Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Prueba de concepto sobre gestión con CORECONF

Autor: Antonio Estévez Begines
Tutor: Antonio José Estepa Alonso

Dpto. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2024

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Prueba de concepto sobre gestión con CORECONF

Autor:
Antonio Estévez Begines

Tutor:
Antonio José Estepa Alonso
Profesor titular

Dpto. de Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2024

Autor: Antonio Estévez Begines

Tutor: Antonio José Estepa Alonso

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2024

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

Con estas líneas pretendo agradecer el apoyo recibido por parte de todos los compañeros y amigos que he encontrado durante el transcurso de este grado, y a mi familia, en especial a mi hermana Inma por ser un constnte apoyo y estar siempre ahí.

También me gustaría agradecer a todos los profesores que han participado en mi formación, tanto en los últimos años en este grado como durante toda la secundaria y primaria. Todos han aportado y han formado parte del proceso que me ha llevado hasta aquí y a ser quien soy.

Por último, agradecer a mi grupo de amigos desde la infancia que año tras año siguen ahí y se siguen preocupando porque todo vaya bien.

Muchas gracias a todos.

Resumen

En este Trabajo Fin de Grado se realiza un estudio del borrador del protocolo de gestión CORECONF del IETF, realizando una implementación del mismo y comparativas de rendimiento respecto a su competidor LwM2M. Las implementaciones de ambos protocolos se han probado en placas de desarrollo NodeMCU por estar enfocados en nodos restringidos.

A nivel teórico se ha observado una mayor complejidad del protocolo LwM2M, ya que resuelve cuestiones como el resgistro y desregistro de dispositivos o el uso obligatorio de objetos de seguridad que CORECONF no tiene en cuenta. Además, es más flexible en aspectos como la codificación de los datos, ya que permite formatos como JSON, TLV o texto plano mientras que CORECONF siempre utiliza CBOR como codificación.

Comparando el rendimiento de ambas implementaciones, hemos obtenido que CORECONF cumple con su objetivo de ser más eficiente en el intercambio de información, utilizando menos bytes que LwM2M en la mayoría de peticiones/respuesta.

CORECONF también cumple con el objetivo de ser implementable en entornos restringidos y con pocos objetos de gestión. Necesita menos recursos que LwM2M para realizar una implementación básica, en nodos con alrededor de la decena de objetos de gestión. Esta ventaja se diluye al realizar implementaciones de nodos con más objetos que gestionar.

Abstract

This Final degree Project approaches the study of the draft of the IETF CORECONF management protocol. The study covers the implementation of this protocol and a comparison of its performance against its competitor LwM2M. The implementations of both protocols have been tested on NodeMCU development boards as they are focused on restricted nodes.

At a theoretical level, the LwM2M protocol is more complex, as it solves issues such as device registrarion and de-registration or the mandatory use of security objects that CORECONF does not take into account. In addition, it is more flexible in aspects such as data encoding, as it allows formats such as JSON, TLV or plain text while CORECONF always uses CBOR as encoding.

Comparing the performance of both implementations, we have obtained that CORECONF meets its goal of being more efficient in the exchange of data, using fewer bytes than LwM2M in most request/responses.

CORECONF also meets the goal of being deployable in restricted environments with few management objects. It requires fewer resources than LwM2M to run a basic implementation, on nodes aroun ten management objects. This advantage is diluted when implementing nodes with more objects to manage.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

En medio de la dificultad, reside la oportunidad.

- Albert Einstein-

El avance de la tecnología y el desarrollo de nuevas aplicaciones crean nuevos retos en cuanto a los protocolos de gestión a utilizar en cada escenario. La existencia de distintas organizaciones y organismos encargados de definir estándares acaban provocando la existencia de múltiples protocolos con funcionalidades similiares. En concreto, en este TFG se abordará un estudio sobre la interfaz de gestión de redes y nodos restringidos CORECONF propuesta por el IETF y se comparará con su principal rival en el mercado actualmente, el estándar LwM2M definido por la OMA.

1.1. Problemática

En 2023, se alcanzaron los 15.100 millones de dispositivos conectados en el mundo [1]. Esta cifra se espera que continúe creciendo hasta duplicarse en 2030 debido al crecimiento del IoT. Dentro de esta gran cantidad de dispositivos, nos encontramos con los llamados "dispositivos restringidos", es decir, dispositivos con pocos recursos computacionales y pequeño tamaño.

Existe una amalgama de estándares de gestión que son necesarios analizar y ver cuál es el más adecuado para este tipo de dispositivos. Los dos principales son el estándar LwM2M (definido por la OMA) y el aún en proceso de draft, CORECONF (del IETF). También encontramos otros que, aunque no son específicos para el IoT y los dispositivos restringidos, son usados como SNMP o NETCONF.

1.2. Solución a abordar

CORECONF ha sido diseñado por el IETF para ser usado específicamente en nodos y redes restringidos. Su principal objetivo es dar solución a la gestión de este tipo de nodos, usando la menor cantidad de recursos y mensajes posibles. Actualmente estos borradores se actualizan cada 6 meses o menos y continúan añadiendo mejoras al mismo. Hasta su versión 10 recibió el nombre de CoMI y actualmente se encuentra en su versión 16 [2].

El IETF pretende que CORECONF se convierta en el estándar usado por la industria de este sector, ocupando el lugar en el mercado con el que se ha hecho LwM2M.

En este proyecto se pretende realizar un estudio sobre el borrador de CORECONF. Esto implicará la profundización en arquitecturas y protocolos relacionados (REST, YANG, CoAP), la puesta en marcha de una implementación y el estudio y comparativa con respecto a su competidor, el protocolo LwM2M de la OMA.

1.3. Estructura del documento

La estructura de este documento recogerá las tareas abordadas en la realización del trabajo:

En la sección dos, se hablará del estado del arte en el que se encuentran los protocolos de gestión.

En la sección tres, se abordará la base teórica con los conceptos necesarios para comprender CoMI, como la arquitectura REST, YANG y el protoclo CoAP usado en la capa inferior.

En la sección cuatro, se explicará la interfaz de gestión CORECONF tomando como base el borrador 16 de la IETF.

En la sección cinco, se mostrará la implementación realizada del protocolo, así como las pruebas al mismo.

En la sección seis, se realizará una comparativa con LwM2M y se explicará la implementación que se ha utilzado del mismo.

Por último, en la sección siete, se realizará un análisis de los resultados obtenidos de las implementaciones de ambos protocolos y se obtendrán las conclusiones de su comparativa.

2 ESTADO DEL ARTE

El buen discípulo pasa al maestro

-Dicho popular-

Debido a la existencia de múltiples protocolos disponibles para la gestión, existen diversos trabajos enfocados en cuáles son apropiados para su utilización en aplicaciones IoT y en redes con nodos restringidos.

Sinche et al. [3] analizan los siguientes protocolos de gestión: CMIP, SNMP, LNMP, NETCONF, RESTCONF, LwM2M, DM 2.0 y CoMI (renombrado CORECONF). El análisis se realiza a nivel característico sin realizar implementaciones. Considera a DM 2.0, LwM2M y CoMI lso tres a usar en nodos restringidos sin tene que limitar características del protocolo. También analiza la madurez de los mismos, destacando que LwM2M es el más utilizado hasta el momento y mencionando los esfuerzos que está haciendo la IETF por desarrollar un protocolo centrado en la gestión IoT que le pueda hacer competencia (CORECONF)

Parmigiani y Dettmar [4] comparan los dos protocolos más utilizados en la gestión de nodos restringidos, LwM2M y MQTT. Como resultado obtienen que LwM2M consume menos energía en payloads pequeños, pero más con payloads grandes como por ejemplo durante actualizaciones de firmware. También destacan que LwM2M utiliza menos bytes en sus comunicaciones, especialmente si no se utiliza ningún tipo de capa de seguridad.

Saif y Matrawy [5] comparan el rendimiento de tres protocolos utilizados para la configuración masiva de nodos IIoT: MQTT, SNMP y un protocolo que ellos mismos desarrollan con arquitectura publicador/suscriptor sobre HTTP/3. Descartan el uso de CORECONF por estar en borrador y no disponer de implementaciones open-source y a LwM2M por no poder automatizar tareas de gestión. Obtienen como resultado que su protocolo tiene un rendimiento similar al uso de MQTT y superior al de SNMP en entornos con un gran número de nodos.

Bhat et al. [6] realizan una comparativa entre los protocolos CORECONF, NETCONF y RESTCONF en entornos de dispostivos restringidos, realizando para ello una implementaciónd de los tres protocolos. Obtienen como resultado que CORECONF es el más adecuado en términos de consumo de energía y memoria.

Tomando estos trabajos como putnos de partida, teniendo en cuenta la popularida actual de LwM2M y que CORECONF parece el más indicado entre el resto a ser su competencia en entornos restringidos, este trabajo pretende compararlos no solo a nivel teórico sino también realizando implementaciones de ambos y comparar así sus prestaciones.

3 BASE TEÓRICA

El que resiste, gana

Camilo José Cela

En este capítulo se presentan las bases teóricas sobre las que se ha desarrollado el protocolo CORECONF. Se empieza con una introducción a la arquitectura REST en la que se basa tanto CoAP como CORECONF. A continuación, se presentan los protocolos relacionados como YANG y CoAP. Por último, se profundiza en CBOR, la codificación utilizada por CORECONF en sus comunicacioens.

3.1 REST

La Transferencia de Estado Representacional (REST) es un estilo de arquitectura para sistemas hipermedia distribuidos. Fue definida en el 2000 por Roy Thomas Fielding [7] y su principal objetivo era mejorar la escalabilidad y rendimiento que ofrecían hasta ese momento otras arquitecturas (principalmente SOAP).

3.1.1 Principios de la arquitectura REST

-Separación cliente-servidor: En el lado cliente, mejora la portabilidad de la interfaz de usuario, permitiendo la multiplataforma. En el lado servidor, mejora la escalabilidad al simplificar componentes y permitir la evolución por separado de las versiones clientes.

-Sin estado: Cada petición del cliente debe contener toda la información necesaria para que sea entendida por el servidor o algún intermediario. Esto mejora la visibilidad, al estar toda la información incluida en la petición, la fiabilidad porque se facilita la recuperación ante fallos parciales y la escalabilidad al permitir que el servidor libere recursos más rápidamente. Como desventaja, baja el rendimiento de la red al aumentarse la cantidad de datos repetidos en una serie de peticiones.

-Caché: Mejora la eficiencia de la red. Obliga a que las respuestas sean marcadas como cacheables o no-cacheables según puedan ser usadas como respuesta a peticiones equivalentes. El caché mejora la eficiencia, escalabilidad y rendimiento percibido por el cliente al disminuir el número de peticiones. Por el contrario, disminuye la fiabilidad si los datos en caché difieren de los del servidor.

-Interfaz uniforme. La principal característica de la arquitectura REST es la uniformidad en las interfaces de sus componentes, al definir 4 restricciones sobre ellas: la identificación de recursos, la manipulación de recursos mediante representaciones, el uso de mensajes autodescriptivos y el uso de hipermedia como motor del estado de la aplicación.

Estos principios son definidos por Fielding [7] y detallados en artículos posteriores enfocados en su utilidad para el diseño de servicios web [8] y [9].

