Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Estudio de implementaciones del Protocolo CoAP

Autor: Antonio Millán Yanes

Tutor: Antonio Estepa Alonso

Dpto. Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Estudio de implementaciones del Protocolo CoAP

Autor:
Antonio Millán Yanes

Tutor:
Antonio Estepa Alonso
Profesor Titular

Dpto. Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Proyecto Fin de Carrera: Estudio de implementaciones del Protocolo CoAP

Autor: Antonio Millán Yanes
Tutor: Antonio Estepa Alonso

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

La entrega y evaluación de este trabajo conlleva el final de una etapa en mi vida. Esta etapa ha visto pasar los mejores momentos de mi vida, pero también ha conllevado grandes dificultades en ella. En esos momentos más duros son en los que las personas que importan han estado a mi lado. Agradezco, por ello, en primer lugar, a las personas que desde siempre han estado conmigo: agradezco a mi padre, Antonio Millán por su firmeza a la hora de apoyarme en los estudios y su plena confianza durante todo el proceso, así como su capacidad de hacerme desconectar en mis momentos estrés, a mi madre, Maria Jesus Yanes , por ser la persona que siempre aporta felicidad y tranquilidad a mi vida sin importarle el coste, a mi hermana, Maria Millán por ser la persona de confianza que me ha enseñado a hacer lo que debo y, a veces, lo que no debo, que me ha dado paciencia y que más confianza Ha demostrado en mí desde el primer momento. Mis padrinos Antonio y Amalia y mi abuela Dionisia , que siempre han estado ahí incluso cuando yo no he podido estar con ellos todo lo que quisieran.

También he de agradecer a todos aquellos amigos que me han hecho disfrutar de esta etapa como nadie, que me han apoyado en los estudios y han demostrado su total fidelidad. En especial he de mencionar a Javier Echevarría, a Nacho Barrio, a Jaime Guerra, a Gonzalo Pérez-Marín, Leonor Gonzalo e Irene Martel.

A mis compañeros de carrera, que han sufrido conmigo y han celebrado ese 5 como nadie a base de cervezas, Lucas Roldán, Pablo Ruiz y Jorge Mora. En especial, agradecer a Carmen LLamas su paciencia y su ayuda en, prácticamente, todos los aspectos de la carrera, así como su siempre amable compañía en el trabajo.

A Aurora Lopez, que puede ir incluida en cualquiera de las anteriores descripciones, familia, amiga y compañera de estudios. Me ha tolerado en todos los momentos y me ha apoyado siempre en todas mis decisiones con la mayor paciencia que se puede tener.
Por último, académicamente quiero agradecer a Rafael Herrara por sus sabios consejos en mis elecciones dentro de la universidad, a Juan Manuel Vozmediano y Javier Muñoz que han conseguido que me enamore de mis especialidad. Y, en último lugar, a Antonio Estepa que ha depositado confianza en mí desde el primer momento, tanto en el grado como en el máster, y me ha aportado sabios consejos para mi futuro.

Antonio Millán Yanes
Estudiante del grado en Ingeniería de Telecomunicación

Sevilla, 2019

Resumen

A lo largo de este documento se realiza un estudio de las implementaciones más destacadas de código libre del protocolo CoAP. El estudio previo realizado alcanza las funcionalidades de CoAP implementadas y las características extras de las implementaciones. Además, se realizan dos pruebas que nos permiten obtener valores relacionados con el tiempo de iniciación del cliente, el tiempo de respuesta del servidor y el uso de CPU y de RAM en condiciones desfavorables.

La primera prueba, prueba de compatibilidad, realiza una petición entre cada uno de los clientes con cada uno de los servidores. La segunda, prueba de estrés, reenvía un alto número de peticiones a distinta velocidad contra cada uno de los servidores.

Con este estudio se concluyen las ventajas y desventajas de cada una de las implementaciones. Qué implementación es mejor para nodos y redes más restringidos, qué implementación soporta un flujo de tráfico mayor o qué implementación permite un uso más sencillo de cara a realizar pruebas.

Abstract

Throughout this document a study of the most outstanding implementations of open source CoAP protocol is carried out. The previous study reaches the implemented CoAP functionalities and the extra features of the implementations. In addition, two tests are performed that allow us to obtain values related to the time of initiation of the client, the response time of the server and the use of CPU and RAM in unfavorable conditions.

The first test, compatibility test, makes a request between each of the clients with each of the servers. The second, stress test, resends a high number of requests at different speeds against each of the servers.

This study concludes the advantages and disadvantages of each of the implementations. Which implementation is best for more restricted nodes and networks, which implementation supports a higher traffic flow, or which implementation allows a simpler use for testing.
-Traducido con www.DeepL.com/Translator-

Índice

1 Introducción

The Internet of Things, commonly abbreviated as IoT, refers to the connection of devices (other than typical fare such as computers and smartphones) to the Internet. Cars, kitchen appliances, and even heart monitors can all be connected through the IoT. And as the Internet of Things grows in the next few years, more devices will join that list.

Andrew Meola, 2018

En la actualidad, el concepto de internet de las cosas, más conocido por su acrónimo en inglés, IoT, está causando un gran impacto. Se trata de la interconexión de objetos ordinarios a través de internet con el fin de poder gestionarlos y controlarlos con facilidad.

Para conseguir esta relación es necesario que desde el 'objeto' más grande hasta el más pequeño puedan comunicarse. Esto implica una adaptación a las limitaciones que pueda tener un dispositivo de tamaño pequeño. Limitaciones como la duración de la batería o las restricciones en recursos, como la memoria o el procesador.

Constrained Application Protocol, CoAP[1], es un protocolo de red creado para el uso en nodos limitados en redes limitadas. Estos nodos, por definición, están muy restringidos en varios aspectos, como, por ejemplo, memoria RAM y procesador. Las redes donde se utilizan, por lo general, son 6LoWPANs[2].

CoAP tiene un gran parecido a HTTP[3] y funciona con conceptos web como, entre otros, las URI. Sin embargo, al estar diseñado para machine to machine, M2M, contiene cambios en el formato de mensaje. El protocolo permite la comunicación con HTTP a través de un proxy, entre otras prestaciones.

El despegue definitivo del Internet de las Cosas
Las empresas españolas se enfrentan al reto de aprovechar las oportunidades que brinda el Internet de las Cosas (IoT por sus siglas en inglés), es decir, un entorno en el que los objetos pueden conectarse a Internet para recoger, enviar y recibir datos. En el ámbito empresarial, los beneficios se han centrado principalmente en una mayor productividad, automatización de procesos y reducción de costes de mantenimiento.
Expansión, 20 de agosto de 2019

A día de hoy el uso de nodos IoT se extiende a lo largo de todos los sectores. [4]. Unos ejemplos del artículo anterior son: en la industria automovilística, SEAT de la fábrica de Martorell, en el entorno rural, la bodega vallisoletana Emilio Moro o incluso en las 'Smart Homes', una gama de colchones de Pikolin . Cada uno de estos usos tiene restricciones distintas. En algún caso es necesario solo el uso de un servidor CoAP con las prestaciones mínimas. En otros casos es de mayor interés reducir el tiempo de respuesta del servidor independientemente del tamaño de la implementación y el uso del procesador. Por último también podemos encontrar casos en los que el cliente se tenga que levantar con frecuencia y el tiempo deificación del cliente es crítico. Es por esta razón que se realiza un estudio de las implementaciones más destacadas de código libre que permita al usuario elegir la más cómoda.

Para este problema existen otros estudios que nos dan una comparación de las implementaciones, la intención de este trabajo es ampliar las características de estos estudios y realizar un proyecto actual ya que existen nuevas modificaciones en todas las implementaciones.[5]

Este trabajo también definirá detalladamente el procedimiento seguido y las herramientas utilizadas para que pueda ser reproducido por el lector con estas implementaciones o con cualquier tipo de implementación que permita la arquitectura de cliente y servidor.

2 Protocolo CoAP

The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use with constrained nodes and constrained networks in the Internet of Things. The protocol is designed for machine-to-machine (M2M) applications such as smart energy and building automation.

Pagina web oficial, 2014

El protocolo CoAP fue estandarizado por el IETF [1] en Junio de 2014 en el estándar RFC7252. CoAP es un protocolo de red creado para dispositivos limitados en recursos, normalmente, debido al tamaño de estos.

La arquitectura del protocolo CoAP incluye conceptos básicos de Web como los tipos de medio o las URIs. Además, está basada en REST[6] para la comunicación entre nodos.

Pese al gran parecido con HTTP[7], CoAP no se ha creado únicamente para comprimir HTTP. Se ha diseñado como un protocolo de web genérico para entornos restringidos, basado en el ahorro de energía, la automatización y aplicaciones para M2M. Posee mejoras respecto a HTTP en este entorno, gracias al uso de mensajes asíncronos, descubrimiento de otros nodos y difusión.

En la Figura 2.1 se muestran dos pilas de protocolos, una para HTTP y otra para CoAP. Un cambio de gran importancia es la capa de transporte. El uso de UDP lo hace más eficiente, no obstante, es menos fiable que TCP[8]. La división en dos subcapas del protocolo CoAP acompañado de los distintos tipos de mensaje añaden un mecanismo de transferencia fiable que permite prescindir del servicio de TCP.

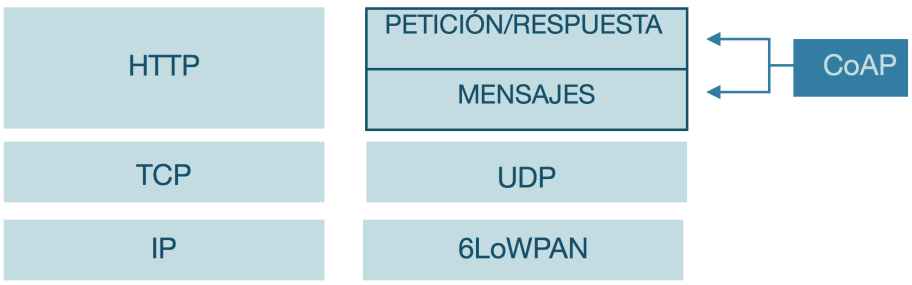

Figura 2.1 Pilas de protocolos; izquierda HTTP, derecha CoAP.

2.1 Formato del mensaje

El protocolo CoAP está formado por una cabecera fija de 4 bytes, seguida de un Token de tamaño máximo de 8 bytes. Después existe una secuencia de ceros o las opciones en formato Type-Length-Value (TLV) y el

Payload. Se muestra la estructura en la Figura 2.2.

Figura 2.2 Formato del mensaje.

- *Version (Ver)*, define la versión del protocolo.
- *Type (T)*, el tipo de mensaje, puede tomar cuatro valores: Confimable, Non-confirmable, Acknowledgement y Reset. Se explicará con detenimiento en el apartado tipos de mensaje.
- *Token Length (TKL)*, longitud del Token, los bytes 9-15 están reservados y no deben usarse.
- *Code*, este campo se asemeja al campo Status Code del protocolo HTTP. Indica, por ejemplo, el 2 como correcto, el 4 como error de la respuesta cliente y el 5 como error del servidor. Un ejemplo de mensaje correctamente enviado sería '2.05' y un mensaje de error típico, el mensaje de no encontrado sería '4.04'.
- *Message ID*, permite evitar duplicidad y facilitar mensajes del tipo Acknowledgement y Reset.
- *Payload*, carga útil del mensaje.

2.2 Tipos de mensaje

CoAP se puede subdividir en dos subcapas como se ha mostrado en la Figura 2.1, la capa inferior es la que permite ofrecer fiabilidad en intercambio de mensajes a través de UDP[9] y la capa superior la que se encarga de los tipos de mensaje REST. El protocolo proporciona cuatro tipos de mensajes: Confirmable (CON), Non-confirmable (NON), Acknowledgement (ACK), Reset (RST).

El tipo de mensaje utilizado para conseguir un mecanismo seguro es Confirmable. Este método consta de temporizador y backoff. Tras la transmisión de un mensaje CON, se espera un ACK. Existen tres situaciones posibles con este tipo de mensaje representadas en la figura 2.3.

