts**. Moitas das tarefas deste traballo realizáronse a man por ser un proxecto cunha simulación relativamente pequena pero, se fose necesario ampliala a centos ou miles de motas, é imperativo realizar un script que sexa capaz de crear o código fonte das motas segundo unha táboa de comportamento das temperaturas e un script que sexa capaz de crear e lanzar as múltiples instancias de Cooja de forma automática.

- **Ampliación do dominio de datos**. Ademáis de para temperaturas, pódense engadir mais datos ó dataset como o de humidade, mediante a modificación do código das motas para recolectar os datos do sensor, ou, se tampouco admitise ese dominio o simulador, mediante a creación de funcións que simulen de forma realista o seu comportamento.

- **Probas de escalado**. Sería recomendable probar esta arquitectura ampliándoa de forma masiva a varias máquinas virtuais executándose en varios servidores físicos e interco-

nectadas entre si, de forma que cada máquina execute media ducia de instancias. Neste escenario poderíase comprobar que tan ben captura os datos wireshark e que tan ben é capaz de soportar a carga o broker de MQTT-SN.

Apéndice A

Script de instalación en Instant Contiki e MQTT-SN

```
#Enrutamento IPv6 (require root)
sudo sysctl -w net.ipv6.conf.all.forwarding=1
#Cambiar o nome do cartafol de Contiki
cd ~
rm -rf contiki
mv contiki-3.0/ contiki/
#Instalar mspsim
cd ~/contiki/tools/
rmdir mspsim
wget https://github.com/contiki-os/mspsim/archive/master.zip
unzip master.zip
rm master.zip
mv mspsim-master/ mspsim/
#Compilar tunslip
make
make tunslip6
#Instalar mqtt-sn
cd ~/contiki/
git clone \
https://github.com/aignacio/mqtt-sn-contiki_example \
mqtt-sn-contiki
cd mqtt-sn-contiki
make all
cd tools
unzip mosquitto.rsmb.zip
cd mosquitto.rsmb/rsmb/src
make clean
make
#Iniciar Cooja
cd ~/contiki/tools/cooja
ant run
```

Apéndice B

Datos da temperatura dos sensores reais

Sensor	Tmin(P)		Tmax(P)
Rack 11 Dianteiro	19(0,42)		20(0,58)
Rack 11 Traseiro	28(0,85)		29(0,15)
Rack 12 Dianteiro	19(0,03)		20(0,97)
Rack 12 Traseiro	27(0,42)		28(0,58)
Rack 14 Dianteiro	20(0,98)		21(0,02)
Rack 14 Traseiro	29(0,30)		30(0,70)
Rack 16 Dianteiro	20(0,94)		21(0,06)
Rack 16 Traseiro	26(0,31)		27(0,69)
Rack 17 Dianteiro	20(0.96)		21(0,04)
Rack 17 Traseiro	27(0,75)		28(0,25)
Rack 21 Dianteiro	19(0,53)		20(0,47)
Rack 21 Traseiro	30(0,98)		31(0,02)
Rack 22 Dianteiro	19(0,12)		20(0,88)
Rack 22 Traseiro	29(0,93)		30(0,07)
Rack 26 Dianteiro	19(0.84)		20(0.16)
Rack 26 Traseiro	33(0.96)		34(0.04)
Rack 27 Dianteiro	19(0.25)		20(0.75)
Rack 27 Traseiro	25(0.34)	26(0.64)	27(0.02)

Táboa B.1: Temperaturas dos racks. Cada columna indica a probabilidade P para unha temperatura Tmin e Tmax. O sensor traseiro do rack 27, ten 3 posibles temperaturas.