3.1.2 Elementos de la arquitectura REST

3.1.2.1 Elementos de datos

Los componentes REST se comunican transfiriendo una representación de un recurso en un formato estandarizado que variará según la naturaleza del recurso.

Un recurso es una abstracción de la información en REST, cualquier cosa que podamos nombrar puede ser un recurso. Un recurso puede variar su información con el tiempo o ser estático. El recurso no es la información en si, sino su significado, por ejemplo, en un control de versiones de código fuente el recurso "última versión" puede contener la misma información en un momento puntual que el recurso "versión 1.1", aunque sean recursos distintos. Esta abstracción permite no tener que alterar todas las referencias al concepto cuando se produce un cambio en la información del mismo.

La forma de nombrar un recurso es su identificador de recurso, generalmente será una URL o URN.

Las representaciones de los recursos son secuencias de bytes más los metadatos que describen esos bytes. Una representación de un recurso indica su estado en un momento dado. El formato de los datos, conocido como *media type* indica al receptor de un mensaje como debe procesar los datos recibidos.

Los datos de control definen el propósito del mensaje, como por ejemplo la acción que queremos realizar (GET, PUT, DELETE, etc) o el significado de una respuesta (OK, Error: no soportado, Error: no encontrado, etc).

3.1.2.2 Conectores

Los conectores permiten el acceso a los recursos y transferir representaciones de los mismos. Al ser las interacciones sin estado, se evita que los conectores tengan que retener ningún tipo de información sobre el estado de la aplicación, permite procesar interacciones en paralelo y hace que toda la información que pueda influir en la reutilización de una respuesta esté presente en cada solicitud, facilitando el uso de respuestas en caché. Se definen los siguientes conectores en la arquitectura REST:

-Cliente: inicia la comunicación mediante una petición. Incluye los siguientes datos:

- Datos de control: indica la acción a realizar.
- Identificador de recurso: recurso o servicio al que se dirige la solicitud.
- Representación (opcional): para crear o modificar un recurso puede ser necesario incluir la información del mismo

-Servidor: acepta peticiones para proporcionar acceso a servicios o recursos. Incluye la siguiente información en sus respuestas:

- Datos de control: indica el significado de la respuesta (Ok, creado, error, no encontrado, etc).
- Metadatos del recurso (opcional): detalles sobre el recurso como la fecha de creación o modificación del recurso, tamaño, tipo de contenido, identificadores, etc.
- Representación (opcional): representación del estado del recurso en ese instante.

-Caché: puede incluirse en la interfaz del cliente para evitar la repetición de peticiones o en servidor para evitar el proceso de generación de la respuesta.

-Traductor de direcciones: traduce identificadores de recursos en direcciones de red.

3.1.2.3 Componentes

Se definen 4 componentes según el papel que cumplen en las aplicaciones:

-Agente de usuario: usa un conector cliente para realizar una petición y recibe la respuesta del servidor. El más común es un navegador web.

-Servidor de origen: usa un conector de servidor para gestionar un espacio de nombres al que pertenece un recurso y es el responsable último de servir una respuesta a las peticiones del agente de usuario. Los detalles de la implementación de cada recurso se esconden tras la interfaz que muestra al cliente.

-Proxy: intermediario en el lado del cliente que encapsula servicios, tipos de datos, seguridad, etc.

-Pasarela o proxy inverso (gateway): intermediario en el lado del servidor que encapsula servicios, tipos de datos, seguridad, etc.

En el diagrama 1 podemos ver un resumen da la arquitectura completa. El agente de usuario realiza las peticiones y recibe las respuestas a través de un proxy y el servidor de origen las recibe y envía a través de un Gateway o pasarela. Todas estas operaciones son transparentes para sus conectores que actúan como si estuviesen directamente conectados. Además, los conectores incluyen caché que detectan si las peticiones coinciden con alguna realizada con anterioridad y evitar duplicidades.

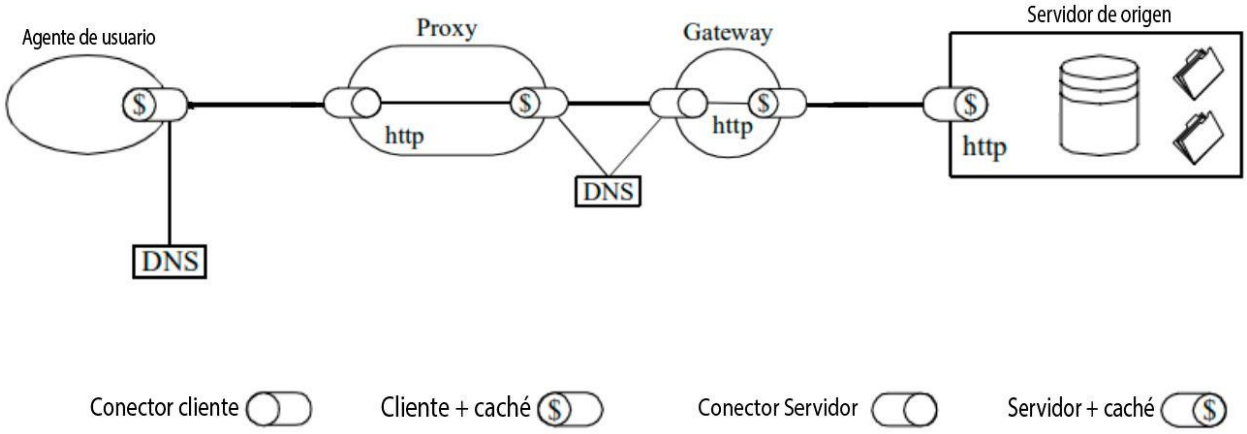

Figura 1. Diagrama de arquitectura REST.

Se han desarrollado distintos patrones para aplicar la arquitectura REST dando solución a problemáticas como la creación de recursos, operaciones que necesitan un mayor tiempo de procesamiento, actualizaciones de recursos concurrentes, etc. [10]

3.2 YANG

YANG es un lenguaje de modelado de datos definido en la RFC 7950 [11]. Es usado para modelar datos de configuración, estado, llamadas a procedimientos remoto y notificaciones.

Los modelos YANG siguen una organización jerárquica en árbol, en el cual cada nodo tiene o bien un valor o bien un conjunto de nodos hijos. Cada nodo cuenta con una descripción del mismo y las interacciones que permite. YANG cuenta con una serie de tipos definidos, sobre los cuales se pueden definir otros nuevos.

YANG estructura los modelos de datos en módulos y submódulos. Esto permite que un módulo extienda a otro importando el modelo de datos definido, lo cual facilita la interoperabildiad.

YANG es usado en protocolos como NETCONF o interfaces de gestión como CORECONF por poner el foco en la semántica y la facilidad de lectura de sus modelos por parte de humanos, en oposición a otros lenguajes de modelados [12] [13]

3.2.1 Nodos de datos YANG

YANG define 4 tipos principales de nodos de datos para el modelado:

- Nodos hoja: contienen un dato simple y no tiene hijos.
- Nodos de lista de hojas: define una secuencia de valores de un tipo en particular.
- Nodos contenedores: agrupan a otros nodos. No tienen ningún valor, solo nodos hijos.
- Nodos lista: definen una secuencia de listas en las que cada entrada es un contenedor identificado por el valor de su clave.

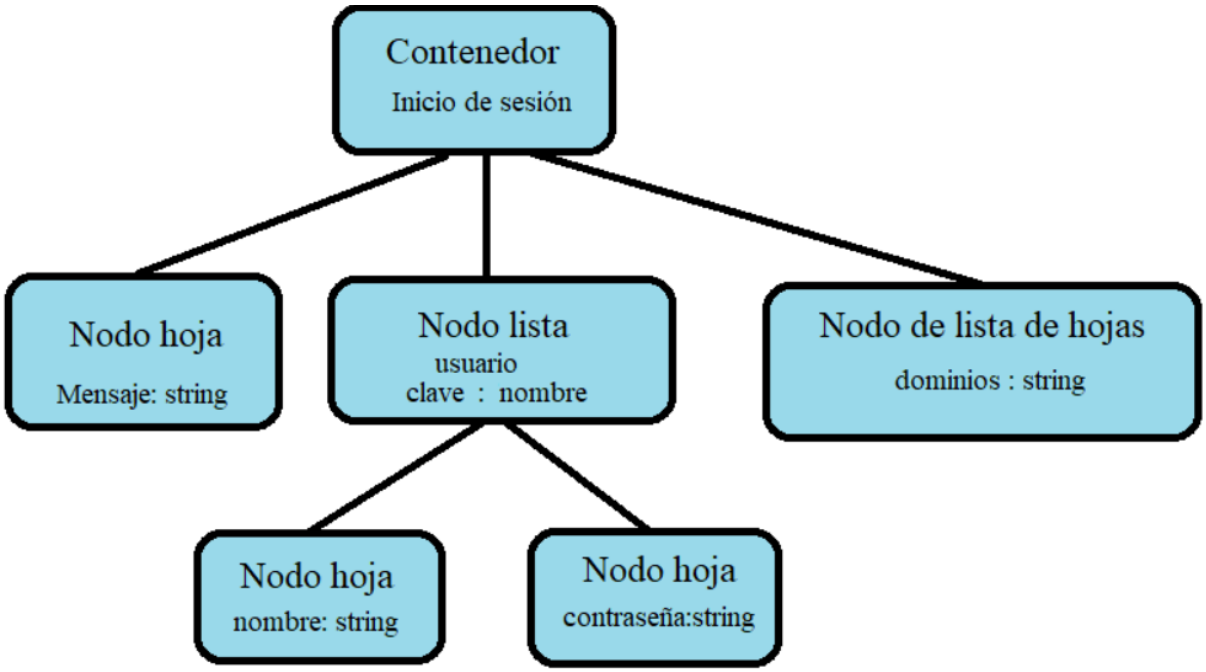

Figura 2. Nodos de datos YANG

3.2.2 Tipos de datos

YANG define un conjunto de datos similar a los de cualquier lenguaje de programación. En esencia, los tipos de datos vienen derivados de los usados anteriormente en los sistemas SMI.

Categoría	Tipo	Restricciones
Entero	{u}int {8,16,32,64}	rango
Decimales	decimal64	Rango, número de dígitos decimales
Cadenas	string	Longitud, patrón
Enumeración	Enumeration	Los propios de la enumeración
Booleanos y bits	Boolean, bits	
Binario	Binary	Longitud
Referencias	Leafref	Camino
Referencias	Identityref	Base
Referencias	Instance-identifier	
Otros	empty	

Tabla 1. Tipos incorporados de base por YANG

A partir de estos, se pueden construir nuevos tipos de datos. Para ello se hace uso de las siguientes instrucciones:

- Typedef: permite crear un tipo derivado de otro.

- Grouping: permite agrupar grupos de nodos para facilitar su reutilización como un tipo. Para reutilizar un grupo de nodos hacemos referencia a la instrucción grouping que lo define con una sentencia "uses".
- Choice – case: Permite a un nodo tener un tipo u otro en función de una condición.

3.2.3 Instrucciones YANG

Las instrucciones o declaraciones YANG se utilizan para definir la estructura y semántica de los datos, así como para especificar operaciones permitidas. A continuación, se describen algunas de las principales. Para más detalle sobre estas instrucciones se recomienda el tutorial sobre NETCONF y YANG [14] o la propia RFC de YANG [11]

-Module: La instrucción "module" define el nombre del módulo y agrupo al resto de declaraciones.