1. La respuesta a la petición del mensaje CON se envía directamente en el ACK con el mismo ID que la petición. Este caso ocurre cuando el servidor dispone de la respuesta en el momento de la petición. El tipo de respuesta se denomina Piggybacked.
2. La respuesta se envía más adelante en otro mensaje del tipo CON y otro ID distinto. Para evitar que se retransmita constantemente la petición del cliente, el servidor responde con un ACK vacío a la petición, hasta que disponga de la información.
3. El servidor no dispone de la información, entonces responde con un mensaje de tipo RST.

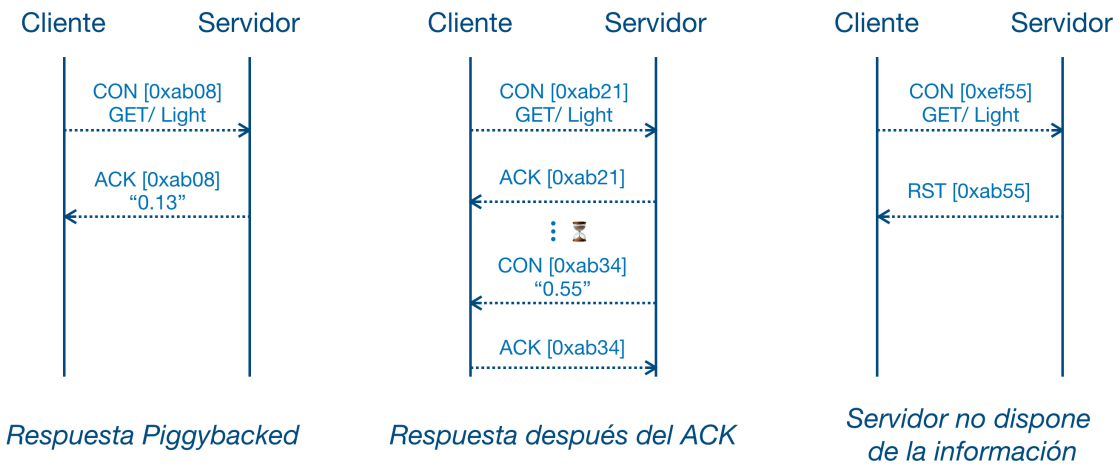

Figura 2.3 Situaciones en mensajes del tipo Confirmable.

Los menajes de tipo Non-Confirmable no esperan un asentimiento del servidor. La información que se ha pedido será enviada en otro mensaje del tipo NON. Este modelo se utiliza en mensajes que no necesitan ser fiables. Existe también el caso de que el servidor no pueda responder la información y responda con un mensaje RST. Un ejemplo se muestra en la figura 2.4.

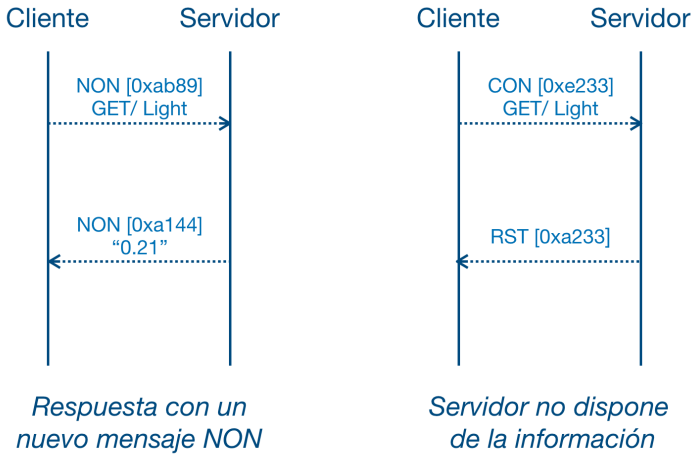

Figura 2.4 Situaciones en mensajes del tipo Non-Confirmable.

Por otro lado, en la capa de petición/respuesta es donde encontramos el uso de de la arquitectura Representational State Transfer (REST) para el intercambio de información. REST utiliza Universal Resource Identifier (URI) para interpretar las peticiones y hace uso de las operaciones GET, POST, PUT y DELETE.

2.3 Funciones del protocolo

Independientemente de las ventajas que nos ofrecen los tipos de mensaje nombrados en el anterior punto, el protocolo ofrece varias funciones que serán estudiadas y que son de gran interés.

2.3.1 Envío de bloques

De acuerdo con las limitaciones de este tipo de redes y dispositivos, este protocolo aporta algunas soluciones. En este caso, para la transferencia de un número elevado de bytes, CoAP define en la RFC 7959 de IETF la

función *Block-Wise Transfers*. Para conseguir estos envíos el protocolo define la división de un bloque de información en fragmentos pequeños. Las principales características son:

- Bloques de un tamaño definido entre 16 bytes y 1 kilobyte.
- Todos los bloques siguen la misma arquitectura de mensaje, CON o NON.
- El tamaño de los bloques será negociado entre las entidades CoAP.
- Las capas superiores no pueden modificar estos parámetros, la aplicación recibe la información directamente ensamblada.

Al igual que con los tipos de mensaje, existen diferentes diagramas dependiendo de la negociación entre los dispositivos.En la figura 2.5 se muestra el caso más común:

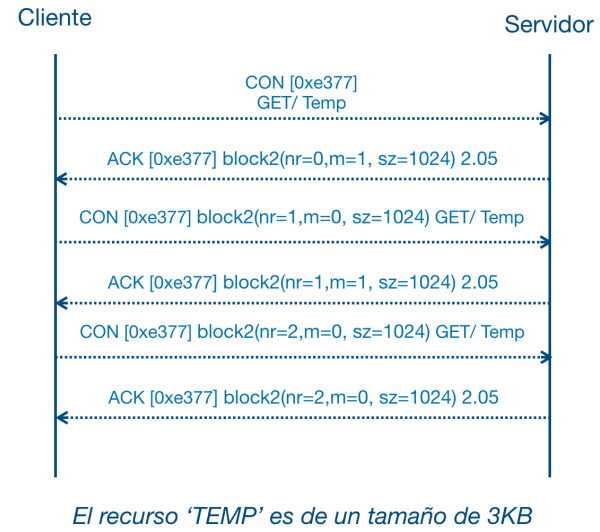

Figura 2.5 Block-wise paso de mensajes.

2.3.2 Observación de recursos

REST permite a un cliente CoAP acceder a los recursos que el servidor ofrezca, no obstante, si un sujeto desea recibir el valor de un recurso cuando éste varíe es necesario el uso de la observación de recursos. Esta función esta definida en la RFC 7641 de IETF[10]. La referencia define en este escenario un sujeto, un observador, el mensaje de registro y los mensajes de notificación. Un ejemplo básico sería el mostrado a continuación en la figura 2.6.

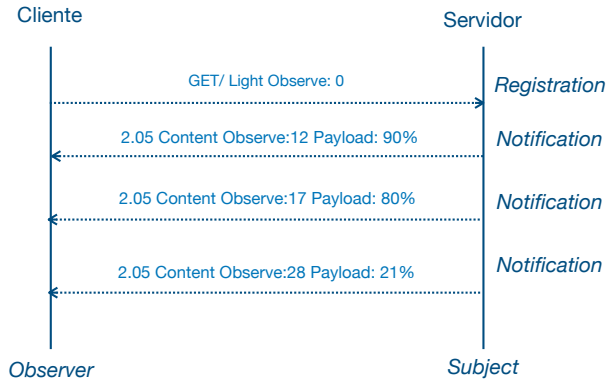

Figura 2.6 Observación de recursos.

2.3.3 Descubrimiento de recursos

Para el uso del protocolo en entornos Machine-to-Machine es de gran importancia esta funcionalidad. El descubrimiento de recursos nos permite conocer los recursos que un servidor es capaz de ofrecer en ese instante. Es tan sencillo como el envío de una petición */well-known* por parte del cliente. Figura2.7.

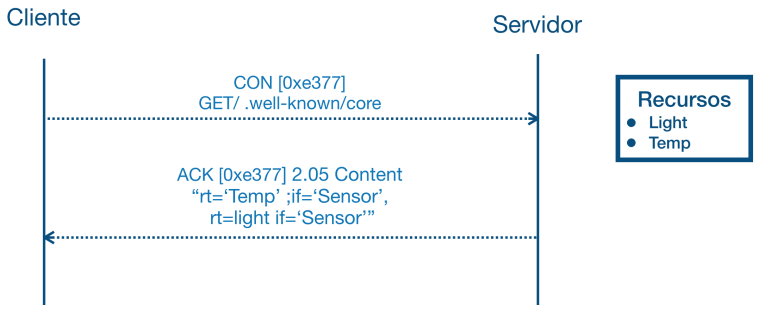

Figura 2.7 Descubrimiento de recursos.

2.3.4 Mapeo CoAP-HTTP

Esta función es utilizada en varios escenarios, pero destacan dos: [11]implementar una interfaz web para usarla sobre este protocolo e implementar un proxy HTTP-CoaP.
CoAP implementa algunas funciones de http como se ha nombrado en los puntos anteriores, esto permite hacer un mapeo directo entre los dos protocolos en la mayoría de los casos.
El mapeo de CoAP a HTTP es más sencillo, se realiza una conversión directa de los método y las opciones a los correspondientes de HTTP. No obstante, la conversión de HTTP a CoAP puede resultar más tediosa ya que existen métodos, opciones y content-type que no implementa CoAP. En estos casos, cuando falla el mapeo de HTTP, hay dos opciones, o despreciarla de forma silenciosa o lanzar un error 502 (Bad Gateway).

2.4 Usos del protocolo

En la actualidad existen nodos de IoT en casi todos los sectores, desde la agricultura a la sanidad[4]. Este protocolo ha sido utilizado entre otros escenarios en los siguientes:

- Red de sensores sanitarios[12], se han probado para medir de forma fiable las medidas de salud vital del paciente como el ritmo cardíaco y la saturación de oxígeno en sangre. El doctor en estos casos puede visualizar a tiempo real en el servidor los datos del paciente. Estos nodos no procesan la información, la envían directamente al servidor que posteriormente calcula y muestra.
- Sustituto a los protocolos de movilidad como IP Movil en los entornos de IoT. Los resultados de un estudio[13] muestran que la estructura propuesta con el protocolo CoAP es mejor que Ip Movil en términos de latencia de transferencia y pérdida de paquetes.
- Fiware[14] es una plataforma impulsada por la Unión Europea para el desarrollo de aplicaciones IoT. Esta plataforma intenta proporcionar de forma libre una arquitectura y un conjunto de especificaciones. Una de las principales vertientes de este proyecto son las Smart Cities[15]. Este proyecto utiliza a nivel de aplicación el protocolo CoAP. A día de hoy esta plataforma es utilizada en más de 1000 startups.
- Protocolo de gestión de IoT. CoAP se utiliza como base en LWM2M, un estándar de Open Mobile Alliance que proporciona diversos servicios para machine to machine. A día de hoy consta de una implementación para contiki. LWM2M utiliza los métodos simples de CoAP así como GET y PUT[16].

  Existe otro estándar que todavía no ha terminado de definirse denominado CoAP Management Interface, CoMI.[17]

  Un ejemplo son los módulos fabricados por la empresa Telit que pertenecen a la familia de módulos IoT y son dispositivos LTE. Este módulo utiliza la funcionalidad de Block-Wise del protocolo CoAP contra un servidor de versiones para la actualización del mismo. Este tipo de módulos se utilizan en aplicaciones de Smart Cities como podrían ser contadores de gas y de luz inteligentes.

3 Estudio de las implementaciones

CoAP is simple enough to implement from scratch for a simple application. For applications where that is not desirable, generic implementations are becoming available for a variety of platforms.

Pagina web oficial, 2014

3.1 Estudio previo

La documentación oficial del protocolo ofrece información resumida de las implementaciones disponibles. Las implementaciones de interés para este estudio no incluirán las implementaciones para aplicaciones móviles, las implementaciones comerciales, y las implementaciones para navegadores. Estas últimas podrán ser utilizadas para la comprobación previa del correcto funcionamiento de los servidores debido a su uso trivial.

Tras un estudio a fondo de las implementaciones de mayor interés, se ha creado la siguiente tabla resumen. En ella encontramos los campos de mayor interés para las pruebas. Además se incluyen campos sobre el proyecto cómo el número de colaboradores y la fecha de las últimas actualizaciones. Estos últimos campos nos permiten conocer la vida del proyecto, dejando así de lado aquellos proyectos muertos que no mantienen soporte.