Sensor	1	2	3	4	5	6	7	8	9	10	11	12	minR	maxR	mean
I13A	-0,124695	-0,051707	-0,067152	0,050233	0,067330	0,027532	0,058648	0,089960	0,075017	0,044773	-0,044523	-0,125414	-2	2	12,148519
I13B	-0,088623	-0,058666	-0,061827	0,028188	0,045429	0,017699	0,036416	0,086018	0,062167	0,047297	-0,019370	-0,094729	-2	2	16,054442
I13C	-0,077929	-0,055658	-0,035041	0,025161	0,059284	0,031482	0,058076	0,064744	0,047718	0,010074	-0,032885	-0,095026	-1	1	20,001128
I13D	-0,031703	-0,041760	-0,031128	-0,022292	0,008958	0,022895	0,022796	0,023398	0,020165	0,022248	0,014202	-0,007780	-1	0	27,318053
I15A	-0,049889	-0,040119	-0,100823	-0,013323	0,083013	0,020513	0,064515	0,137750	0,053703	0,031361	-0,047734	-0,138969	-3	2	11,912535
I15B	-0,036426	-0,045550	-0,083121	-0,027949	0,049494	-0,007834	0,046681	0,109605	0,070255	0,036060	-0,006613	-0,104601	-2	1	15,602253
I15C	-0,042654	-0,051131	-0,046174	-0,010470	0,044056	0,038308	0,042967	0,083574	0,027389	0,008567	-0,023976	-0,070456	-2	1	19,992394
I15D	-0,016048	-0,010660	-0,027686	-0,027614	-0,013821	0,012041	0,017922	0,026538	0,016926	0,019081	0,005791	-0,002470	-1	1	27,238450
I23A	-0,040120	-0,119215	-0,060810	0,042423	0,061389	-0,002548	0,048371	0,068993	0,036820	0,026978	0,045369	-0,107649	-3	2	11,357323
I23B	-0,048055	-0,108112	-0,065152	-0,004736	0,033698	-0,007465	0,023343	0,067763	0,043468	0,037362	0,071342	-0,043457	-2	2	15,169014
I23C	-0,036189	-0,048258	0,003466	0,029328	0,027532	0,003035	0,013300	0,037171	0,022000	0,011871	0,014098	-0,077353	-1	1	20,004788
I23D	-0,015088	-0,021769	-0,021554	-0,026367	-0,020979	0,006679	0,004248	0,004177	0,015228	0,038216	0,032038	0,005170	-1	0	26,326901
I25A	-0,076609	-0,071509	-0,114253	0,000258	0,084598	0,006724	0,092775	0,138189	0,076767	0,036250	-0,035877	-0,137313	-3	2	11,471368
I25B	-0,053043	-0,076965	-0,085227	-0,022655	0,051339	0,009744	0,070327	0,097819	0,058595	0,038624	-0,002253	-0,086305	-3	1	15,569252
I25C	-0,005159	-0,013636	-0,039642	-0,005447	0,014812	0,023074	0,026866	0,068835	0,019051	0,009065	-0,018090	-0,079728	-2	1	19,995345
I25D	-0,015088	-0,021769	-0,021554	-0,026367	-0,020979	0,006679	0,004248	0,004177	0,015228	0,038216	0,032038	0,005170	0	0	26,841889

Táboa B.2: Temperaturas dos inrows. Para cada sensor hai 12 períodos de 5 minutos, un mínimo (minR) e máximo (maxR) entre os que calcular un valor aleatorio e unha base (mean). Sumar os tres valores produce a temperatura actual, tal e como se indica en 5.2.

Apéndice C

Script para xerar a gráfica da función de temperaturas en R

O seguinte script xera en R a gráfica da imaxe 5.1.

```
n = 100000
b = floor(runif(n)*3-1)
c = floor(runif(n)*50+1)
d = integer(n)
d = ifelse(b==-1,-2/c,d)
d = ifelse(b==1,2/c,d)
as.data.frame(table(d))
f = round(d,digits=1) #digits=decimais de redondeo
hist(f,breaks=40,prob=TRUE) #Engadir en breaks tantos ceros
#como decimais de redondeo. Exemplo: 3 decimais, breaks=4000
```

Apéndice D

Glosario de acrónimos

WSN *Wireless Sensor Networks*

RTOS *Real Time Operating System*

DODAG *Destination Oriented Directed Acyclic Graph.*

LR-WPAN *Low-Rate Wireless Personal Area Network.*

MQTT *Message Queuing Telemetry Transport.*

DIO *DODAG Information Object.*

DIS *DODAG Information Solicitation.*

DAO *Destination Advertisement Object.*

uIP *micro IP.*

6LoWPAN *IPv6 over Low -Power Wireless Personal Area Networks.*

RPL *Routing Protocol for Low-Power and Lossy Networks.*

CPD *Centro de Procesamento de Datos.*

CITIC *Centro de Investigación en Tecnoloxías da Información e das Comunicacións.*

API *Application Programming Interface.*

QoS *Quality of Service.*

FPU *Floating Point Unit.*

Apéndice E

Glosario de termos

Open-source Dise dunha peza software cuxo código fonte está dispoñible de forma pública.

Microcontrolador Un circuito integrado que é programable. Executa as instrucións gravadas na súa memoria.

Kernel Peza de software fundamental que forma un SO.