-Submodule: La instrucción "submodule" define el nombre de un submódulo y agrupa a las declaraciones que contiene. Los submódulos son utilizados para dividir un módulo complejo en partes más pequeñas.

-Namespace: La instrucción "namespace" define el espacio de nombre XML que usarán todos los identificadores definidos en el módulo.

-Import: La instrucción "import" hace disponible las definiciones de un módulo dentro de otro módulo o submódulo.

-Type: La instrucción "type" indica el tipo yang (base o derivado).

-Instrucciones para el modelado de datos: las instrucciones "leaf", "leaf-list", "container" y "list" definen los 4 tipos principales de nodos de datos usados en yang, los nodos hojas, nodos de lista de hojas, nodos contenedores y nodos lista respectivamente.

-Rpc: La instrucción "rpc" se usa para definir operaciones RPC. La instrucción va seguida de un identificador y y permite definir múltiples parámetros como sus entradas, salidas, descripciones para añadir contexto, referencias, etc.

-Notification: La instrucción "notification" se usa para definir nodos de tipo notificación. La instrucción va seguida de un identificador y permite definir información sobre la notificación (tipos, descripción, etc).

-When: La instrucción "when" convierte a la instrucción padre en condicional. El nodo definido por la información del padre solo es válido cuando la condición especificada con esta instrucción es verdadera.

-Anydata: La instrucción "anydata" define un nodo interior dentro del esquema y va seguida de un identificador. Se utiliza para representar un conjunto de nodos del cual no se conoce el modelo de datos en el momento del diseño del módulo.

-Anyxml: la instrucción "anyxml" define un nodo interior dentro del esquema y va seguida de un identificador. Se utiliza para representar un xml de formato desconocido al que podemos indicar restricciones con subinstrucciones como "mandatory" o "must".

-Augment: la instrucción "augment" permite añadir un módulo a la definición de un esquema. Va seguido de un argumento que identifica al nodo en el esquema

3.3 CoAP

El protocolo CoAP (Constrained Application Protocol) es un protocolo definido en la RFC 7252 [15] especializado en la transferencia de información entre nodos restringidos, generalmente nodos con pocos recursos y en redes con alto ratio de error. Este protocolo proporciona un modelo petición/respuesta, así como métodos para el descubrimiento de recursos, URIs para referirse a estos recursos y métodos para acceder a ellos o modificarlos. Uno de sus principales objetivos fue adaptar la arquitectura REST a los entornos restringidos [16]

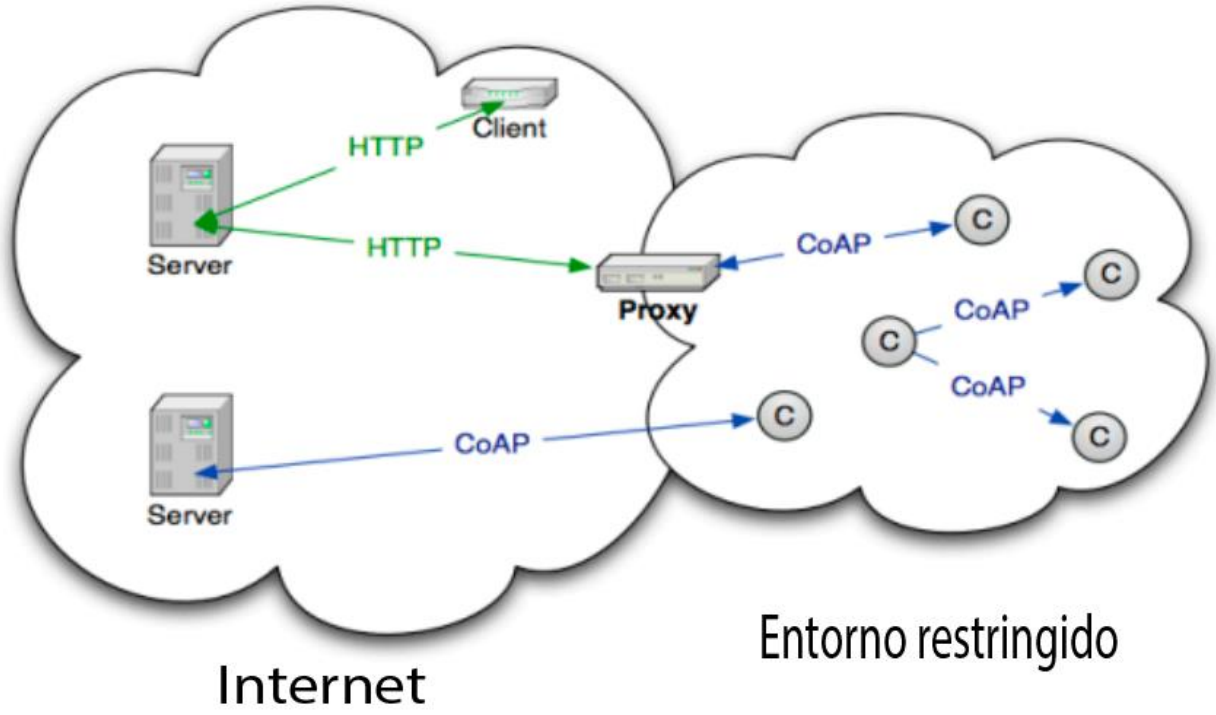

Figura 3. Arquitectura CoAP

CoAP es un protocolo de trasferencia diseñado para aplicaciones M2M entre dispositivos con poca RAM y en redes con una ratio de paquetes perdidos alto. Define 4 tipos de mensajes: confirmables, no confirmables, mensajes de confirmación y reset.

- Confirmable (CON), mensajes que necesitan ser confirmados por el receptor.
- No confirmable (NON), mensajes que no necesitan ser confirmados, útil en aplicaciones que repitan mensajes cada poco tiempo y no preocupa la pérdida de alguno (por ejemplo, la lectura de un sensor)
- Mensje de confirmación (ACK) confirma la recepción de un mensaje CON, aunque no indica el éxito o fracaso de la solicitud de ese mensaje. Las respuestas pueden ir junto al mensaje ACK o de manera separada en otro mensaje.
- Reset (Reset): Indica que se recibió un mensaje CON o NON pero no se pudo procesar correctamente por la falta de contexto.

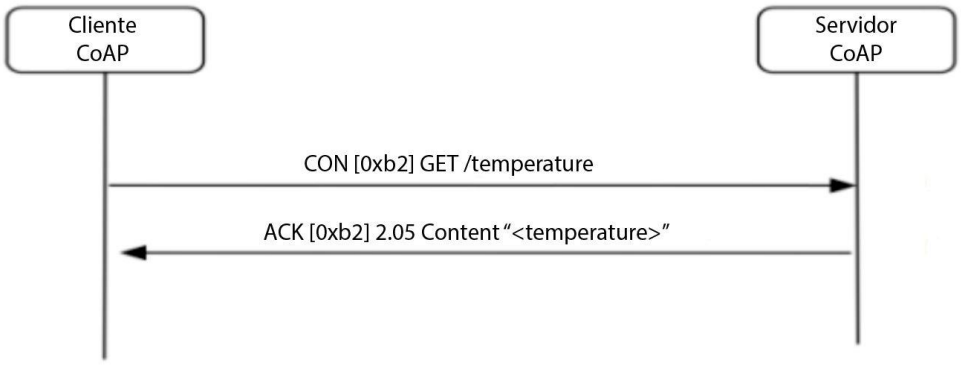

Figura 4. Ejemplo de petición confirmable CoAP

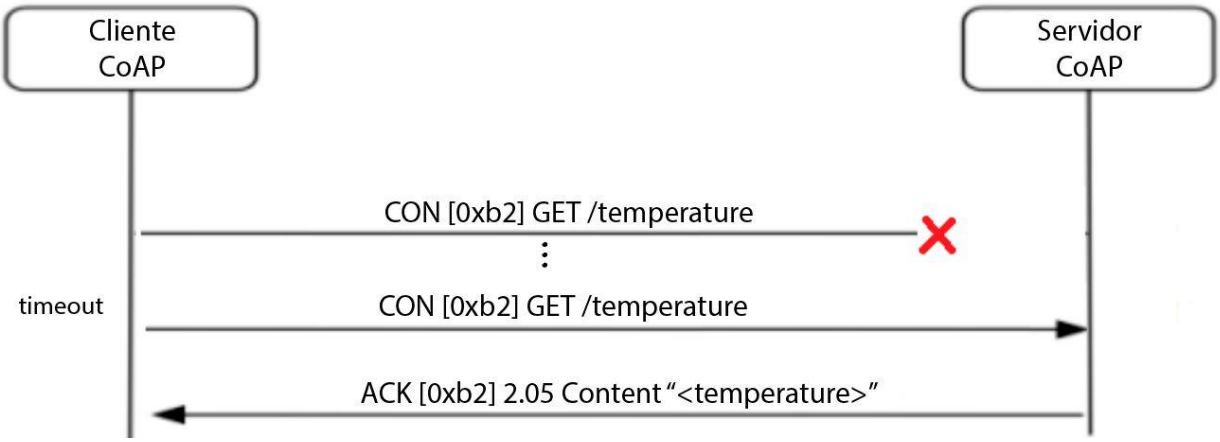

Figura 5. Repetición de petición CoAP tras un paquete perdido

CoAP utiliza el protocolo de transporte UDP para sus mensajes. Su cabecera está compuesta por un mínimo de 4 bytes aunque puede ser mayor según se describe:

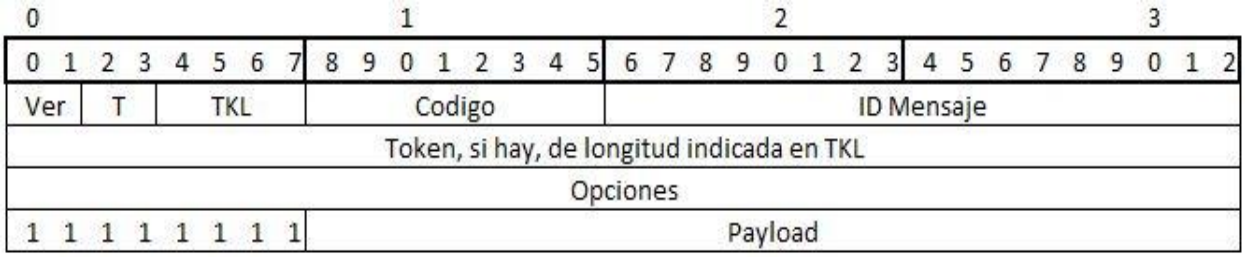

Figura 6. Cabecera CoAP

- Versión (Ver): Versión del protocolo.
- Tipo (T): entero de 2 bits que indica si el mensaje es confirmable (0), no confirmable (1), ACK (2) o Reset(3).
- Longitud de los Tokens (TKL): entero de 4 bits que indica la longitud de los tokens (0-8 bytes).
- Código: entero de 8 bits dividido en 3 y 5 bits. Los 3 primeros bits hacen referencia a petición (0), respuesta exitosa (2), respuesta errónea por error del cliente (4) o respuesta errónea por error en el servidor (5). El resto de bits son subclases de estos códigos.
- ID del mensaje: entero de 16 bits incrementado en orden para detectar duplicidades y ACKs.
- Token: opcional, usado para correlar peticiones y sus respuestas.
- Opciones: campo opcional de longitud indeterminada.
- Marcador de payload: Byte que será el último de las opciones con todos los bits a '1'.
- Payload: Carga del mensaje, si es necesaria.