3.2 Implementaciones elegidas

- **Libcoap**[18]: Implementación en C de CoAP, está diseñada para dispositivos embebidos. Sigue la especificación que lo define, RFC 7252, tanto en la implementación del servidor como en la del cliente. Además, esta implementación soporta todas las funcionalidades nombradas anteriormente. También soporta una fácil implementación junto a DTLS.
- **LibNyoci**[19]: LibNyoci es una implementación que parte de la antigua implementación SMCP. SMCP aparece en el articulo xx debido a su antigüedad. Esta implementación también se desarrolla para el lenguaje C y también la permiten dispositivos con RAM de tamaños desde kilobytes hasta megabytes. Se implementa la parte del servidor y del cliente y algunas de las funcionalidades definidas en la RFC 7252. El soporte de DTLS en este caso es todavía experimental.
- **Microcoap**[20]: Es una implementación limitada, únicamente del servidor de CoAP, escrita en C. No implementa cliente, y solo soporta la funcionalidad de Piggybacked ACK. Tiene como objetivo pequeños microcontroladores Lleva sin actualizarse desde Febrero de 2015.
- **Gen-coap**[21]: Implementación en Erlang de CoAP. Soporta tanto servidor como cliente. La función de proxy no está implementada todavía, pero está en desarrollo.

- **Node-coap**[22]: Implementación en JavaScript de la RFC 7252. Implementa tanto el cliente como el servidor, y las funcionalidades nombradas anteriormente 'Block-wise'y Óbserve Mode'. También implementa Proxy para nodos finales de CoAP, no así proxy HTTP-CoAP.
- **Aiocoap**[23]: Implementación que sigue la RFC 7252. Esta implementación es para python 3 e implementa tanto servidor, como cliente como proxy así como algunas de las funcionalidades definidas anteriormente. Se muestran en la tabla las funcionalidades que define.

3.2.1 Implementaciones descartadas

Tras el estudio previo de las implementaciones más conocidas del protocolo CoAP, se hizo la selección de las anteriores implementaciones. No obstante, se escogieron a su vez tres implementaciones más que fueron descartadas en el periodo de instlación:

- Copper[24]: Implementación basada en una extensión del navegador Firefox. La implementación al comienzo del estudio estaba disponible y superaba las primeras pruebas de compatibilidad, no obstante, la versión de Firefox ascendió y a partir de las versión 60 dejó de estar disponible la implementación en la tienda de extensiones y dejó de tener soporte.
- Lobaro Coap[25]: Implementación seleccionada en un primer lugar por sus características y descartada por falta de documentación. La única información disponible era para la instalación del mismo en un modulo ESP8266 de Arduino.
- CoaPy[26]: Implementación descartada al rechazar las primeras pruebas de compatibilidad y tener la última actualización del repositorio 31 de octubre de 2013.

3.3 Funcionalidad

Las funcionalidades que pueden ofrecer cada una de las implementaciones son las mostradas en el apartado *2.3* del documento. En la tabla 3.1 se muestra una comparación de cada una de ellas. También se incluyen características como el lenguaje en el que están escritas y la última versión subida al repositorio. La fecha de última versión nos permite saber si se le sigue dando soporte a la implementación.

Tabla 3.1 Resumen de las implementaciones.

	Funcionalidades del protocolo							Extras	
	Lenguaje	Servidor	Cliente	Resource Observation	Group communication	Proxy	Block-wise transfer	Log	Ejemplos de funcionalidades
Libcoap	C	SI	SI	SI	NO	NO	SI	SI	SI
LibNyoci	C	SI	SI	SI	SI	NO	NO	SI	SI
Microcoap	C	SI	NO	-	-	NO	-	NO	NO
Gen-coap	Erlang	Si	Si	SI	NO	SI	SI	NO	NO
Node-coap	NodeJs	SI	SI	SI	NO	SI	SI	SI	SI
Aiocoap	Python	Si	Si	SI	NO	NO	SI	SI	SI

3.3.1 Observaciones y otras funcionalidades

Dentro de la selección de las implementaciones existe una gran diferencia entre algunas implementaciones más elaboras y a su vez más pesadas como puede ser Libcoap y Libnyoci y otras más ligeras y con menos detalle como Microcoap. En la tabla 3.2 se muestra la última subida en repositorio así como su tamaño.

Una característica es la implementación de logs. En este caso, Gen-coap y Microcoap no contemplan el uso de logs dentro de su implementación y Libcoap, Libnyoci, Aiocoap y Node-coap si.

Además de los códigos de ejemplos que contienen cada una de estas implementaciones, la implementación de Libnyoci incluye una herramienta denominada nyocict. Una interfaz por linea de comando que facilita al usuario el uso de la implementación. Esta herramienta permite realizar pruebas con facilidad ya que incluye varias posibilidades dentro de las funciones de un cliente CoAP.

Tabla 3.2 Repositorio y tamaño.

Implementación	Repositorio	Tamaño (kilobyte)	Versión
Libcoap	https://github.com/obgm/libcoap	25716	02/03/2019
LibNyoci	https://github.com/darconeous/libnyoci	9732	16/04/2019
Node-coap	https://github.com/mcollina/node-coap	1392	10/04/2019
Gen-coap	https://github.com/gotthardp/gen_coap	2252	23/6/2018
Microcoap	https://github.com/1248/microcoap	308	14/5/2018
Aiocoap	https://github.com/chrysn/aiocoap	5248	26/08/2019

3.4 Instalación

El dispositivo utilizado para la instalación de las implementaciones es una Raspberry Pi 3 Model B v1.2[27]. Este dispositivo simulará un dispositivo IoT debido a sus características principales:

- Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
- 1GB RAM
- BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board
- 100 Base Ethernet
- 8GB memoria

El sistema operativo utilizado es *Raspbian GNU/Linux 9 (stretch)*[28] con kernel *Linux 4.14.98-v7+*. Todas las instalaciones han sido clonadas directamente de los repositorios oficiales de las implementaciones. La siguiente tabla incluye todos ellos.

3.4.1 Dificultades

La mayor dificultad encontrada en la instalación de una implementación es con aiocoap. Una de las librerías de las que depende y el makefile han tenido que ser modificados para eliminar el uso de SSL debido a un problema de compatibilidad al que no han dado solución en el repositorio oficial.
Otra dificultad menor, ha sido encontrada en la instalación de gen-coap. Esta implementación utiliza referencias relativas que impiden el uso de los ejemplos desde un directorio diferente al de la instalación.

3.4.2 Documentación

La instalación más tediosa de todas es la de Libcoap debido al gran número de ficheros y dependencias. No obstante contiene una guía completa[29] en su página oficial con todos los usos e instalación.

Otras implementaciones incluyen una guía completa del uso como aiocoap [30], libnyoci y node-coap. Libnyoci además facilita una herramienta por linea de comando ñyocictl'que facilita un uso reducido de la implementación.

Las dos implementaciones restantes son node-coap y microcoap que incluyen una guía rápida del uso e instalación.

4 Pruebas

En esta sección se ejecutan dos pruebas de rendimiento que nos permiten obtener una comparación más amplia de las implementaciones más allá de las funcionalidades de la tabla 3.1.

La primera prueba la nombraremos como **prueba de compatibilidad** en la que se realizarán peticiones entre todas las implementaciones posibles para conocer si todas las implementaciones pueden comunicarse entre ellas. Además esta prueba nos proporcionará los resultados de tiempos de iniciación del cliente, tiempos de respuesta del servidor y el tiempo completo, además de la compatibilidad entre los clientes.

La segunda prueba es denominada la **prueba de estrés**. Esta prueba consiste en mandar un gran número de peticiones variando la velocidad. Sirve para obtener la tasa de peticiones soportadas por el servidor. De esta prueba se obtienen los resultados de porcentaje de paquetes soportados y uso de CPU de las implementaciones.

El objeto de nuestras pruebas será la Raspberry Pi del apartado 3.4.

4.1 Prueba de compatibilidad

4.1.1 Definición

El fin de la prueba es comprobar que todas las implementaciones son compatibles entre sí. Es decir, en el caso de que un usuario quisiera utilizar una implementación en el cliente y otra distinta en el servidor, por motivo de espacio o rendimiento, esta prueba nos permitiría saber si todas son compatibles. Además se busca encontrar los tiempos promedios de respuesta de cada uno de los servidores y los tiempos de iniciación del cliente más sencillo.

4.1.2 Desarrollo

Esta prueba consiste en iniciar en el dispositivo Raspberry Pi del apartado 3.4 cada uno de los servidores disponibles, en este caso 6. Se inician secuencialmente, con una espera entre el cierre de uno y el comienzo del siguiente. En local, a cada uno de los servidores se le realizarán las 5 peticiones posibles (recordemos que la implementación microcoap no contiene la funcionalidad de cliente). Todos los servidores ofrecerán un único recurso y todos los clientes realizarán la misma petición *.well-known/core*. Durante el proceso de la prueba se medirán tanto el tiempo de respuesta del paquete en la propia interfaz local como los tiempos de envio del cliente. **Esta prueba se repetirá 3 veces para poder obtener un promedio de los resultados y aportar fiabilidad a la prueba.** Se muestra en figura 4.1.

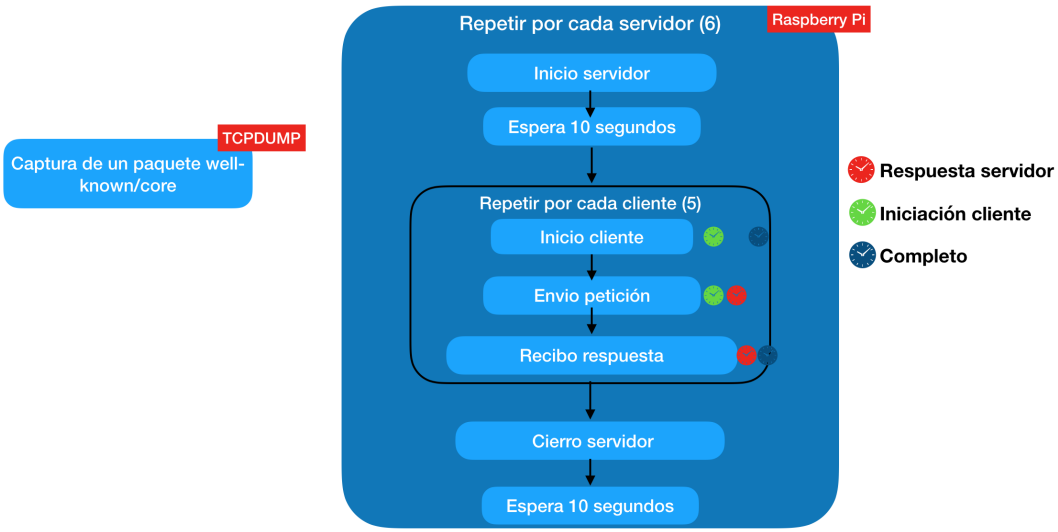

Figura 4.1 Procedimiento prueba compatibilidad.

Escenario

En el escenario de esta prueba se encuentran se encuentra: la Raspberry Pi sobre la que se realiza la prueba, seis servidores, cinco clientes y la red local en la que se comunican estos elementos durante la prueba. Descrito en la figura 4.1 Los servidores son Microcoap, Libcoap, Aiocoap, Gen-coap, Node-coap y Libnyoci. Los clientes son los mismos a excepción de Microcoap que solo implementa servidor. Se muestra en figura 4.2

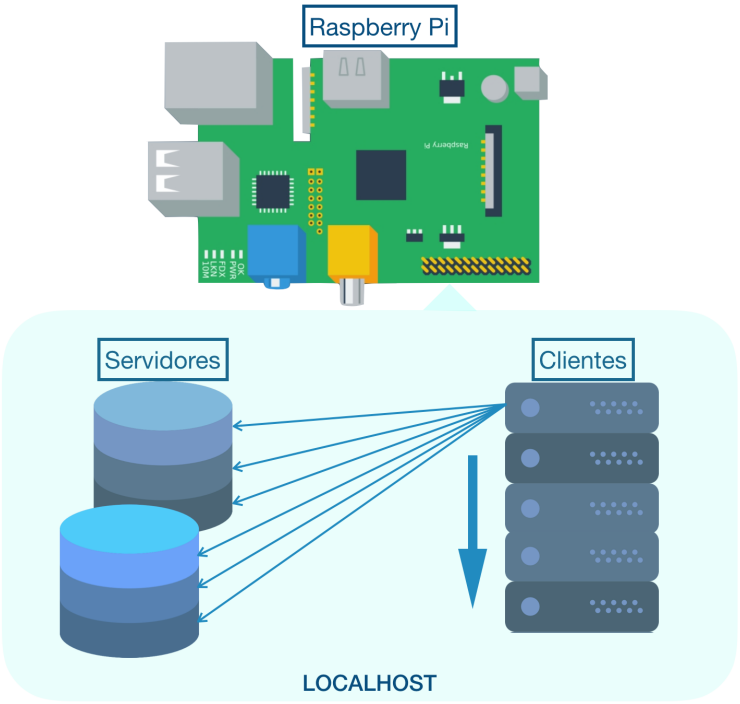

Figura 4.2 Escenario prueba compatibilidad.