Kernel monolítico Un tipo de kernel onde todo excepto a aplicación final execútase en modo kernel.

Coffe Filesystem Un sistema de ficheiros propio de Contiki deseñado para ser moi lixeiro.

Rime Stack de rede lixeiro usado como alternativa ó stack IP.

Mota Dispositivo sen fíos con capacidades sensoriales, de procesamento e memoria, conectado a unha rede xunto a outras motas. Tamén chamado sensor.

Broker de MQTT Peza de software contra a cal se realizan as subscripcións a temas e se envían as publicacións. É a encargada de retransmitir estas publicacións ós seus subscriptores.

WireShark Ferramenta utilizada para analizar redes e capturar os paquetes transmitidos por elas.

Overhead Exceso de uso en recursos necesario á hora de realizar unha tarefa.

Arquivo pcap Arquivo resultante da captura dos paquetes da rede.

Rack Armario onde se están colocados os aparellos de tecnoloxía da información nos CPD.

Inrow Aparello que se coloca no CPD entre dous racks e enfría o aire que pasa a través del mediante convección cun fluxo de auga fría.

Keep-alive Un tipo de mensaxe que se envía cando o orixe quere comprobar se o enlace de comunicación có destinatario segue activo.

Gateway Dispositivo que actúa como interface de conexión entre dous protocolos ou outros dispositivos.

Switch Dispositivo que interconecta enlaces de rede a nivel físico.

Ping Un tipo de mensaxes que se envían para detectar a dispoñibilidade e tempo de resposta de outros elementos nunha rede.

Superusuario/root Usuario especial en sisemas Linux que permite facer tarefas administrativas e ten todos os permisos habilitados.

Bibliografía

[1] "Osrtos - lista de rtos," https://www.osrtos.com/, Último acceso: 2019-07-11.

[2] "Enquisa sobre os so de iot mais usados [2018]," https://www.itprotoday.com/iot/survey-shows-linux-top-operating-system-internet-things-devices, Último acceso: 2019-07-11.

[3] "Requisitos hardware de linux," https://docstore.mik.ua/orelly/linux/run/ch01_09.htm, Último acceso: 2019-07-11.

[4] "Requisitos de windows iot," https://docs.microsoft.com/en-us/windows-hardware/design/minimum/minimum-hardware-requirements-overview, Último acceso: 2019-07-11.

[5] "Arrow. iot operating systems," https://www.arrow.com/en/research-and-events/articles/iot-operating-systems, Último acceso: 2019-07-11.

[6] "Amazon freertos," https://aws.amazon.com/es/freertos/, Último acceso: 2019-07-11.

[7] "Embd os," https://www.mbed.com/en/platform/mbed-os/, Último acceso: 2019-07-11.

[8] "Comparativa entre contiki e tinyos," https://pdfs.semanticscholar.org/57a6/26f9be7b446713ca4448854b2c8369bb859f.pdf, Último acceso: 2019-07-11.

[9] "Riot os: Towards an os for the internet of things," https://riot-os.org/docs/riot-infocom2013-abstract.pdf, Último acceso: 2019-07-11.

[10] "Repositorio en github de contiki," https://github.com/contiki-os/contiki, Último acceso: 2019-07-11.

[11] "Wireless sensor network simulators: A survey and comparisons," https://www.researchgate.net/publication/228748210_Wireless_Sensor_Network_Simulators_A_Survey_and_Comparisons, Último acceso: 2019-07-12.

[12] "Protothreads," http://dunkels.com/adam/pt/, Último acceso: 2019-07-11.

[13] "Contiki os - an operating system for iot," https://www.linkedin.com/pulse/contiki-os-operating-system-iot-t-s-pradeep-kumar/, Último acceso: 2019-06-04.

[14] "Licencia de contiki," http://www.contiki-os.org/license.html, Último acceso: 2019-06-01.

[15] "Páxina principal de contiki," http://www.contiki-os.org/index.html#why, Último acceso: 2019-06-01.

[16] "Simulador cooja," http://www.contiki-os.org/start.html#start-cooja, Último acceso: 2019-06-01.

[17] "Contiki motes on cooja," https://github.com/contiki-os/contiki/wiki/An-Introduction-to-Cooja#Contiki_level_the_Contiki_Mote_Type, Último acceso: 2019-06-01.

[18] "Contiki motes on cooja," http://www.contiki-os.org/hardware.html, Último acceso: 2019-06-01.

[19] "Using cooja for wsn simulations: Some new uses andlimits," https://hal.inria.fr/hal-01240986v2/document, Último acceso: 2019-07-15.