3.3.1 Métodos soportados por CoAP

CoAP define 4 métodos similares a los establecidos en HTTP, todos ellos idempotentes:

- GET: código 0.01. Permite obtener un recurso identificado por una URI.
- POST: código 0.02. Su resultado depende de la implementaciónd el servidor, pero generalmente debería provocar la creción de un recurso.
- PUT: código 0.03. Crea o actualiza un recurso.

- DELETE: código 0.04. Elimina un recurso.

Posteriormente, se definieron 3 métodos más con el objetivo de poder acceder a partes de un recurso. Esto es útil para recursos grandes o complejos en los que es más eficaz obtener, crear, actualizar o borrar solo una parte del mismo. Los 3 métodos son:

- FETCH: código 0.05. Permite obtener un recurso o parte de él. Los parámetros para su obtención se incluyen en el payload del mensaje en lugar de en la URI como hace el método GET. Es idempotente.
- PATCH: código 0.06. Permite actualizar parcialmente uno o varios recursos. No es idempotente.
- iPATCH: código 0.07. Es la versión idempotente de PATCH.

Además, define los códigos de respuesta mostrados en la figura 7.

Code	Description
2.01	Created
2.02	Deleted
2.03	Valid
2.04	Changed
2.05	Content
4.00	Bad Request
4.01	Unauthorized
4.02	Bad Option
4.03	Forbidden
4.04	Not Found
4.05	Method Not Allowed
4.06	Not Acceptable
4.12	Precondition Failed
4.13	Request Entity Too Large
4.15	Unsupported Content-Format
5.00	Internal Server Error
5.01	Not Implemented
5.02	Bad Gateway
5.03	Service Unavailable
5.04	Gateway Timeout
5.05	Proxying Not Supported

Figura 7. Códigos de respuesta CoAP

3.4 CBOR

CBOR (Concise Binary Object Representation) es un formato de codificación de datos más compacto que XML y JSON. Está definido en la RFC 7409 [17]. Los principales objetivos de CBOR son:

- Codificar sin ambigüedad los formatos de datos más usados en Internet.

- Permitir implementaciones en equipos de bajo recursos.
- Decodificar los datos sin un descriptor de esquema.

El byte inicial de cada elemento contiene información sobre el tipo de elemento (con sus 3 primeros bits) e información adicional o el propio dato en si en los bits restantes.

3.4.1 Tipos principales

Se definen un total de 7 tipos principales:

- Tipo 0 (000): entero sin signo. Los 5 bits adicionales hacen referencia al entero, si su valor va de 0 a 23 o a la longitud adicional, siendo 24 un uint_8, 25 uint_16, 26 uint_32 y 27 uint_64.

Ejemplo 3–1. *Ejemplos de enteros sin signo en codificación CBOR*

El número 10 necesita solo un byte: 0000 1010

El 35 necesitaría de 2 bytes: 0001 1000 0010 0011

- Tipo 1 (001): entero negativo. Los 5 bits adicionales hacen referencia al valor del número, si su valor va de 0 a 23 o a la longitud adicional, siendo 24 un int_8, 25 int_16, 26 int_32 y 27 int_64.

Ejemplo 3–2. *Ejemplos de enteros negativos en codificación CBOR*

El número -10 necesita solo un byte: 0010 1001

El -35 necesitaría de 2 bytes: 0011 1000 0010 0010

- Tipo 2 (010): cadena de bytes. La longitud de la cadena en bytes se indica siguiendo las reglas de codificación para los enteros sin signo del tipo 0 y a continuación se añaden la propia cadena.

Ejemplo 3–3. *Ejemplos de cadenas de bytes en codificación CBOR*

Cadena de 4 bytes de longitud. Se indica directamente el número 4: 0100 0100 [cadena de 4 bytes]

Cadena de 35 bytes de longitud. En el primer byte se indica que el número de la longitud necesita 1 byte (uint_8) y en el segundo byte se indica la longitud de la cadena: 0101 1000 0010 0011 [cadena de 35 bytes]

- Tipo 3 (011): cadena de texto. Cadena codificada específicamente en UTF-8. El formato es idéntico al del tipo 2.

Ejemplo 3–4. *Ejemplos de cadenas de texto en codificación CBOR*

Cadena de 4 bytes. Se indica directamente el número 4: 0100 0100 [cadena de 4 bytes en UTF-8]

Cadena de 35 byte. En el primer byte se indica que el número de la longitud necesita 1 byte (uint_8) y en el segundo byte se indica la longitud de la cadena: 0101 1000 0010 0011 [cadena de 35 bytes]

- Tipo 4 (100): tabla, lista o secuencia. El número de elementos de la tabla se indica de manera análoga a la longitud de las cadenas de tipo 2 y a continuación se indican los elementos.

Ejemplo 3–5. *Ejemplos de tablas en codificación CBOR*

Tabla de 2 elementos: 1000 0010 [2 elementos de la tabla]

Tabla de 35 elementos: 1001 1000 0010 0011 [35 elementos de la tabla]

- Tipo 5 (101): mapas, diccionarios u objetos json. Los mapas se componen de parejas de elementos o tuplas clave, valor. El número de elementos del mapa se indica de manera análoga a la longitud de las cadenas de tipo 2, seguido de los elementos del mapa.

Ejemplo 3–6. *Ejemplos de mapas en codificación CBOR*

Mapa de 2 elementos: 1010 0010 [clave 1, valor 1, clave 2, valor 2]

Mapa de 35 elementos: 1011 1000 0010 0011 [clave 1, valor1, clave 2, valor 2, … clave 35, valor 35]

- Tipo 6 (110): etiquetas de significado adicional para los otros tipos.

- Tipo 7 (111): números de punto flotante y valores sin contenido como verdadero, falso y null. El número de bytes utilizados para representar el número de punto flotante se indica como en los enteros sin signo, mientras que los valores sin contenido se indican en los 5 propios bits restantes.

Ejemplo 3–7. *Ejemplos valores sin contenido en CBOR*

Falso (20): 1111 0100

Verdadero (21): 1111 0101

Null (22): 1111 0110

---

4 CORECONF

La simplicidad es la máxima sofisticación.

- Leonardo Da Vinci -

La Interfaz de Gestión sobre CoAP (CORECONF) es una interfaz de gestión de red orientada a redes y dispositivo de recursos limitados (baja batería, capacidad de procesmiento, etc). CORECONF utiliza CoAP para acceder a recursos especificados en YANG utilizando CBOR+yang como representación. Está siendo desarrollado por el IETF y cada 6 meses aproximadamente elaboran un nuevo borrador actualizado. El último borrador hasta la fecha es el número 16 [2]

4.1 Arquitectura

CORECONF sigue una arquitectura REST, en la que se define un cliente y un servidor. El cliente realizará las peticiones y modificaciones de los recursos definidos en modulos YANG utilizando para ello CoAP. El servidor se encarga de recibir las peticiones y responder. El servidor contará con al menos un almacén de datos del que obtener y modificar los recursos implementados de los módulos YANG.

Figura 8 . Esquema cliente y servidor CORECONF

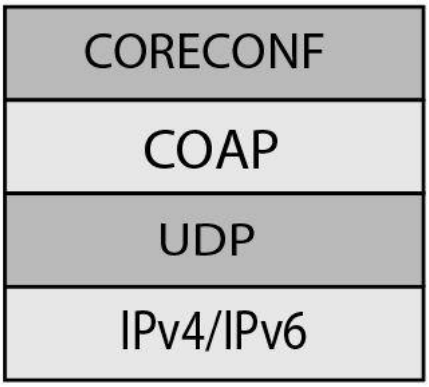

Figura 9. Torre de protocolos CORECONF

4.2 Modelo de datos

El modelo de datos de CORECONF se basa en el uso de módulos YANG. En YANG se define si los objetos son de solo lectura, lectura/escritura o ejecutables (RPC). Es importante destacar que para simplificar el protocolo y facilitar su implementación en dispostitivos de recursos limitados, se realizan cambios en los identificadores YANG y se simplifican los almacenes de datos.

4.2.1 Compresión de los identificadores YANG

En las especificaciones de los módulos YANG, los identificadores son cadenas de caracteres. Para reducir el tamaño de las peticiones, CORECONF utiliza sus equivalentes identificadores numéricos o SID (del inglés, YANG Schema Item iDentifier). Estos SID además son codificados en CBOR como números enteros sin signo.

4.2.2 Almacén de datos

CORECONF permite la implementción de un único almacén de datos que contenga tanto datos de configuración como de operación. Esto permite además que los cambios en la configuración tengan un retraso mínimo, al tratarse igual que los datos de operación. También se contempla el uso de almacenes separados para configuración y operación siguiendo el modelo definido en la Arquitectura de Almacenes de Datos para la Gestión de Redes (NMDA) definido en la RFC8342 [19].

4.2.3 Formato de las respuestas

En la siguiente tabla se define el formato del contenido de los distintos métodos CORECONF.

Método	Recurso	Formato del contenido
Petición FETCH	Almacén de datos	/application/yang-identifiers+cbor-seq
Respuesta FETCH	Almacén de datos	/application/yang-instances+cbor-seq
Petición iPATCH	Almacén de datos	/application/yang-instances+cbor-seq
Respuesta GET	Notificaciones	/application/yang-instances+cbor-seq
Petición POST	Rpc, acción	/application/yang-data+cbor-seq
Respuesta POST	Rpc, acción	/application/yang-data+cbor-seq

Tabla 2. Formato del contenido de los métodos CORECONF

Todos los formatos están basados en el formato YANG mapeado en CBOR definido por el IETF [17]. A continuación, se explican los 3 formatos:

-Application/yang-data+cbor-seq: Mapa CBOR en el que las claves son los SID de los nodos de datos y los valores son los valores de los nodos de datos.

-Application/yang-identifier+cbor-seq. Tabla CBOR de identificadores de instancia, codificados como enteros sin signo. Es una lista de identificadores de instancias de datos de un almacén de datos.

-Application/yang-instances+cbor-seq. Tabla CBOR de mapas CBOR, cuyas claves son identificadores de

instancias de los nodos de datos y sus valores los valores de las instancias.

Actualmente estos formatos no tienen asociado un identificador de contenido de formato CoAP. Solo está definido el formato 80 como application/cbor.

4.3 Interfaz CoAP de gestión

CORECONF utiliza métodos CoAP para retirar y editar datos, realizar notificaciones y llamar a procedimientos remotos. Las diferentes posibilidades a la hora de realizar estas acciones son explicadas en los siguientes subapartados.

Los recursos CoAP que implementen el protocolo CORECONF deben soportar al menos un recurso de gestión descubrible, de forma que los clientes puedan saber a qué recursos pueden acceder. Es recomendable que este recurso tenga la ruta /c. También se recomienda que los eventos tengan una ruta diferente, /s.

4.3.1 Retirada de datos

Para la retirada de datos, CORECONF utiliza el método FETCH.

FETCH es utilizado para obtener el valor de una o varias instancias de un nodo de datos. Las instancias a obtener se especifican en el payload del mensaje, codificando los SIDs en una secuencia CBOR.