Herramientas

Para el desarrollo de esta prueba han sido necesario las siguientes herramientas:

- TCPdump[31]: Herramienta que nos permite capturar los paquetes transmitidos. Uso en Raspberry.

- pruebas.py: Script que automatiza la iniciación y cierre de los servidores y realiza y mide las peticiones de todas las implementaciones. Este script es reutilizado para la prueba de estrés. Uso en Raspberry. Apendice A
- Wireshark[32]: Herramienta que nos permite analizar y obtener las estadísitcas de los ficheros pcap capturados en la Raspberry. Uso en el ordenador personal.
- Excel: Herramienta que nos permite recopilar y tratar los datos para obtener las estadísticas. Uso en el ordenador personal.
- Raspberry Pi 3 B. Apartado 3.4

Red

Todas las peticiones se han realizado en local con dirección destino 127.0.0.1. Para esta prueba de compatibilidad es suficiente. No existe más tráfico durante la prueba por esta interfaz y permite trabajar cómodamente.

La petición realizada por todos los clientes es *coap://127.0.0.1/.well-known/core* y todos los servidores deben ofrecer un único recurso *light*.

Estudio previo

Para realizar esta prueba ha sido necesario la instalación de todas las implementaciones y la modificación de los servidores de ejemplo para poder igualar las características con el resto de servidores. Se ha comprobado los tiempos necesarios para que el servidor sea estable a la hora de responder.

Automatización

El script pruebas.py, se encuentra en el apéndice A, automatiza el proceso completo. En el se definen todas las peticiones necesarias para la inicialización de servidores y la petición del cliente.

Se considera como tiempo suficiente para una inicialización correcta de los servidores 4 segundos. El mismo tiempo se utiliza como intervalo desde que se cierra uno hasta que se abre el siguiente. De esta forma se pretende que no se interfieran los servidores entre sí.

Se mide el tiempo desde que se inicia el cliente hasta que se cierra automáticamente. Todos los clientes se cierran al recibir la respuesta del servidor. A este tiempo lo denominaremos "tiempo completo". Se utilizará posteriormente en el apartado de resultados.

En el siguiente fragmento de la salida de este programa se muestran las peticiones de los cinco clientes a la implementación microcoap:

salida.txt

```
    ______________COMIENZA PRUEBA DE COMPATIBILIDAD_________________
EL SERVIDOR ES: /home/pi/tfg/installed-microcoap/microcoap/coap
All servers listening.../home/pi/tfg/installed-microcoap/microcoap/coap
EL CLIENTE ES: nodejs /home/pi/tfg/node-coap/node_modules/coap/examples/client.
    js
command: nodejs /home/pi/tfg/node-coap/node_modules/coap/examples/client.js
</.well-known/core>;ct=40,</light>;ct=0
--- 0.566246986389 seconds ---

EL CLIENTE ES: /home/pi/tfg/gen_coap/gen_coap/coap-client.sh coap
    ://[127.0.0.1]/.well-known/core
command: /home/pi/tfg/gen_coap/gen_coap/coap-client.sh coap://[127.0.0.1]/.well
    -known/core
get "coap://[127.0.0.1]/.well-known/core"
{ok,content,
    {coap_content,undefined,60,<<"application/link-format">>,
                <<"</.well-known/core>;ct=40,</light>;ct=0">>}}
```

```
--- 0.598052978516 seconds ---

EL CLIENTE ES: /home/pi/tfg/installed-libcoap/libcoap/examples/coap-client -m
    get -T cafe coap://127.0.0.1/.well-known/core
command: /home/pi/tfg/installed-libcoap/libcoap/examples/coap-client -m get -T
    cafe coap://127.0.0.1/.well-known/core
</.well-known/core>;ct=40,</light>;ct=0

--- 0.0867249965668 seconds ---

EL CLIENTE ES: /home/pi/tfg/aiocoap/aiocoap/aiocoap-client coap://127.0.0.1/.
    well-known/core
command: /home/pi/tfg/aiocoap/aiocoap/aiocoap-client coap://127.0.0.1/.well-
    known/core
</.well-known/core>;ct=40,</light>;ct=0
--- 1.32815885544 seconds ---

EL CLIENTE ES: /home/pi/tfg/libnyoci/libnyoci/src/nyocictl/nyocictl get coap
    ://127.0.0.1/.well-known/core
command: /home/pi/tfg/libnyoci/libnyoci/src/nyocictl/nyocictl get coap
    ://127.0.0.1/.well-known/core
</.well-known/core>;ct=40,</light>;ct=0

--- 0.085834980011 seconds ---

Closing server: /home/pi/tfg/installed-microcoap/microcoap/coap
```

Durante la prueba se lanza la captura de paquetes en la interfaz local con la herramienta TCPdump. El único tráfico que circula por la interfaz local durante la prueba es el tráfico CoAP generado por la prueba. Un ejemplo de los paquetes capturado durante esta primera iteración de la prueba en la que se comprueba el servidor de microcoap es:

microcoap.pcap

```
17:47:32.521643 IP localhost.37916 > localhost.5683: UDP, length 25
17:47:32.524770 IP localhost.5683 > localhost.37916: UDP, length 51
17:47:35.141303 IP localhost.35288 > localhost.5683: UDP, length 25
17:47:35.142963 IP localhost.5683 > localhost.35288: UDP, length 51
17:47:37.246651 IP localhost.37703 > localhost.5683: UDP, length 25
17:47:37.248309 IP localhost.5683 > localhost.37703: UDP, length 51
17:47:40.391557 IP localhost.34570 > localhost.5683: UDP, length 33
17:47:40.393369 IP localhost.5683 > localhost.34570: UDP, length 49
17:47:42.643027 IP localhost.61616 > localhost.5683: UDP, length 33
17:47:42.649084 IP localhost.5683 > localhost.61616: UDP, length 49
```

La diferencia de tamaño de las dos últimas peticiones y las dos últimas respuestas respecto a las tres primeras peticiones y las tres primeras respuestas se explica en el apartado 4.1.4 Observaciones.

4.1.3 Resultados

Compatibilidad

El estudio de este trabajo abarca seis implementaciones. Microcoap es la única implementación de las seis que no implementa cliente, el resto de implementaciones permiten actuar como cliente y como servidor. De esta forma, la prueba tiene que realizar 30 iteraciones que nos permitan ver todas las interacciones posibles.

La prueba ha resultado exitosa ya que todas las implementaciones permiten comunicación entre ellas,

como era de esperar si implementaban correctamente el protocolo. La tabla 4.1 muestra con un tick las comunicaciones exitosas entre todas las implementaciones.

Tabla 4.1 "Tabla compatibilidad implementaciones".

Servidor | Cliente	Libcoap	LibNyoci	Node-coap	Gen-coap	Aiocoap
Libcoap	✓	✓	✓	✓	✓
LibNyoci	✓	✓	✓	✓	✓
Node-coap	✓	✓	✓	✓	✓
Gen-coap	✓	✓	✓	✓	✓
Microcoap	✓	✓	✓	✓	✓
Aiocoap	✓	✓	✓	✓	✓

Tiempo completo

Este resultado es el obtenido directamente desde *pruebas.py*. Se mide iniciando un temporizador en el momento en el que se realiza la llamada al cliente y se termina en el instante en el que se recibe la respuesta del servidor. La figura 4.3 muestra la división de los tiempos.

Este resultado nos puede orientar a la hora de clasificar como se relacionan implementaciones distintas. En los siguientes apartados se dividirá en dos tiempos distintos. El tiempo de iniciación del cliente y el tiempo de respuesta del servidor.
Se han realizado tres pruebas idénticas para promediar los resultados. A continuación se muestran las tablas de resultados.

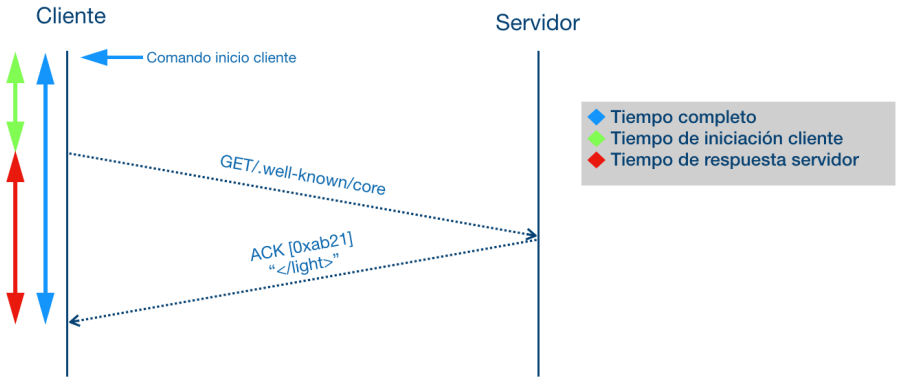

Figura 4.3 Tiempos prueba compatibilidad.

Tabla 4.2 Prueba 1: Tiempo en segundos, columna servidores, filas clientes.

	Servidores					
Clientes	MICROCOAP	NODE-COAP	GEN-COAP	LIBCOAP	AIOCOAP	LIBNYOCI
NODE-COAP	0,6082	0,6642	0,6665	0,6352	0,6380	0,6121
GEN-COAP	0,6125	0,5944	0,6051	0,5852	0,6199	0,6321
LIBCOAP	0,0897	0,0892	0,0879	0,0853	0,0985	0,0857
AIOCOAP	1,2806	1,2861	1,3269	1,2972	1,3115	1,2919
LIBNYOCI	0,0859	0,0877	0,0876	0,0857	0,0997	0,0857

Tabla 4.3 Prueba 2: Tiempo en segundos, columna servidores, filas clientes.

	Servidores					
Clientes	MICROCOAP	NODE-COAP	GEN-COAP	LIBCOAP	AIOCOAP	LIBNYOCI
NODE-COAP	0,6201	0,6384	0,6562	0,6132	0,6342	0,6122
GEN-COAP	0,6115	0,6209	0,6132	0,5919	0,6097	0,5970
LIBCOAP	0,0856	0,0893	0,0881	0,0852	0,0983	0,0849
AIOCOAP	1,2824	1,2999	1,2655	1,2996	1,2855	1,2901
LIBNYOCI	0,0856	0,0878	0,08881	0,0846	0,1003	0,0852

Tabla 4.4 Prueba 3: Tiempo en segundos, columna servidores, filas clientes.

	Servidores					
Clientes	MICROCOAP	NODE-COAP	GEN-COAP	LIBCOAP	AIOCOAP	LIBNYOCI
NODE-COAP	0,6398	0,6759	0,6569	0,6270	0,6250	0,6241
GEN-COAP	0,5916	0,6232	0,6239	0,5984	0,6031	0,5799
LIBCOAP	0,0872	0,0890	0,0880	0,0849	0,0972	0,0853
AIOCOAP	1,2666	1,3091	1,3016	1,2840	1,3069	1,2863
LIBNYOCI	0,0857	0,0884	0,0882	0,0853	0,0987	0,0851

Tabla 4.5 Promedio: Tiempo en segundos, columna servidores, filas clientes.