[20] "Sky mote," https://github.com/contiki-ng/contiki-ng/wiki/Platform-sky, Último acceso: 2019-06-01.

[21] "Z1 mote," https://github.com/contiki-os/contiki/wiki/Zolertia-z1-motes, Último acceso: 2019-06-01.

[22] "Páxina web de phidgets," https://www.phidgets.com/, Último acceso: 2019-07-13.

[23] "Stack uip," https://github.com/adamdunkels/uip, Último acceso: 2019-06-01.

[24] "Stack contiki," https://anrg.usc.edu/contiki/index.php/Network_Stack, Último acceso: 2019-06-01.

[25] "Networking contiki," https://en.wikipedia.org/wiki/Contiki#Networking, Último acceso: 2019-06-01.

[26] "Mensaxes de rpl," https://tools.ietf.org/html/rfc6550#section-6, Último acceso: 2019-06-03.

[27] "Páxina web de mqtt," https://mqtt.org/, Último acceso: 2019-06-01.

[28] "Cisco's ppdioo network cycle," http://www.ciscopress.com/articles/article.asp?p=1697888&seqNum=2, Último acceso: 2019-07-18.

[29] "Inrow apc acrc103," https://www.apc.com/shop/my/en/products/In-Row-RC-Chilled-Water-200-240V-50-60-Hz-IEC-309-16/P-ACRC103?isCurrentSite=true, Último acceso: 2019-07-22.

[30] "Sensor de temperatura apc ap9335t," https://www.apc.com/shop/es/es/products/APC-Temperature-Sensor/P-AP9335T, Último acceso: 2019-07-22.

[31] "Sensor de humidade e temperatura apc ap9335th," https://www.apc.com/shop/es/es/products/Sensor-de-temperatura-y-humedad-de-APC/P-AP9335TH, Último acceso: 2019-07-22.

[32] "Monitor de sensores apc nbrk0551," https://www.apc.com/shop/my/en/products/NetBotz-Rack-Monitor-550-with-120-240V-Power-Supply-/P-NBRK0551?isCurrentSite=true, Último acceso: 2019-07-22.

[33] "Switch de rede 24x 10/100 mbit apc ap9224110," https://www.apc.com/shop/in/en/products/APC-24-Port-10-100-Ethernet-Switch/P-AP9224110, Último acceso: 2019-07-22.

[34] "Aplicación de supervisión do cpd apc ap9465," https://www.apc.com/shop/es/es/products/StruxureWare-Data-Center-Expert-Basic/P-AP9465, Último acceso: 2019-07-22.

[35] "Proxmox," https://www.proxmox.com/en/, Último acceso: 2019-07-25.

[36] "Vmware esxi," https://www.vmware.com/es/products/esxi-and-esx.html, Último acceso: 2019-07-25.

[37] "Páxina de descarga de instant contiki," https://sourceforge.net/projects/contiki/files/Instant%20Contiki/, Último acceso: 2019-06-04.

[38] "Cooja non simula temperaturas," https://stackoverflow.com/questions/28013883/simulating-different-temperatures-on-cooja-contiki, Último acceso: 2019-06-04.

[39] "How to read temperature, humidity and light," https://stackoverflow.com/questions/21047965/how-to-read-temperature-humidity-and-light-measures-with-contiki-os, Último acceso: 2019-08-13.

[40] "Cooja simulator speed," https://sourceforge.net/p/contiki/mailman/message/29951329/, Último acceso: 2019-07-27.

[41] "Mqtt-sn en contiki," https://blog.aignacio.com/2017/06/25/6lowpan-mqtt-and-contiki-os/, Último acceso: 2019-06-012.

[42] "Mqtt-sn sobre porto serie," https://stackoverflow.com/questions/27560549/when-mqtt-sn-should-be-used-how-is-it-different-from-mqtt, Último acceso: 2019-06-12.

[43] "Broker de mqtt-sn," https://mqtt.org/tag/rsmb, Último acceso: 2019-06-12.

[44] "Repositorio en github da libraría de mqtt-sn para contiki," https://github.com/aignacio/mqtt-sn-contiki_example, Último acceso: 2019-06-12.

[45] "Licencia de apache 2.0," https://www.apache.org/licenses/LICENSE-2.0, Último acceso: 2019-06-12.

[46] "Floating point unit - contiki," https://sourceforge.net/p/contiki/mailman/message/25920389/, Último acceso: 2019-07-30.