Formato de la petición:

FETCH /c

(Formato del contenido: application/yang-identifier+cbor-seq)

Formato de la respuesta:

2.05 Contenido (Formato: application/yang-instance+cbor-seq)

Ejemplo 4–1. *Retirada del dato /ietf-system:system/location*

El SID correspondiente al identificador yang /ietf-system:system/location es el 1753.

Petición: FETCH /c

(Formato del contenido: application/yang-identifier+cbor-seq)

{1753}

Respuesta: 2.05 (Formato application/yang-instance+cbor-seq)

[

1753 : “Sevilla”

]

Si se quiere incluir algún parámetro como clave para identificar un nodo de dato, se añade en la secuencia de CBOR, sustituyendo el entero por un mapa {SID (entero), clave (tipo de la clave).

FETCH además permite retirar múltiples datos en una misma llamada, para ello se añaden los SIDs en la secuencia CBOR

Ejemplo 4–3. *Retirada de la interfaz eth0 definida en /ietf-interfaces/interface y de ietf-system/contact*

El SID correspondiente al identificador yang /ietf-interfaces:interface es el 1553. Si el dispositivo tiene varias interfaces le indicamos que solo queremos la información de la interfaz cuyo nombre es “eth0”. Se elige como clave el nombre ya que es la primera clave definida en el móduyo YANG para los nodos de tipo interfaz. El SID de contact es el 1741.

Petición: FETCH /c/

(Formato del contenido: application/yang-identifier+cbor-seq)

```
[
   {1533, “eth0”},                    // SID interface = 1533, nombre de la interfaz “eth0”
    1741                              // SID contact = 1741
]
```

Respuesta: 2.05 (application/yang-identifier+cbor-seq)

```
{
1553 :  [
{
   4: “eth0”,
   1: “Ethernet adaptor”,
   5: 1890,
   2: true,
  11: 3
],
1741, “Administrador”
}
```

4.3.2 Edición de datos

El contenido de los almacenes de datos puede ser creado, modificado y borrado. Para ellos CORECONF utiliza el método iPATCH.

- Crear o reemplazar una instancia de un recurso. La instancia del nodo de datos es enviada en el payload. Se codifica en CBOR como una secuencia de mapas SID-valor.

Formato:

iPATCH /c (Formato: application/yang-instances+cbor-seq).

2.01 Creado

2.04 Modificado

Ejemplo 4–4 . *Edición del dato /ietf-system:system/location*

El SID correspondiente al identificador yang /ietf-system:system/location es el 1753.

Petición: iPATCH /c (Formato: application/yang-instances+cbor-seq)

```
[
{1753 : “Sevilla”}
]
```

Respuesta: 2.04 Changed

- Borrar una instancia de un recurso. La instancia del nodo de datos es enviada en el payload. Se codifica en

CBOR como una secuencia de mapas SID-null

Formato:

iPATCH /c (Formato: application/yang-instances+cbor-seq).

2.02 Borrado

Ejemplo 4–5. *Borrado de la instancia del objeto /ietf-system:system/location*

Petición: Petición: iPATCH /c (Formato: application/yang-instances+cbor-seq)

[

{1753 : null}

]

Respuesta: 2.02 Deleted

El uso de iPATCH permite realizar varias operaciones de creación, edición y borrado en una sola petición, codificando los valores en la secuencia CBOR que forma el payload.

Ejemplo 4–6. *Edición de la instancia del objeto /ietf-system:system/location y borrado de /ietf-system:system/contact*

El SID correspondiente al identificador yang /ietf-system:system/location es el 1753 y el identificador yang /ietf-system:system/contact es el 1741

Petición: POST /c/Bz (Formato: applicaiton/yang-data+cbor)

Petición: Petición: iPATCH /c (Formato: application/yang-instances+cbor-seq)

[

{1753 : “Sevilla”},

{1741 : null}

]

Respuesta: 2.04 Changed

4.3.3 Notificaciones

El método GET es también utilizado para suscribirse a un flujo de notificaciones. Para ello el campo Observar debe ir a 0 y se debe especificar un recurso que permita notificaciones por parte del servidor CORECONF. Cada cuanto tiempo deben enviarse las notificaciones o bajo qué circunstancias y el cómo indicarlo se deja abierto a las implementaciones.

4.3.4 Llamadas a procedimientos remoto (RPC)

En los módulos YANG se definen llamadas a procedimientos remotos o acciones, que implican una ejecución por parte del servidor. En CORECONF, se utiliza el método POST. El payload se codifica en CBOR y en el se envía el SID y los datos de entrada si son necesarios.

Formato:

POST /c (Formato: application/yang-instances+cbor-seq)

2.05 Content (Formato: application/yang-instances+cbor-seq)

4.4 Seguridad

CORECONF confía su seguridad en los mecanismos de CoAP, incluyendo DTLS para el acceso protegido a recursos, además de otros mecanismos de autenticación y autorización. Se recomienda el uso de OSCORE, método de seguridad para entornos REST con nodos restringidos definido en la RFC 8613 [20], que proporciona protección de extremo a extremo entre cliente y servidor CoAP.

5 LIGHTWEIGHT M2M

I'm not like them, but i can pretend.

- Nirvana -

EL OMA Lightweight Machine to Machine (LwM2M) es un protocolo para la gestión de dispositivos embebidos, sensores y en general, dispositivos emergentes en el Internet de las Cosas (IotT) [21]. El protocolo LwM2M utiliza CoAP como protocolo de transporte sobre UDP, TCP o SMS y actualmente es el principal competidor en el mercado de CORECONF [22]. En este capítulo se tratará su arquitectura y modelo de datos.

5.1 Arquitectura

El protocolo LwM2M define tres tipos de entidades: el cliente LwM2M, el servidor de arranque LwM2M y el servidor LwM2M.

Los clientes LwM2M se comunican con ambos tipos de servidores para permitir a los servidores LwM2M gsetionrar y monitorizar sus recursos.

El servidor de arranque LwM2M se encarga de establecer los parámetros de configuración inicial de los clientes LwM2M.

El servidor LwM2M gestiona y monitoriza a clientes LwM2M.

LwM2M define la capa de aplicación de comunicación entre un cliente y un servidor. Los clientes suelen residir en dispositivos con recursos limitados mientras que el servidor, usado para gestionar a uno o varios clientes, suele formar parte de equipos de mayor potencia e incluso centros de datos. Un esquema de la arquitectura puede verse en la figura 4.

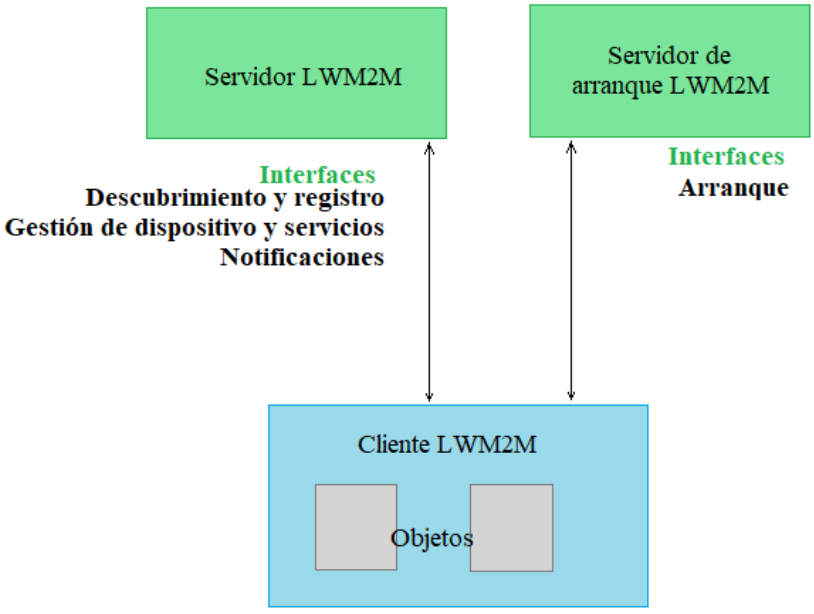

Figura 10. Arquitectura LwM2M

5.2 Interfaces

Se definen 4 interfaces lógicas entre el servidor y cliente LwM2M. Los roles de cliente y servidor CoAP van cambiando según la interfaz y operación concreta.

5.2.1 Interfaz de arranque

En la interfaz de arranque, tanto el servidor como el cliente LwM2M actúan como cliente y servidor CoAP. En esta interfaz se definen una serie de comandos para realizar la configuración inicial del cliente LwM2M y para que pueda contactar con servidores LwM2M. Cualquiera puede iniciar el establecimiento de una conexión en esta interfaz.

A través de esta interfaz, el cliente LwM2M obtiene la información de al menos un servidor LwM2M. Además, puede obtener datos adicionales como niformación acerca de la seguridad o instancias de objetos adicionales.

La interfaz de arranque define las siguientes operaciones: Petición de arranque, Fin de arranque, Descubrir, Leer y Escribir,

5.2.1.1 Petición de Arranque

La operación de Petición de Arranque (Bootstrap Request) es representada por un mensaje POST /bs?ep?=[Identificador del cliente] enviado a un servidor de arranque. Le informa de su existencia en la red.

5.2.1.2 Fin de Arranque

La operación Fin de Arraque (Bootstrap Finish) es representada por un mensaje POST /bs. Tras recibir el mensaje, el cliente puede conectarse a servidores LwM2M.

5.2.1.3 Descubrir

La operación Descubrir (Bootstrap Discover) es representada por un mensaje GET. El cliente responde con información sobre el modelo de datos del que dispone.

5.2.1.4 Escribir

La operación de Escribir (Bootstrap Write) es enviada por el servidor de arranque mediante un mensaje PUT. Permite crear y escribir instancias de objetos y recursos.

5.2.1.5 Borrar

La operación Borrar (Bootstrap Delete) es enviada por el servidor de arranque mediante un mensaje DELETE para borrar objetos y recursos.

5.2.2 Interfaz de descubrimiento de dispositivo y registro

En la interfaz de registro, el servidor LwM2M actúa como servidor CoAP y el cliente como cliente CoAP. Es utilizada por el cliente para informar sobre su presencia y disponibilidad al servidor.

En esta interfaz, se permiten las operaciones: registrar, actualizar y borrar registro

5.2.2.1 Registrar

La operación Registrar (Register) es representada por un mensaje CoAP POST /rd?... El cliente la envía al iniciarse e informa al servidor de su disponibilidad para recibir mensajes en la interfaz de gestión de dispositivo y habilitación de servicios y presenta su modelo de datos. Además, permite al servidor obtener la ip y el puerto en el que realizar las peticiones. El servidor responde con una URL a la que enviar las operaciones de tipo Actualizar.

5.2.2.2 Actualizar

La operación Actualizar (Update) es representada por un mensaje CoAP POST enviada a una URL devuelta por la respuesta a u la operación Registrar. Se utiliza periódicamente para informar al servidor de que el cliente sigue estando disponible o cuando la información incluida en la operación Registrar cambia. Si el campo tiempo de vida de la conexión es superado sin enviar un mensaje de actualizar, el servidor entenderá que el cliente no está disponible y será necesario realizar un nuevo registro para realizar cualquier operación.

5.2.2.3 Borrar Registro

La operación Borrar Registro (De-Register) es representada por un mensaje CoAP DELETE. El cliente la envía si determina que se apagará en breve finalizando la asociación con el servidor.