	MICROCOAP	NODE-COAP	GEN-COAP	LIBCOAP	AIOCOAP	LIBNYOCI	Promedio cliente
NODE-COAP	0,6227	0,6595	0,6599	0,6251	0,6324	0,6161	0,636002156
GEN-COAP	0,6052	0,6128	0,6141	0,5918	0,6109	0,6030	0,60634066
LIBCOAP	0,0875	0,0892	0,0880	0,0851	0,0980	0,0853	0,08889695
AIOCOAP	1,2765	1,2984	1,2980	1,2936	1,3013	1,2894	1,292911914
LIBNYOCI	0,0858	0,0880	0,0882	0,0852	0,0996	0,0853	0,088716851
Promedio servidor	0,5355	0,5496	0,5496	0,5362	0,5484	0,5358	

Tiempo de respuesta del servidor

Este resultado nos ofrece el tiempo que tarda el servidor en responder al cliente. Para determinar este tiempo se han usado las herramientas TCPDump y Wireshark. Se calcula restando el tiempo en el que se captura la petición al tiempo de captura del cliente. Se considera que la red no añade retardos ya que se utiliza la interfaz local y es el único tráfico que circula por ella.

Este resultado nos da una característica que nos permite elegir las implementaciones que respondan con mayor rapidez en la implementación de un servidor. Se han realizado tres pruebas, y posteriormente se han promediado.

Se obtienen 30 resultados en cada prueba resultantes de las 6 peticiones a los servidores de cada cliente (5 clientes). Las siguientes tablas muestran los tiempos de las 6 peticiones del cliente en cada prueba. La cuarta tabla muestra el promedio de las 3 pruebas. Por último se muestra una gráfica que permite al lector identificar con mayor facilidad la diferencia:

Tabla 4.6 Prueba 1: Tiempo en milisegundos, columna servidores, filas clientes.

	Servidores					
Clientes	MICROCOAP	NODE-COAP	GEN-COAP	LIBCOAP	AIOCOAP	LIBNYOCI
NODE-COAP	0,553	31,354	52,076	0,271	17,598	0,193
GEN-COAP	0,222	5,704	1,575	0,151	6,521	0,113
LIBCOAP	0,389	3,5	2,614	0,221	12,558	0,212
AIOCOAP	0,25	1,704	1,55	0,209	7,477	0,153
LIBNYOCI	0,439	2,835	2,624	0,247	13,918	0,187

Tabla 4.7 Prueba 2: Tiempo en milisegundos, columna servidores, filas clientes.

	Servidores					
Clientes	MICROCOAP	NODE-COAP	GEN-COAP	LIBCOAP	AIOCOAP	LIBNYOCI
NODE-COAP	0,519	31,296	51,951	0,233	17,69	0,194
GEN-COAP	0,225	5,663	1,422	0,151	7,068	0,12
LIBCOAP	0,4	3,588	2,55	0,245	12,792	0,203
AIOCOAP	0,27	1,887	1,546	0,164	8,269	0,152
LIBNYOCI	0,586	2,641	2,538	0,234	14,858	0,181

Tabla 4.8 Prueba 3: Tiempo en milisegundos, columna servidores, filas clientes.

	Servidores					
Clientes	MICROCOAP	NODE-COAP	GEN-COAP	LIBCOAP	AIOCOAP	LIBNYOCI
NODE-COAP	0,569	31,174	51,569	00,23	17,186	0,213
GEN-COAP	0,275	5,666	1,4	0,15	6,517	00,12
LIBCOAP	0,416	3,458	2,491	0,199	11,329	0,209
AIOCOAP	0,253	1,713	1,68	0,159	7,142	0,151
LIBNYOCI	0,438	2,636	2,504	0,242	13,054	0,169

Tabla 4.9 Promedio(varianza) de las 3 pruebas: Tiempo de respuesta servidor.

	TIEMPO PROMEDIO(ms) Y VARIANZA DE LAS TRES PRUEBAS			Resultado Final
LIBNYOCI	0,1724(0,0393)	0,17(0,0339)	0,1716(0,03907)	0,171333
LIBCOAP	0,196(0,0411)	0,2054(0,0442)	0,2198(0,0453)	0,207067
MICROCOAP	0,39020,129436	0,4(0,155147)	0,3706(0,136851)	0,386933
NODE-COAP	8,9294(12,520898)	9,015(12,535624)	9,0194(12,570321)	8,987933
AIOCOAP	11,0456 (4,916731)	12,1354(4,8325)	11,6144(5,11299)	11,598467
GEN-COAP	11,9288 (22,1649)	12,0014(22,3388)	12,0878(22,3603)	12,006

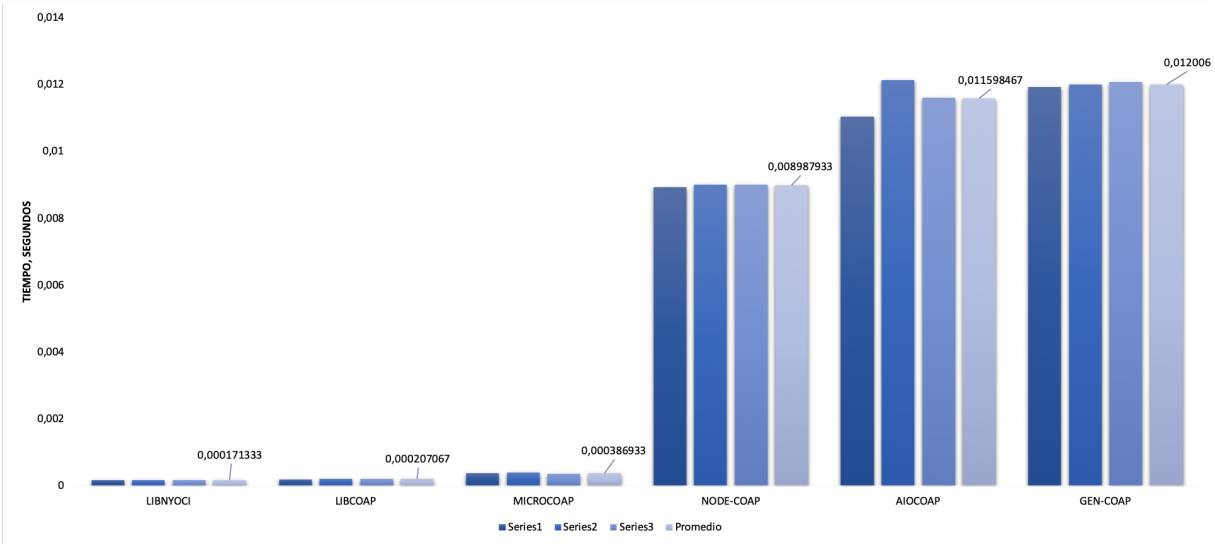

Figura 4.4 Tiempos de respuesta del servidor.

Tiempo de inicialización del cliente

Se considera como tiempo de inicialización del cliente al tiempo que transcurre desde que se lanza el comando para que el cliente realice la petición *.well-known/core* hasta el momento en el que el cliente lanza la petición por la interfaz y es capturada por TCPDump.

Se ha realizado la prueba con los clientes más simples posibles y ofreciendo un solo recurso el servi-

dor. Este resultado puede ser útil para aquellos usuarios que deseen iniciar un cliente repetidamente en los nodos.

Se obtienen 30 resultados en cada prueba resultantes de las 6 peticiones a los servidores de cada cliente (5 clientes). Las siguientes tablas muestran los tiempos de las 6 peticiones del cliente en cada prueba. La cuarta tabla muestra el promedio de las 3 pruebas. Por último se muestra una gráfica que permite al lector identificar con mayor facilidad la diferencia:

Tabla 4.10 Prueba 1: Tiempo en segundos, columna servidores, filas clientes.

	Servidores					
Clientes	MICROCOAP	NODE-COAP	GEN-COAP	LIBCOAP	AIOCOAP	LIBNYOCI
NODE-COAP	0,607689035	0,632904957	0,614520889	0,634955011	0,620474968	0,611961007
GEN-COAP	0,612290827	0,588751957	0,603551858	0,585138001	0,61341189	0,6320181
LIBCOAP	0,089378218	0,085714087	0,08530595	0,08510591	0,085981829	0,085537149
AIOCOAP	1,280356031	1,284418084	1,325435121	1,297040079	1,3040822	1,29179508
LIBNYOCI	0,085521865	0,084948813	0,085073983	0,085533859	0,085791988	0,085518996

Tabla 4.11 Prueba 2: Tiempo en segundos, columna servidores, filas clientes.

	Servidores					
Clientes	MICROCOAP	NODE-COAP	GEN-COAP	LIBCOAP	AIOCOAP	LIBNYOCI
NODE-COAP	0,619632043	0,607139841	0,604309967	0,613050873	0,616567078	0,61205609
GEN-COAP	0,611311026	0,615246929	0,611796069	0,591775813	0,602663913	0,596948071
LIBCOAP	0,085239	0,085803947	0,085581905	0,084973906	0,08557283	0,084748878
AIOCOAP	1,282167086	1,298093164	1,263970043	1,299450906	1,277262998	1,289960019
LIBNYOCI	0,085066828	0,085220061	0,086273874	0,084450134	0,085490949	0,085043867

Tabla 4.12 Prueba 3: Tiempo en segundos, columna servidores, filas clientes.

	Servidores					
Clientes	MICROCOAP	NODE-COAP	GEN-COAP	LIBCOAP	AIOCOAP	LIBNYOCI
NODE-COAP	0,639279948	0,644726936	0,605346188	0,626794889	0,607869075	0,623891023
GEN-COAP	0,591357128	0,617568034	0,622526878	0,598254169	0,596636229	0,579821988
LIBCOAP	0,086823027	0,085612082	0,08555412	0,08476909	0,085939105	0,085130069
AIOCOAP	1,26637002	1,307395019	1,300006764	1,283933903	1,299758978	1,286189952
LIBNYOCI	0,085350965	0,085775808	0,08571302	0,085120196	0,085702075	0,084975043

Tabla 4.13 Promedio de las 3 pruebas: Tiempo en segundos, columna servidores, filas clientes.

	TIEMPO INICIACIÓN CLIENTE (s)			Resultado Final
LIBNYOCI	0,085439518	0,085257619	0,085398251	0,085365129
LIBCOAP	0,085637915	0,085320078	0,086170524	0,085709506
GEN-COAP	0,601027404	0,60495697	0,605860439	0,603948271
NODE-COAP	0,624651343	0,612125982	0,620417645	0,61906499
AIOCOAP	1,290609106	1,285150702	1,297187766	1,290982525

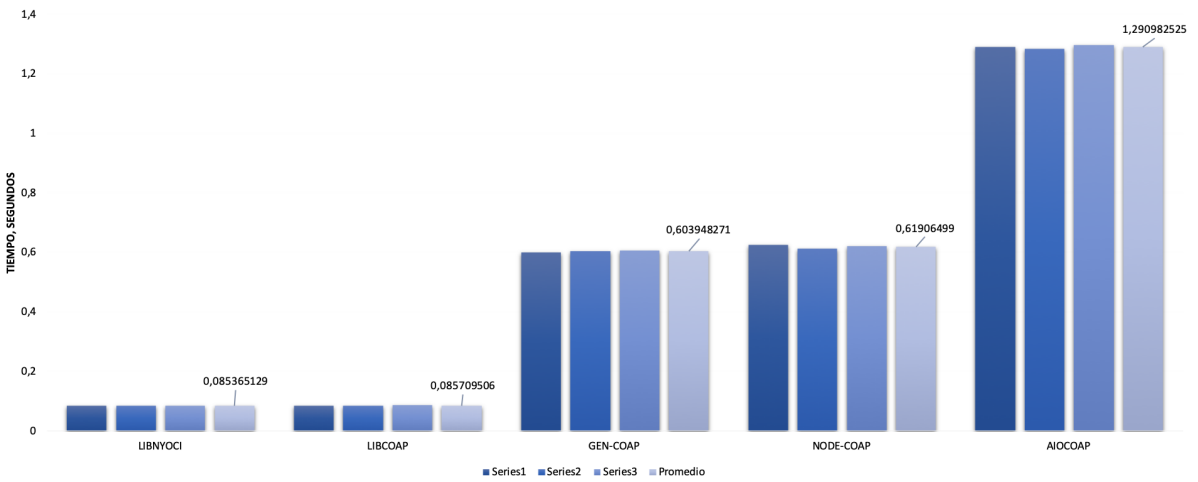

Figura 4.5 Tiempos de inicialización cliente.

4.1.4 Observaciones

Tamaño de paquete y adaptación del servidor

En esta prueba se analiza el intercambio de paquetes con Wireshark, para poder realizar una comparación real se han adaptado los clientes para realizar la misma petición.