5.2.3 Interfaz de gestión de dispositivo y habilitación de servicios

En la interfaz de gestión de dispositivo, el cliente LwM2M actúa como servidor CoAP y recibe peticiones del servidor LwM2M, que actúa como cliente CoAP.

En esta interfaz, se permiten las siguientes operaciones: Descubrir, Leer, Escribir, Ejecutar, Escribir Atributos, Crear y Borrar.

5.2.3.1 Descubrir

La operación Descubrir (Discover) es representada por un mensaje CoAP GET Accept: application/link-format. El cliente devuelve una lista con todos los objetos, recursos e instancias disponibles.

5.2.3.2 Leer

La operación Leer (Read) es representada por un mensaje CoAP GET. Como argumento se pasa un objeto, recurso o instancia cuyo valor será devuelto por el cliente.

5.2.3.3 Escribir

La operación Escribir (Write) puede tener dos significados, según el tipo de mensaje CoAP utilizado:

-Reemplazar. Representado por un PUT Objeto/Instancia/Recurso. Reemplaza por completo el dato accedido.

-Actualizar. Representado por POST/Objeto/Instancia. Solo reemplaza los recursos indicados en el payload del mensaje.

5.2.3.4 Ejecutar

La operación Ejecutar (Execute) es representada por un mensaje CoAP POST /Objeto/Instancia/Recurso.

5.2.3.5 Escribir Atributos

La operación Escribir Atributos (Write Attributes) es representada por un mensaje CoAP PUT sin formato del contenido. En su lugar, se pasan como argumentos en la cadena de la URL.

5.2.3.6 Crear

La operación Crear (Create) es representada por un mensaje CoAP POST indicando como ruta la ruta raíz. Esta operación crea nuevas instancias de objetos.

5.2.3.7 Borrar

La operación Borrar (Delete) es representada por un mensaje CoAP DELETE y permite borrar instancias de objetos.

5.2.4 Interfaz de notificaciones de información

La interfaz de avisos de información, extiende a la de gestión, por lo que los roles de cliente y servidor son los mismos. Permite al servidor obtener periódicamente actualizaciones de valores del modelo de datos del cliente. Está basado en el comando Observar (Observe) de CoAP.

En esta interfaz, se permiten las siguientes operaciones: Iniciar Observación, Cancelar Observación, y Notificar.

5.2.4.1 Iniciar observación

La operación Iniciar observación (Observe) se consigue mediante una operación de Leer con la opción observar a 0. El cliente comenzará a notificar al servidor cuando el valor cambie o de manera periódica.

5.2.4.2 Cancelar observación

La operación Cancelar Observación (Cancel Observation) se consigue mediante una operación de Leer con la opción observar a 1 o respondiendo a la operación Notificar con un CoAP RESET. El cliente dejará de notificar al servidor de los cambios en el valor observado.

5.2.4.3 Notificar

La operación Notificar (Notify) es representada por la respuesta asíncrona de CoAP Notify. Es una respuesta repetida a una petición Read de forma periódica o cuando el valor observado cambia.

5.3 Modelo de datos

El modelo de datos del protocolo LwM2M se organiza en forma de árbol de tres niveles. Las entidades de cada nivel son identificadas con números de 2 bytes. Estas entidades son los objetos, recursos e instancias de recursos. Estos datos pueden hacer referencia tanto a datos de configuración, datos de estado y de aplicación.

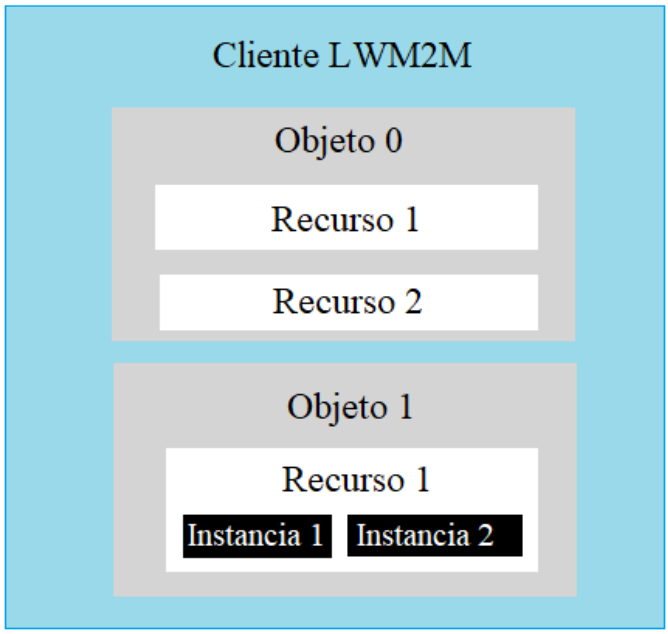

Figura 11. Modelo de datos LwM2M

Los datos en LwM2M pueden ser transferidos usando uno de los siguientes formatos: Texto plano, CoRE Link Format, flujo de octetos, TLV, JSON, SenML CBOR o SenML JSON.

5.3.1 Objetos

El modelo de datos LwM2M y la autoridad de registro de nombres de la OMA proporcionan la lista de objetos disponibles. Es posible crear tus propios objetos y pedir que sean registrados o usarlos sin registrar. La definición

de un objeto se muestra en la siguiente tabla.

Nombre	ID del objeto	Instancias	Obligatorio	URN del objeto
Nombre del objeto	Entero sin signo de 16 bits.	Multiple/Única	Obligatorio/Opcional	urn:oma:lwm2m:{oma,ext,x}:{Id del objeto}

Tabla 3. Definición de un objeto LwM2M

El campo {oma,ext,x} de la URN depende del ID del objeto:

-ID entre 0 y 1023: Se utiliza oma. Están reservados para los objetos definidos por la OMA.

-ID entre 1024 y 2047. No se usa ninguno, al estar el rango reservado para futuros usos.

-ID entre 2048 y 10240. Se utiliza ext. Son objetos registrados por organizaciones de estándares diferentes a la OMA.

-ID superior a 10240. Utilizados por compañías e individuales. El rango 32769-42768 es privado y no se publicará ningún objeto con esos identificadores.

La lista con todos los objetos definidos actualmente puede encontrarse en el registro LwM2M de la OMA [9]

5.3.2 Recursos

Los recursos son la unidad lógica del modelo de datos LwM2M. Todos los objetos están formados por uno o varios recursos. Los recursos pueden contener un valor, si es de lectura y/o escritura o puede provocar una acción si es un ejecutable. Los campos que definen a un recurso se muestran en la siguiente tabla.

ID	Nombre	Operaciones	Instancias	Obligatorio	Tipo	Rango o enumeración	Unidades	Descripción
Entero sin signo de 16 bits.	Nombre del recurso	R – Lectura W – Escritura E – Ejecutable	Multiple o Único	Obligatorio u opcional	String, Integer, Float, Boolean, Opaque, Time, ObjInk, none	Límites del valor del recurso	Unidad del valor del recurso	Descripción del recurso

Tabla 4. Definición de un recurso LwM2M

El ID del recurso lo clasifica en 3 clases:

-Recursos comunes: ID en el rango 0-2047.

-Recursos reusables: ID en el rango 2048-26240. Recursos registrados por compañías y organizaciones de estándar.

-Recursos privados: ID en el rango 26241-32768. No es necesario registrarlo en la OMA y están abiertos a ser reutilizados.

6 IMPLEMENTACIÓN

I have a dream.

- Martin Luther King -

En este capítulo se describirán las implementaciones realizadas tanto del protocolo CORECONF como de LwM2M. Se comentarán las librerías open-source utilizadas para el desarrollo, el hardware y las pruebas realizadas para comprobar el funcionamiento correcto de ambas implementaciones. También se hará hincapié en las limitaciones encontradas.

6.1 Hardware

El objetivo de ambos protocolos es facilitar la gestión de nodos restringidos. Es por ello que se ha decidido utilizar un nodo de bajas prestaciones y bajo coste, el Node MCUv3. Este nodo cuenta con 80kB de memoria RAM y 74kB de memoria de programa.

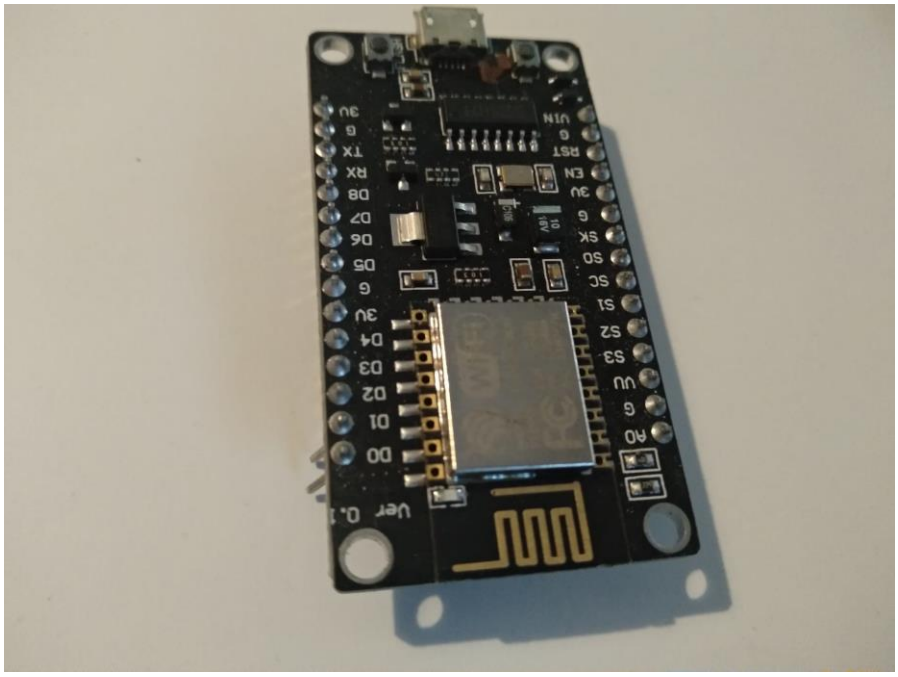

Figura 12. Node MCUv3

El Node MCUv3 contiene un módulo ESP8266, conocido como módulo Wi-Fi de Arduino. Permite su programación en el dialecto de C utilizado por Arduino y es compatible con librarías desarrolladas en C y C++.

6.2 CORECONF.

6.2.1 Escenario

Para la implementación de CORECONF se utilizará el siguiente escenario:

-Servidor: Node MCUv3 con distintos objetos de gestión implementados (dependiendo de la prueba). Recibirá las peticiones y responderá

-Cliente: PC que ejecutará peticiones.