Salvando estas diferencias, en la capa de CoAP encontramos diferencias creadas por los propios clientes de las implementaciones.

1. **TKL**: Longitud del token, indica la longitud que tendrá el token, esta puede ser variable entre 0 y 8 bytes, pero no se fija ningún valor, esta elección la toma el cliente según el entorno.
2. **Message ID**: Identificador único del mensaje.
3. **Token**: Se utiliza para poder relacionar una solicitud con una respuesta. En caso de que exista un alto nivel de tráfico en la red es aconsejable que aumente el tamaño del mismo.
4. **Opciones**: Tanto la petición como la respuesta deben incluir al menos una opción. Estas opciones varían desde la URI hasta las opciones de metadatos de los paquetes HTTP.

En el apartado 4.1.2, el fichero *microcoap.pcap* muestra que las peticiones ocupan los tamaños mostrados en la tabla 4.14.

Tabla 4.14 Tamaño de trama en bytes.

Clientes	Libcoap	Gen-coap	Node-coap	LibNyoci	Aiocoap
Petición Cliente	25	25	25	33	33
Respuesta Microcoap	51	51	51	49	49

Las diferencias que encontramos son las siguientes:

1. Tamaño del Token: en las libcoap, gen-coap y node-coap el token tiene tamaño 4 y en Libnyoci y aiocoap el tamaño es 2.
2. Opciones: Como podemos observar, en las dos implementaciones en las que el tamaño del token es menor, se incluye una opción *Uri-Host*. Esta opción incluye el nombre del host al que se le hace la petición. Véase en figura 4.6 y 4.7.

Este hecho se repite en todas las iteraciones para cada servidor, esto nos lleva a la conclusión de que todos los servidores de estas implementaciones son capaces de interpretar y responder tanto cuando se varía el

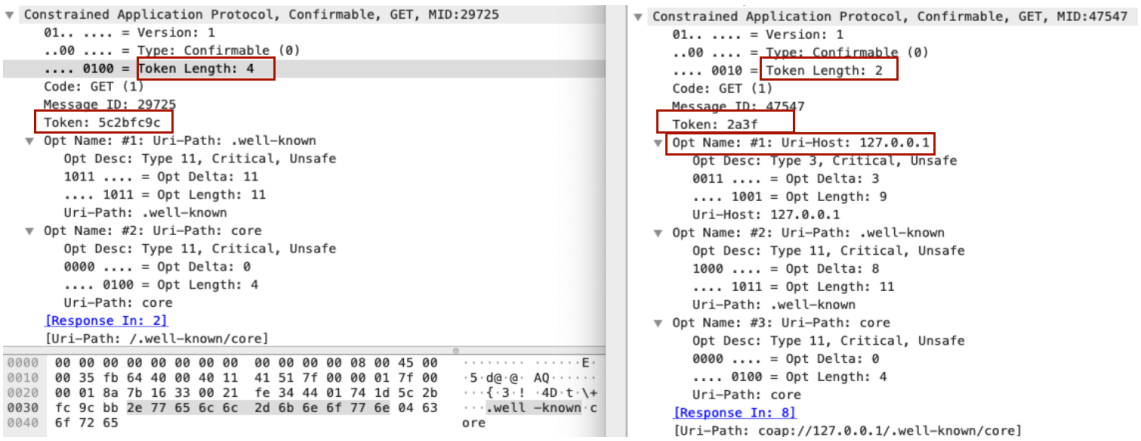

Figura 4.6 Comparación peticiones.

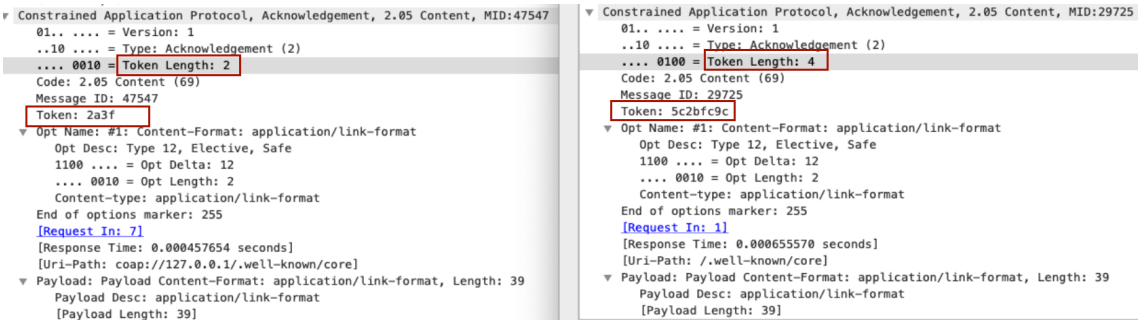

Figura 4.7 Comparación respuestas servidor.

tamaño de token como cuando se añaden nuevas opciones. Figura 4.7.

Implementación excluida

Se puede comprobar que todas las implementaciones son capaces de interaccionar entre sí. No obstante, según el estudio de Markel Iglesias-Urkia, Adrian Orive y Aitor Urbieta[5] existe una implementación que no es capaz de interaccionar con el otras. Esta implementación es CoapPy y no ha sido objeto de este estudio debido a que sus últimas modificaciones se realizaron en octubre de 2013. Esto significa que esta implementación ya no está siendo soportada y ha sido excluida de este estudio.

4.2 Prueba de estrés

4.2.1 Definición

La prueba de estés permite al usuario determinar la capacidad de una computadora, red, programa o servidor para mantener su efectividad bajo condiciones desfavorables. Este proceso conlleva la realización de pruebas de manera repetitiva. Con los resultados de esta prueba podremos hallar la frecuencia de errores del sistema y la vulnerabilidad de cara a un ataque de denegación de servicio. Podemos deducir entonces, que a día de hoy las pruebas de estrés son necesarias[33] no solo para conocer la efectividad sino por seguridad a la hora de prevenir un ataque de denegación de servicios.

Los resultados obtenidos en esta prueba se basarán en el porcentaje de acierto de los servidores en situación desfavorable y la utilización de la CPU de las Raspberry Pi durante la prueba.

4.2.2 Desarrollo

La prueba que se ha realizado para probar las implementaciones CoAP es la siguiente: se ha capturado un paquete de la primera prueba y se ha reenviado a la interfaz del cable Ethernet de la Raspberry Pi desde un ordenador con mejores prestaciones. El paquete se ha enviado en bucle a tres velocidades distintas. Para elegir la cantidad de paquetes enviadas y la velocidad se han realizado pruebas previas. Un mayor número de paquetes saturaban la interfaz y obligaban al kernel del dispositivo a despreciarlos. La interfaz utilizada solo sostiene trafico CoAP y es la nombrada en el apartado de instalación.**Esta prueba se repetirá 3 veces para poder obtener un promedio de los resultados y aportar fiabilidad a la prueba.** El procedimiento completo se meustra en la figura 4.8.

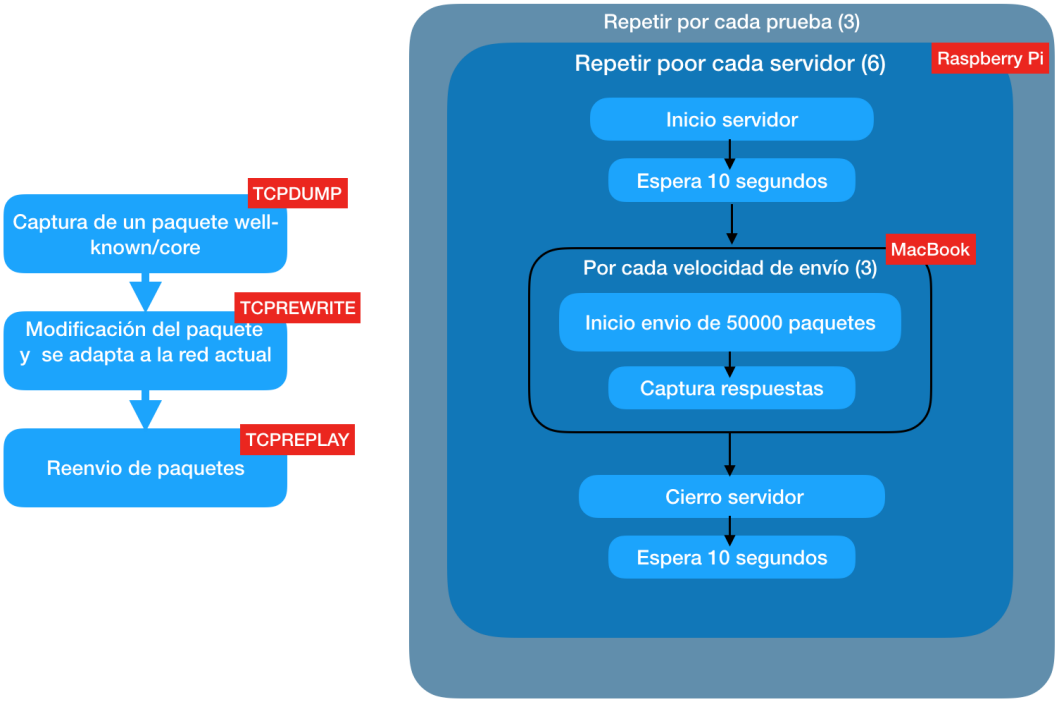

Figura 4.8 Procedimiento prueba de estrés.

4.2.3 Escenario

En esta prueba nos encontramos los siguientes elementos: ordenador MacBook que se encarga de enviar los paquetes de la prueba de estrés, Raspberry Pi que tiene instalado los 6 servidores posibles y una red creada para la conexión de los dos dispositivos nombrados con dirección 10.10.10.0/24. El MacBook posee la dirección 10.10.10.21 y la Raspberry 10.10.10.22. El paquete que envía el MacBook ha sido seleccionado y modificado como se explica a continuación. Este paquete se reenvía 5000 veces a cada uno de los servidores. El escenario se muestra en la figura 4.9.

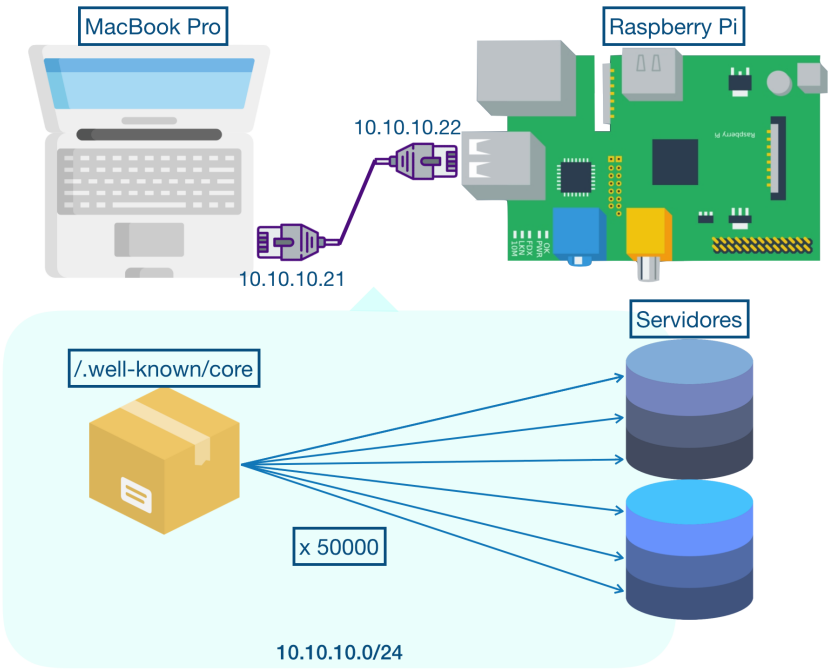

Figura 4.9 Tiempos prueba compatibilidad.

Herramientas

- TCPdump: Herramienta que nos permite capturar los paquetes transmitidos. Uso en Raspberry.
- TCPReplay y TCPRewrite [34]: Herramienta que nos permite modificar un paquete y reenviarlo por una interfaz.
- pruebas.py: Script que automatiza la iniciación y cierre de los servidores y realiza y mide las peticiones de todas las implementaciones. Este script es reutilizado para la prueba de estrés. Uso en Raspberry.
- stas.sh: Script simple usado en el equipo personal para extraer el número de paquetes enviados y recibidos de los archivos .pcap para poder exportarlo a Excel.
- Wireshark y Tshark: Herramienta que nos permite analizar y obtener las estadísitcas de los ficheros pcap capturados en la Raspberry. Uso en el ordenador personal.
- Excel: Herramienta que nos permite recopilar y tratar los datos para obtener las estadísticas. Uso en el ordenador personal.
- Raspberry Pi 3 B. Apartado 3.4
- MacBook Pro (Retina, 13 pulgadas, mediados de 2014), procesador 2,6 GHz Intel Core i5 y memoria 8GB.

stats.sh

```
while IFS= read -r line
do
  echo "$line"
  tshark -r $line -Y "ip.src == 10.10.10.21" | wc -l
  tshark -r $line -Y "ip.src == 10.10.10.22" | wc -l
done < todosnode.txt
```

Red

Se ha creado una subred con dos únicos dispositivos, el equipo con los servidores al que hacer la prueba y el equipo que envía la petición y captura el número y la calidad de las respuestas. El primer dispositivo, las Raspberry tiene la dirección 10.10.10.22 y el segundo la dirección 10.10.10.21. El único trafico que navega por la red una vez establecidos los vecinos es el tráfico procedente de la prueba.

Automatización

Para esta prueba se ha utilizado el mismo script pruebas.py de la anterior prueba pero con el argumento que habilita la prueba de estrés. La función del script es únicamente iniciar y cerrar los servidores correctamente. Además se han utilizado otros scripts como stats.sh para la extracción de estadísticas.

Reenvío de paquetes

El paquete utilizado para la prueba ha sido extraído de la anterior prueba y se ha modificado[35] con la herramienta TCPRewrite. Se le ha cambiado la dirección IP origen, la dirección IP destino y se han sustituido por las nuevas 10.10.10.21 y 10.10.10.22. También ha sido necesario modificar la dirección MAC origen y reconstruir el checksum del paquete.

Estudio previo

Para realizar la prueba se comprobó la capacidad de las Raspberry de recibir paquetes y la velocidad máxima que permitía la conexión. Así es que cuando el número de paquetes enviados a la máxima velocidad permitida era superior a 56000 los paquetes eran ignorados por el kernel, por eso se eligió enviar 50000 paquetes. La máxima velocidad que era estable eran 19MBps, cuando la velocidad era mayor la prueba resultaba con valores no exactos entre 19MBps y 20MBps.

Una vez conocíamos las limitaciones de la red se eligieron los valores de 10MBps, 15MBps y 19MBps y 50000 peticiones como los valores para nuestra prueba de estrés. Estos valores nos permiten diferenciar con claridad la efectividad de todos los servidores implementados bajo una situación no favorable. Las tramas de las respuestas oscilan entre 50 y 60 Bytes aunque todas ofrecen un único recurso. En la tabla 4.16 también se muestra la velocidad en paquetes por segundo (pps) ya que puede ser de mayor interés el lecto.

4.2.4 Resultados

Número de paquetes perdidos

Se analiza los paquetes recibidos y enviaos durante la prueba desde el MacBook. Como resultado obtenemos la siguiente tabla. Todos los paquetes han sido enviados correctamente según todos los archivos .pcap recopilados durante las pruebas. Una vez tenemos los paquetes recibidos podremos obtener el porcentaje de respuestas del servidor según la situación. El porcentaje se mostrará en la segunda tabla 4.16 y se muestra una gráfica que permite al usuario distinguir con facilidad las diferencias entre las implementaciones, Figura 4.6.

Tabla 4.15 Resultados de la prueba de estrés.

		10 MBps			15MBps			19MBps		
		P1	P2	P3	P1	P2	P3	P1	P2	P3
Aiocoap	Paquetes enviados	50000	50000	50000	50000	50000	50000	50000	50000	50000
	Paquetes recibidos	886	907	880	725	679	531	576	590	597
Microcoap	Paquetes enviados	50000	50000	50000	50000	50000	50000	50000	50000	50000
	Paquetes recibidos	12624	12613	17829	7671	7808	7574	5712	5701	5702
Node-coap	Paquetes enviados	50000	50000	50000	50000	50000	50000	50000	50000	50000
	Paquetes recibidos	28038	27864	27937	19052	19027	18095	14587	14785	15218
Gen-coap	Paquetes enviados	50000	50000	50000	50000	50000	50000	50000	50000	50000
	Paquetes recibidos	29430	31605	20520	15938	15965	28492	16017	15980	21227
Libnyoci	Paquetes enviados	50000	50000	50000	50000	50000	50000	50000	50000	50000
	Paquetes recibidos	49100	48316	48491	31709	31758	31455	23436	23254	22775
Libcoap	Paquetes enviados	50000	50000	50000	50000	50000	50000	50000	50000	50000
	Paquetes recibidos	49520	46711	49485	46825	47592	47124	37047	36884	37043

Tabla 4.16 Prueba de estrés, porcentaje de acierto.

	18656.60 pps - 10 Mbps	27984.83 pps - 15Mbps	35443.84 pps - 19 Mbps
AIOCOAP	1,78200 %	1,29000 %	1,17533 %
MICROCOAP	28,71066 %	15,36866 %	11,41000 %
NODE-COAP	55,89266 %	37,44933 %	29,72667 %
GEN-COAP	54,37000 %	40,26333 %	35,48267 %
LIBNYOCI	97,27133 %	63,28133 %	46,31000 %
LIBCOAP	97,14400 %	94,36067 %	73,98267 %

CPU utilizada

El segundo resultado que nos ofrece esta prueba es el porcentaje de CPU y de RAM utilizada por cada uno de los servidores durante la prueba de estrés. Este dato ha sido anotado desde que el servidor durante las tres pruebas, obteniendo el máximo y el mínimo. Se considera que el servidor esta estable cuando el uso de CPU del mismo deja de reducir. En ese momento es en el que se inicia la prueba.

** Este dato se verá en el siguiente apartado. Observaciones*

Tabla 4.17 Resultado CPU de estrés.

	CPU(%) reposo	CPU(%) pico	RAM(%) reposo	RAM(%) pico
NODE-COAP	0.1 %	1 %	2.9 %	3.3 %
LIBCOAP	0.3 %	3.5 %	0 %	0 %
MICROCOAP	3 %	6 %	0 %	0 %
AIOCOAP	4.5 %	6.5 %	1.8 %	1.9 %
LIBNYOCI	5 %	7.5 %	0 %	0 %
GEN-COAP	1 %	18 % *	1.5 %	1.6 %

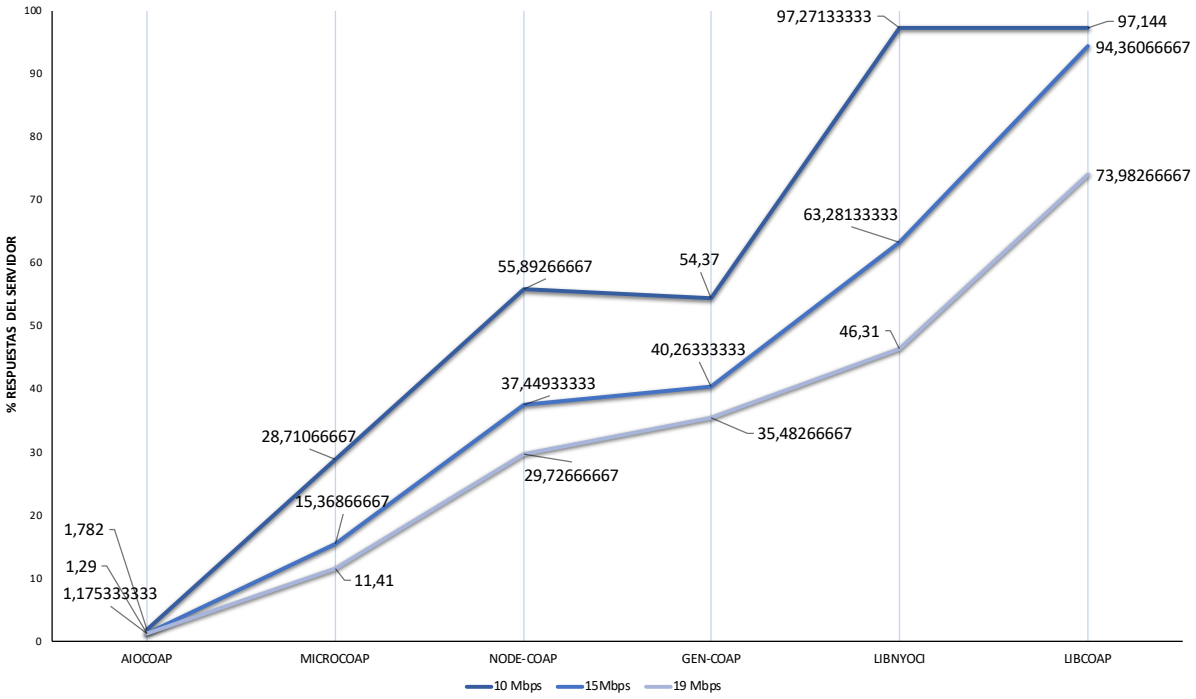

Figura 4.10 Prueba de estrés, porcentaje de acierto.

4.2.5 Observaciones

Durante la preparación de la prueba se realizaron varias peticiones para poder elegir el tiempo medio necesario desde que se inicia el servidor hasta que comienza a ser efectivo. Durante estas pruebas previas hubo dos implementaciones que destacaron: Libcoap necesitaba una espera mucho mayor que el resto de las implementaciones para poder empezar a responder un alto porcentaje de paquetes, es decir, en el caso de realizar la prueba de estrés durante los primeros 5 segundos, la implementación no respondía al 50 % de los paquetes. Sin embargo, libnyoci demostraba el mismo porcentaje de respuestas desde el primer instante.

Otro dato a destacar es la diferencia de tiempo en el que libera la CPU unas implementaciones y otras. En particular la implementación Gen-coap alcanzaba altos niveles de uso de CPU y el tiempo de recuperación era mayor al del resto de las implementaciones. Esto podría permitir un posible ataque DoS utilizando esta vulnerabilidad en cuestión de segundos.

5 Interpretación de resultados

Los resultados mostrados en los anteriores capítulos no permiten la elección de una única implementación que supere en prestaciones al resto de implementaciones en todas las situaciones. Esto se debe a que en todos los resultados obtenidos hay alguna implementación superior a otra pero no coincide esa implementación en todos los resultados.

En cuanto a la **instalación y tamaño de la implementación** existe una gran diferencia entre las implementaciones. Sin duda, la implementación que permite una instalación trivial y ocupa poco espacio es **Microcoap**, pero esta implementación no permite el uso de cliente luego se podría descartar de un gran número de casos en los que se necesita un cliente y no solo un servidor. Descartando esta primera implementación, la implementación más favorable es **Node-coap** en este aspecto: es la segunda implementación que menos espacio ocupa y su instalación es también sencilla y guiada en la documentación. Por dificultad, la implementación que más problemas muestra a la hora de instalar es Aiocoap y la implementación más pesada es Libcoap.

En el ámbito de **funcionalidades del protocolo implementadas y características de la implementación** no es sencillo destacar una implementación sobre las otras. Existen dos implementaciones que realizan un mayor número de funcionalidades, estas son Gen-coap y Node-coap. No obstante, Libcoap ofrece un log más detallado, una mejor documentación y Libnyoci ofrece una herramienta que permite utilizar los ejemplos con facilidad a través de una interfaz. Esto complicaría la toma de decisión del usuario ya que, partiendo de que la implementación deseada realice la funcionalidad buscada, debería basarse en más resultados para tomar la elección.

El siguiente resultado que obtenemos en este documento es **el tiempo de respuesta del servidor**. En el tiempo de respuesta del servidor existe una clara diferencia que destaca a tres servidores sobre el resto. **Estos son Libnyoci, Libcoap y un poco mayor pero en la misma escala, Microcoap**. En este caso la diferencia con el resto de servidores es bastante notable ya que disminuyen mas de 200 veces el tiempo de respuesta.

El siguiente resultado es **los tiempos de iniciación del cliente**, resultado en el que Microcoap queda descartado debido a su carencia de cliente. En este caso **vuelve a existir una gran superioridad de Lbnyoci y Libcoap** respecto al resto. No obstante, la implementación más lenta Aiocoap.