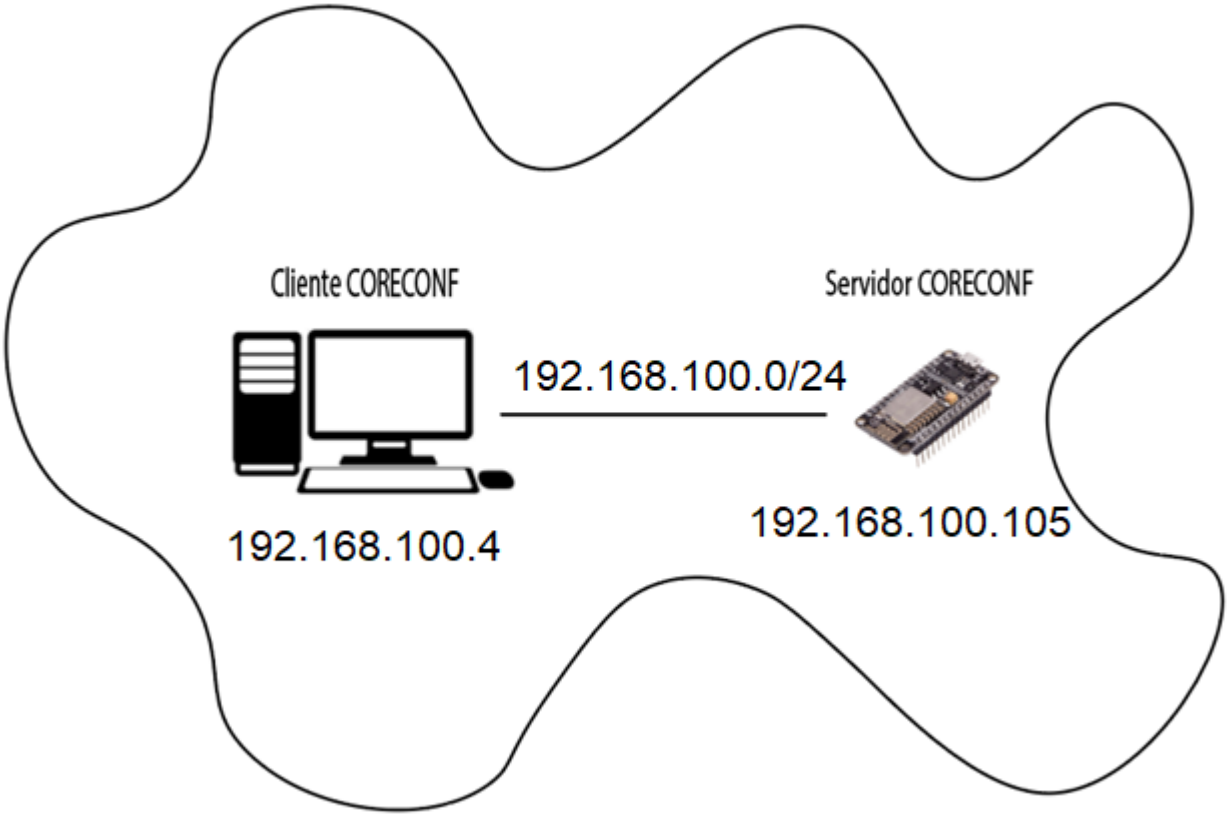

Figura 13. Escenario pruebas CORECONF

6.2.2 Servidor

Para realizar la implementación del lado servidor, se han utilizado las siguientes librerías open-source:

-TinyCBOR: Escrita en C, permite crear los mapas necesarios para la codificación de los contenidos de los mensajes CoAP.

-microCoAP: librería escrita en C para Arduino, que permite el envío y recepción de mensajes GET, PUT y POST, FETCH, iPATCH y PATCH del protocolo CoAP. Esta librería ha sido modificada para incluir CBOR como formato del contenido de los mensajes.

Los nodos de datos implementados son los siguientes:

- yang /ietf-system:system/contact. SID=1741. Cadena de lectura y escritura. Indica el contacto responsable del dispostivo.

```
leaf contact {
```

```
      type string;
      description
        "The administrator contact information for the system.

         A server implementation MAY map this leaf to the sysContact
         MIB object.  Such an implementation needs to use some
         mechanism to handle the differences in size and characters
         allowed between this leaf and sysContact.  The definition of
         such a mechanism is outside the scope of this document.";
      reference
        "RFC 3418: Management Information Base (MIB) for the
                   Simple Network Management Protocol (SNMP)
                   SNMPv2-MIB.sysContact";
    }
```

- yang /ietf-system:system/location. SID=1753. Cadena de lectura y escritura. Indica la localización del dispositivo.

```
leaf location {
      type string;
      description
        "The system location.

         A server implementation MAY map this leaf to the sysLocation
         MIB object.  Such an implementation needs to use some
         mechanism to handle the differences in size and characters
         allowed between this leaf and sysLocation.  The definition
         of such a mechanism is outside the scope of this document.";
      reference
        "RFC 3418: Management Information Base (MIB) for the
                   Simple Network Management Protocol (SNMP)
                   SNMPv2-MIB.sysLocation";
    }
```

-/ietf-interfaces:interfaces-state/interface/ietf-ip:ipv4/address/ip. SID=1605 De lectura y escritura. Hace referencia a la ip de una interfaz del dispositivo.

```
leaf ip {
          type inet:ipv4-address-no-zone;
          description
            "The IPv4 address on the interface.";
        }
```

Como acciones se han definido 2 RPCs:

-/ietf-system:system-restart. SID=1718. Reinicia el dispositivo.

```
 rpc system-restart {
```

```
    nacm:default-deny-all;
    description
      "Request that the entire system be restarted immediately.
       A server SHOULD send an rpc reply to the client before
       restarting the system.";
  }
```

-Luz: SID=9999. No está definida en ningún módulo YANG. Ha sido usada para utilizar una RPC con dato de entrada para apagar/encender la luz.

Además, también se ha definido el recurso obligatorio de gestión descubrible. Generalmente llamado ".well-known/core" en CoAP y en este caso situado en la ruta /c.

En la gestión, generalmente los recursos no son modificados constantemente. Por ello se ha decidido que la forma de implementar estos objetos de gestión sea mediante la escritura y lectura de la EEPROM integrada en el Node MCU v3. Se ha limitado por ello el tamaño de las cadenas de los objetos system/location a 120 caracteres y system/contact a 20 caracteres, ocupando las posiciones 0-19 y 20-139 respectivamente. La dirección IP se almacena en las posiciones 140-143, permitiendo así mantenerla al reiniciar el dispositivo.

Para obtener datos sobre la memoria de programa necesaria al implementar objetos, se ha hecho una segunda versión del servidor CORECONF con 100 contenedores iguales, cada uno conteniendo 2 hojas de tipo cadena, una de solo lectura y otra de lectura y escritura. Cada uno de los 100 contenedores tienen la siguiente definición.

```
container test {
      leaf test_RW{
            type string;
            description
              "Cadena de lectura y escritura".
      }
      leaf test_RO{
            type string;
            description
              "Cadena de solo lectura".
      }
}
```

6.2.3 Cliente

Se ha realizado una implementación de un cliente en C (anexo A), que permite enviar peticiones CORECONF desde línea de comandos y muestra las respuestas en la terminal. El cliente recibe 3 argumentos:

- opcion: Indica el tipo de petición a ejecutar. Se han implementado FETCH, iPATCH o POST.
- datos: indica los recursos a los que acceder con FETCH o el recurso y su nuevo valor al usar iPATCH.
- ip/ruta: identifica el recurso y máquina al que queremos acceder o modificar.

```
dit@dit-virtual-machine:~/codigo/comi$ ./coreconf-client fetch 1753 1741 192.168.100.105
Map[

1753
Los-Palacios-y-Villafranca-Sevilla-Andalucia-España

1741
Antonio
]
```

Figura 14. Ejemplo de petición con resultado OK en cliente CORECONF

```
dit@dit-virtual-machine:~/codigo/comi$ ./coreconf-client fetch 1710 192.168.100.105
4.04
```

Figura 15. Ejemplo de petición con resultado de error en cliente CORECONF

Para realizar la implementación, se ha hecho uso de las siguientes librerías:

-libcoap: librería escrita en C que permite enviar y recibir mensajes CoAP.

-cn-cbor: librería escrita en C para codificar y decodificar mensajes en CBOR.

Se ha desarrollado un decodificador de respuestas CBOR que desglosa en la línea de comandos los elementos recibidos en las respuestas. En el caso de recibir un código de error, también lo muestra por pantalla. Para comprobar la correcta implementación, se han realizado peticiones FETCH, iPATCH y POST y se han capturado los paquetes intercambiados con Wireshark.