En la **prueba de estrés** el resultado se muestra gráficamente en la figura 4.6. En este caso, **la implementación con mayor capacidad** para responder en condiciones desfavorables en las que el número de paquetes sea elevado **es Libcoap**. Aiocoap es la implementación menos efectiva en estas condiciones y Microcoap tampoco llega a responder a más de un tercio de las peticiones. Si el uso de la implementación va a ser sometido a situaciones de estrés la implementación más adecuada es Libcoap seguida de Libnyioci.

Por último, el resultado de **consumo de recursos, CPU y RAM**. En este resultado, teniendo en cuenta solo el consumo de CPU, **la implementación Node-coap** se encuentra por encima del resto de implementaciones pero la diferencia no es tan grande como en otros resultados. Si tenemos en cuenta el consumo de RAM, aunque ninguno supere el 3.3 %, los inferiores son Libcoap, Microcoap y Aiocoap. No obstante, si cree que puede recibir un alto flujo de tráfico habría que limitar el servidor de Gen-coap para evitar llevar al procesador a su limite de uso.

5.1 Conclusión

Para un uso en nodos muy limitados y en una red en el que el tráfico no es elevado, las implementaciones Node-coap y Gen-coap pueden ser una buena elección. Son dos implementaciones que ocupan poco espacio, y en un uso moderado consumen poca CPU. En otras características están las dos muy igualadas, por ejemplo en tiempo de iniciación del cliente, tiempo de respuesta del servidor y situaciones desfavorables. En esta misma situación, si solamente es necesario implementar un servidor simple, la mejor opción es Microcoap. Esta implementación es muy ligera y su tiempo de respuesta es también inferior a las dos anteriores.

No obstante, si lo que el usuario busca es implementaciones con muy buenas prestaciones en tiempo de respuesta y eficiencia en situaciones desfavorables las mejores implementaciones son Libcoap y Libnyioci. Aunque sean más pesadas, te ofrecen más información y más ejemplos.

6 Trabajo futuro

El número de pruebas que abarca este documento podría ser ampliado para poder analizar más funcionalidades de las implementaciones.

Es de interés realizar una prueba que verifique el uso de cada una de las funcionalidades del protocolo CoAP implementadas que se muestran en la tabla 3.1. Así como una prueba de compatibilidad con un mayor número de recursos y funcionalidades en las que se probasen de manera cruzada todas las funcionalidades de las implementaciones. Por ejemplo realizar la prueba de compatibilidad con block-wise.

Una prueba de interés es el desarrollo de un proxy con las implementaciones utilizadas y comparar los resultados de tiempo y eficiencia con otras implementaciones destacadas del protocolo HTTP. Los resultados serían de interés ya que en diversas ocasiones se puede utilizar el protocolo CoAP cómo pasarela entre el ámbito de IoT y el de los nodos con mayor prestación.

Por último, sería conveniente realizar una prueba exclusiva a la implementación Gen-coap. Esta prueba abarcaría el ámbito de la seguridad ya que serán pruebas de alta carga con un tiempo prolongado que permitan hacer una denegación de servicio a la Raspberry Pi. Esta prueba sería de interés debido al resultado obtenido en la prueba de estrés que nos muestra la rapidez con la que Gen-coap abarca el uso de CPU.

Apéndice A
prueba.py

pruebas.py

```
# coding: utf-8
import os
import signal

import sys
import time
import subprocess
import logging
from datetime import datetime


clientes = ["nodejs /home/pi/tfg/node-coap/node_modules/coap/examples/client.js
    ",           #NODE-COAP
     "/home/pi/tfg/gen_coap/gen_coap/coap-client.sh coap://[127.0.0.1]/.well-
         known/core",       #GEN-COAP
     "/home/pi/tfg/installed-libcoap/libcoap/examples/coap-client -m get coap
         ://127.0.0.1/.well-known/core",   #LIBCOAP
     "/home/pi/tfg/aiocoap/aiocoap/aiocoap-client coap://127.0.0.1/.well-known
         /core",        #AIOCOAP
     "/home/pi/tfg/libnyoci/libnyoci/src/nyocictl/nyocictl get coap
         ://127.0.0.1/.well-known/core"]     #LIBNYOCI

servidores = ["/home/pi/tfg/installed-microcoap/microcoap/coap",             #
    MICROCOAP
     "nodejs /home/pi/tfg/node-coap/node_modules/coap/examples/server.js",
           #NODE-COAP
     "/home/pi/tfg/gen_coap/gen_coap/coap-server.sh",                #GEN-
         COAP
     "/home/pi/tfg/installed-libcoap/libcoap/examples/coap-server",          #
         LIBCOAP
     "python3.5 /home/pi/tfg/aiocoap/aiocoap/server.py",               #
         AIOCOAP
     "/home/pi/tfg/libnyoci/libnyoci/src/examples/example-1"]

def start_servers(server):
  logger.debug("Se inicia función start_servers")
  print "Servidor escuchando con el comando:" + server
  result = subprocess.Popen(server, shell=True)
  time.sleep(4)
  return result
```

```
def close_servers(p,server):
  logger.debug("Se inicia función close_servers")
  try:
    print "Se cierra el servidor: "+ server
    os.killpg(os.getpgid(p.pid), signal.SIGINT)
  except:
    print "Exception closing servers"
    logger.debug("EXCAPCIÓN función close_servers")
  time.sleep(4)

def send_client(client):
  logger.debug("Se inicia función send_client")
  print "command del cliente: " +client
  start_time = time.time()
  result = subprocess.check_output(client, shell=True)
  T = "--- & %s seconds ---" % (time.time() - start_time)
  print "El cliente: "+ client + T + "\n"


def prueba_compatibilidad():
  logger.debug("Se inicia prueba_compatibilidad()")
  for i in servidores:
    print "El servidor es: " + i
    p = start_servers(i)
    for e in clientes:
      print "El cliente es: " + e
      send_client(e)
      time.sleep(2)
    close_servers(p,i);


def prueba_estres():
  logger.debug("Se inicia prueba_estrés()")
  f = open('estres.txt', 'w')
  f.write("_____________COMIENZA PRUEBA DE ESTRÉS_____________________\n")
  for i in servidores:
    print "EL SERVIDOR ES: " + i
    p = start_servers(i)
    raw_input("Press Enter to continue...")
    close_servers(p,i);
  f.write("_________________FIN PRUEBA COMPATIBILIDAD___________________\n")
  f.close()



# datetime object containing current date and time
now = datetime.now()
dt_string = now.strftime("%d-%m-%Y--%H-%M-%S")
file = "log-"+dt_string


logging.basicConfig(filename=file,
                    format='%(asctime)s %(message)s',
                    filemode='w')
logger = logging.getLogger()
logger.setLevel(logging.DEBUG)
```

```
if len(sys.argv) <= 1:
  print("introduzca tipo de prueba");
else:
  client = sys.argv[1]
  server = sys.argv[2]
  tipo = int(sys.argv[3])

  if tipo == 1:
    print "_____________COMIENZA PRUEBA DE COMPATIBILIDAD_____________"
    prueba_compatibilidad();
    print "_____________FIN PRUEBA COMPATIBILIDAD_____________"

  else:
    if tipo == 2:

      print "_____________COMIENZA PRUEBA DE ESTRÉS_____________"
      prueba_estres()
      print "_____________FIN PRUEBA ESTRÉS_____________"
```

Índice de Figuras

Índice de Tablas

Bibliografía

[1] Zach Shelby, Klaus Hartke, and Carsten Bormann. The constrained application protocol (coap). Technical report, 2014.

[2] Geoff Mulligan. The 6lowpan architecture. In *Proceedings of the 4th workshop on Embedded networked sensors*, pages 78–82. ACM, 2007.

[3] Tapio Levä, Oleksiy Mazhelis, and Henna Suomi. Comparing the cost-efficiency of coap and http in web of things applications. *Decision Support Systems*, 63:23–38, 2014.

[4] Jurado Pérez Luis Alberto, Velásquez Vargas Washington Adrián, and Vinueza Escobar Nelson Fernando. Estado del arte de las arquitecturas de internet de las cosas (iot). 2014.

[5] Markel Iglesias-Urkia, Adrián Orive, and Aitor Urbieta. Analysis of coap implementations for industrial internet of things: a survey. *Procedia Computer Science*, 109:188–195, 2017.

[6] Roy Thomas Fielding. Architectural styles and the design of network-based software architectures. 2000. *University of California, Irvine*, page 162, 2000.

[7] Tim Berners-Lee, Roy Fielding, and Henrik Frystyk. Hypertext transfer protocol–http/1.0. Technical report, 1996.

[8] Internet Engineering Task Force. *RFC 791 Internet Protocol - DARPA Inernet Programm, Protocol Specification*, September 1981.

[9] J. Postel. User datagram protocol. STD 6, RFC Editor, August 1980. *http://www.rfc-editor.org/rfc/rfc768.txt*.

[10] K. Hartke. Observing resources in the constrained application protocol (coap). RFC 7641, RFC Editor, September 2015.

[11] Z Shelby, K Hartke, C Bormann, and B Frank. Constrained application protocol (coap), draft-ietf-core-coap-13. *Orlando: The Internet Engineering Task Force–IETF*, 2012.

[12] Hasan Ali Khattak, Michele Ruta, and Eugenio Di Sciascio. Coap-based healthcare sensor networks: A survey. In *Proc. 11th Int. Bhurban Conf. Appl. Sci. Technol.(IBCAST)*, pages 499–503, 2014.

[13] Seung-Man Chun, Hyun-Su Kim, and Jong-Tae Park. Coap-based mobility management for the internet of things. *Sensors*, 15(7):16060–16082, 2015.

[14] Unión Europea. Fiware, jan 2012.

[15] Yasir Mehmood, Farhan Ahmad, Ibrar Yaqoob, Asma Adnane, Muhammad Imran, and Sghaier Guizani. Internet-of-things-based smart cities: Recent advances and challenges. *IEEE Communications Magazine*, 55(9):16–24, 2017.

[16] Suhas Rao, Devaiah Chendanda, Chetan Deshpande, and Vishwas Lakkundi. Implementing lwm2m in constrained iot devices. In *2015 IEEE Conference on Wireless Sensors (ICWiSe)*, pages 52–57. IEEE, 2015.

[17] Michel Veillette, Peter Van der Stok, Alexander Pelov, Andy Bierman, and Ivaylo Petrov. CoAP Management Interface. Internet-Draft draft-ietf-core-comi-07, Internet Engineering Task Force, July 2019. Work in Progress.

[18] Olaf Bergmann. libcoap: C-implementation of coap. *URL: https://github.com/obgm/libcoaphttps://libcoap.net/doc/install.html*, 2012.

[19] Libnyoci. *URL: https://github.com/darconeous/libnyoci*.

[20] Microcoap. *URL: https://github.com/1248/microcoap*.

[21] Erlang coap. *URL: https://github.com/gotthardp/gen_coap*.

[22] Node-coap. *URL: https://github.com/mcollina/node-coap*.

[23] Aiocoap. *URL: https://github.com/chrysn/aiocoap*.

[24] Matthias Kovatsch. Copper (cu) coap user-agent. *URL: https://github.com/mkovatsc/Copper*.

[25] Lobaro. Complete coap implementation in c. lobaro coap. *https://github.com/lobaro/lobaro-coap*.

[26] Peter A. Bigot. Coapy: Python implementation of constrained application protocol. *URL: https://github.com/pabigot/coap*.

[27] Raspberry pi 3 b. *URL: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/*.

[28] Raspbian. *URL: https://www.raspberrypi.org/downloads/raspbian/*.

[29] Guía completa de libcoap. *URL: https://libcoap.net/doc/reference/4.2.0/annotated.html*.

[30] Guía completa de aiocoap. *URL: https://aiocoap.readthedocs.io/en/latest/examples.html*.

[31] Van Jacobson, Craig Leres, and S McCanne. The tcpdump manual page. *Lawrence Berkeley Laboratory, Berkeley, CA*, 143, 1989.

[32] Laura A Chappell. *Wireshark network analysis: the official Wireshark certified network analyst study guide*. Protocol Analysis Institute, Chappell University, 2010.

[33] Mutiara Hana. Stress testing is a must, apr 2018.

[34] A Turner. Tcpreplay–pcap editing & replay tools.

[35] Dan Nanni. How to capture and replay network traffic on linux, may 2013.