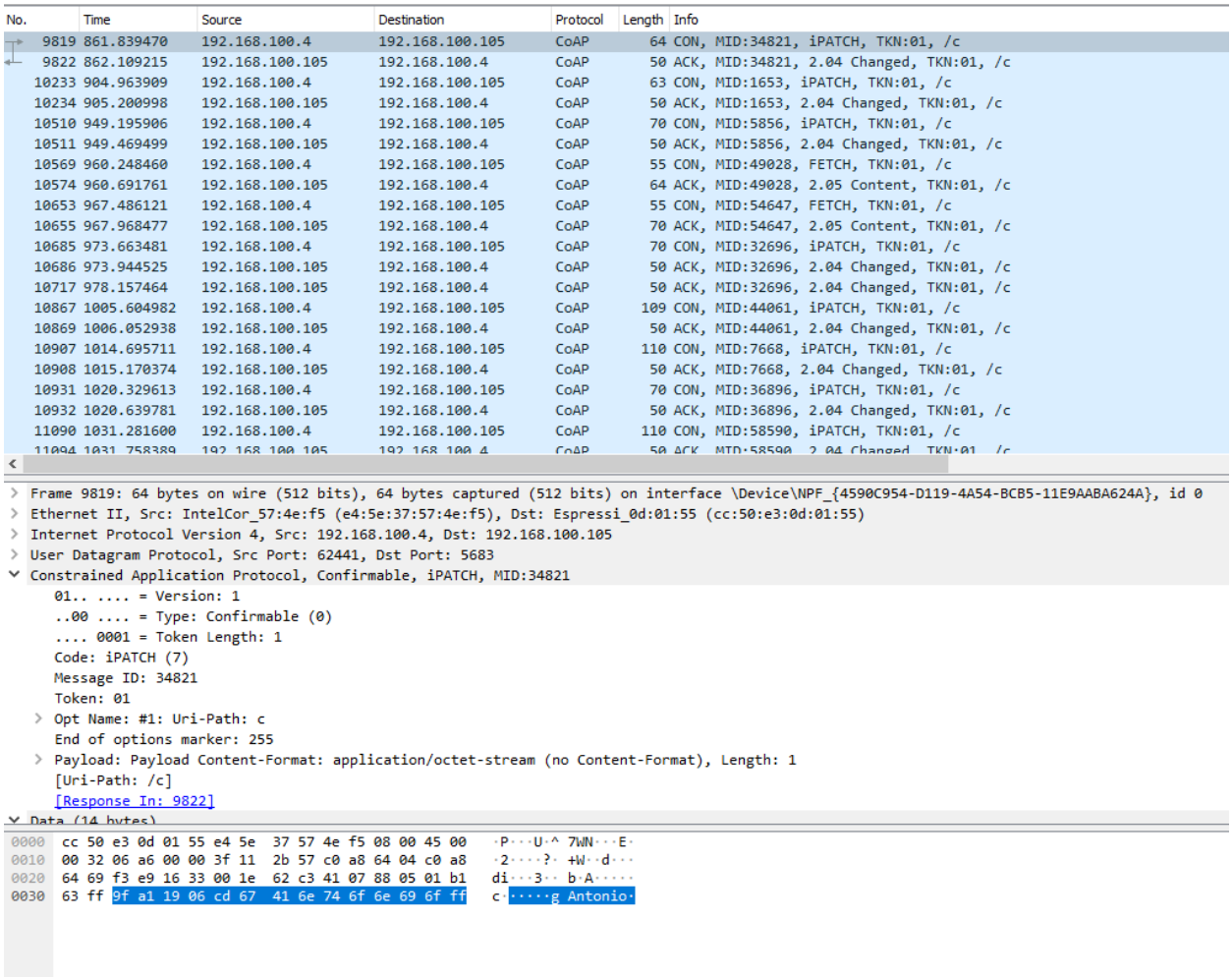

Figura 16. Ejemplo de petición iPATCH

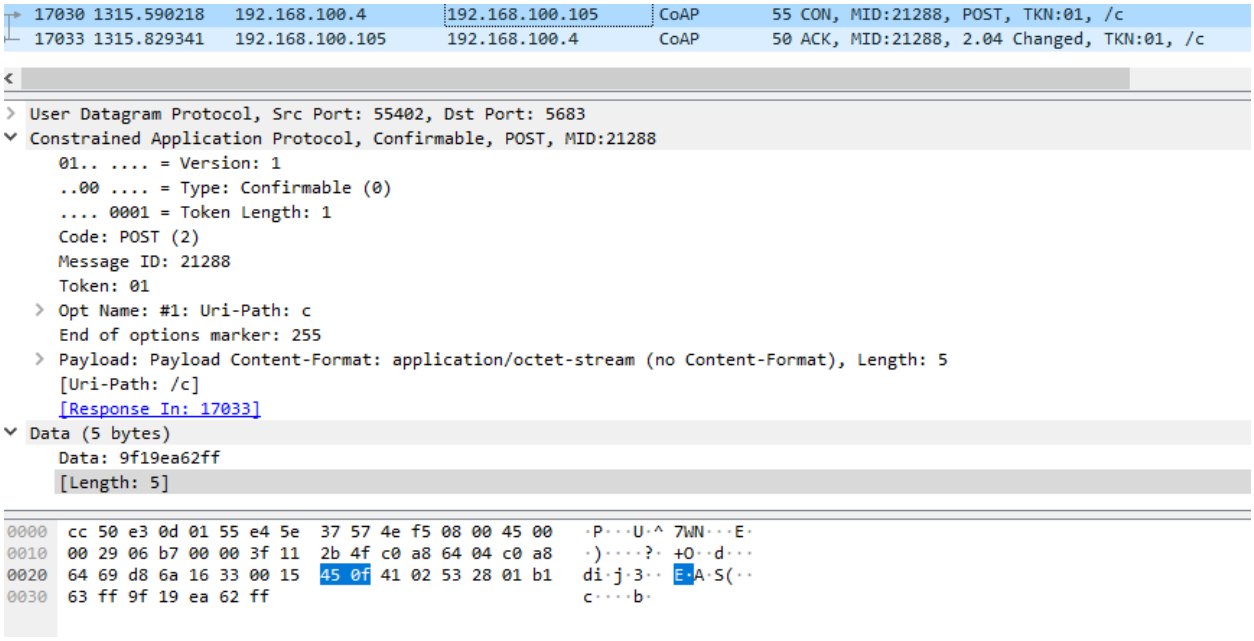

Figura 17. Ejemplo de petición POST

6.3 LwM2M

Para realizar la comparación con LwM2M se ha optado por utilizar implementaciones ya existentes.

6.3.1 Escenario

Para simplificar el escenario y la implementación del cliente, se ha optado por eliminar el servidor de bootstrap y utilizar solo cliente y servidor.

Servidor: PC ejecutando el servidor por defecto de Leshan.

Cliente: Node MCUv3 con distintos objetos de gestión implementados (según la prueba).

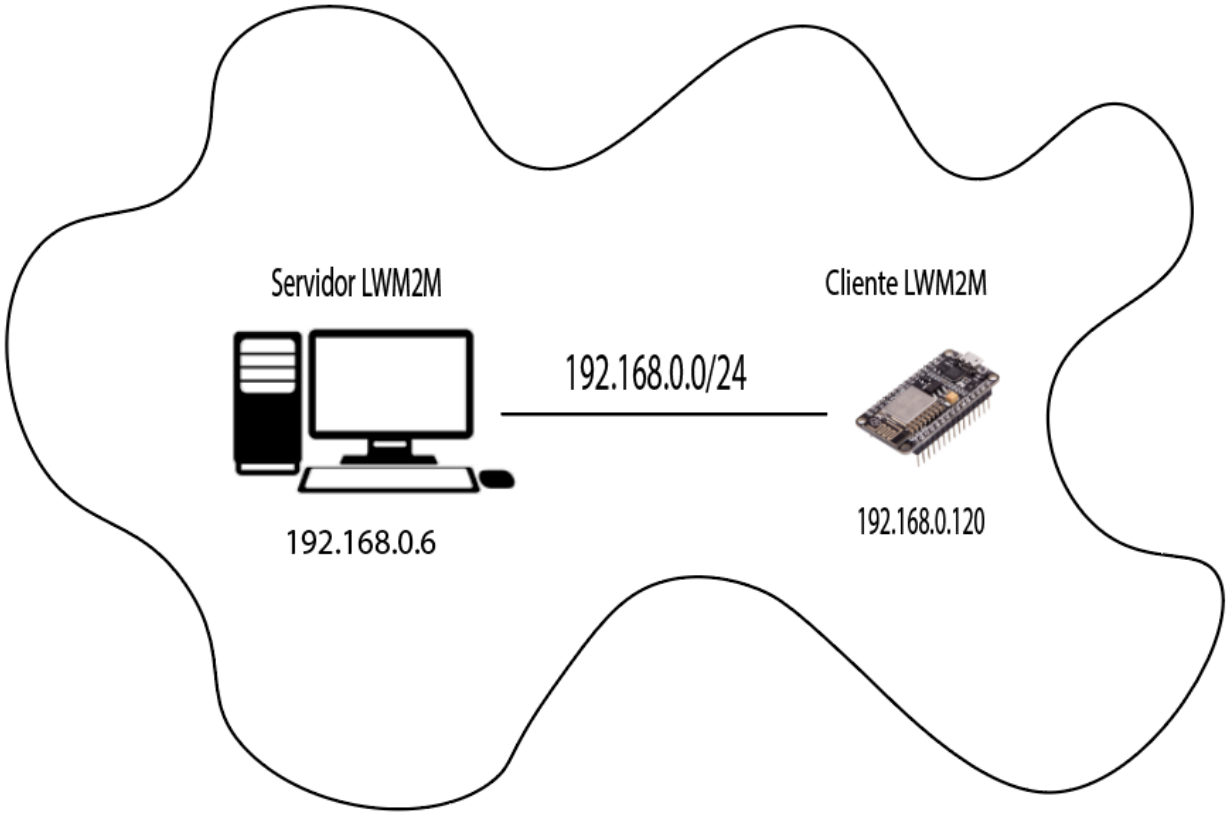

Figura 18. Escenario pruebas LwM2M

6.3.2 Servidor

Como servidor se utilizará el servidor que Leshan trae como ejemplo, desarrollado en java y que cuenta con una interfaz web para acceder a los recursos.

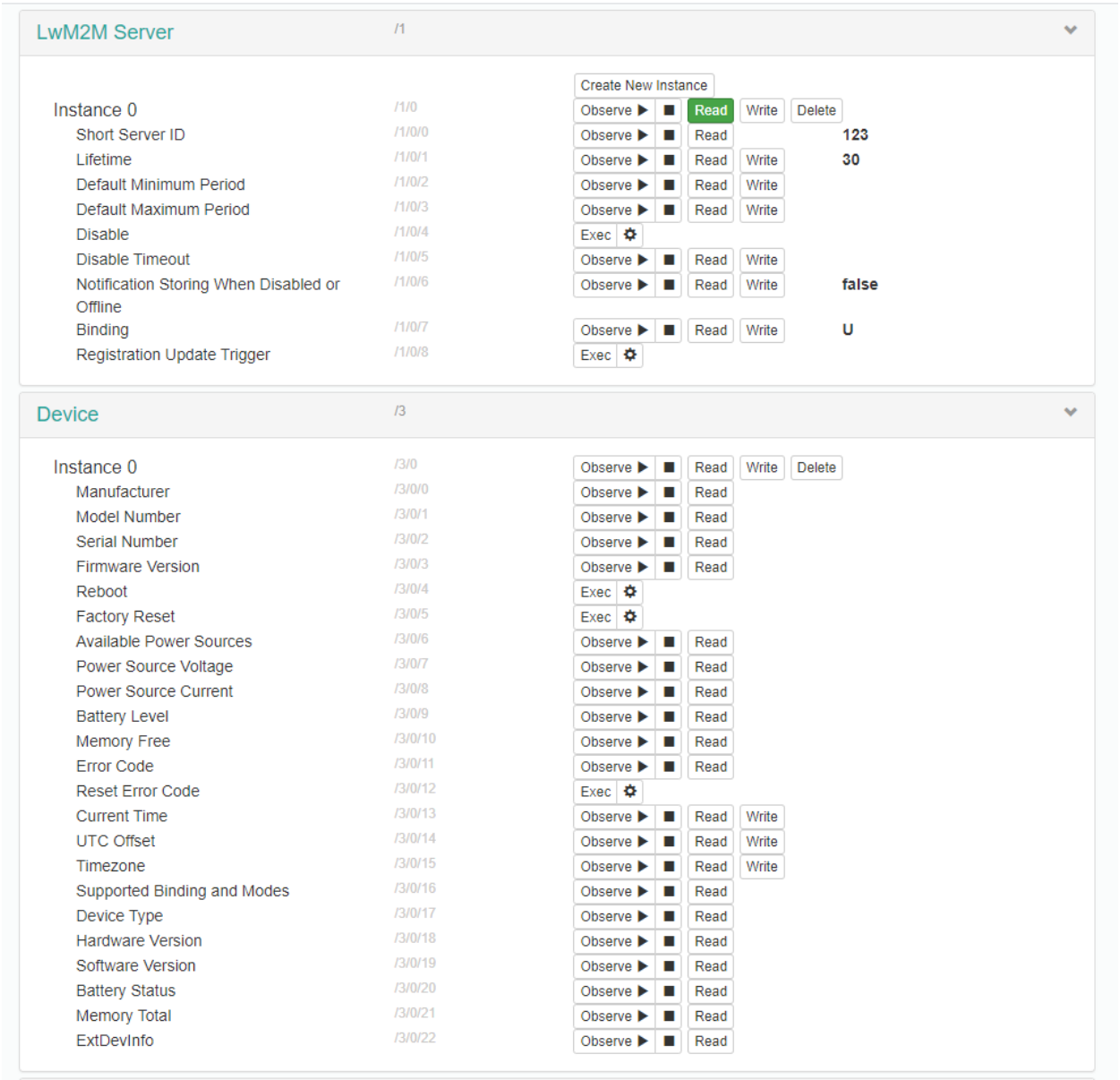

Figura 19. Interfaz gráfica servidor LwM2M Leshan

Accediendo a través del navegador, se pueden realizar las peticiones a todos los clientes registrados en el servidor LwM2M.

6.3.3 Cliente

En el lado cliente se modificará una implementación de ejemplo de Eclipse Wakaama, que consiste en una implementación de LwM2M en C diseñada para ser usada en sistemas POSIX. Se han realizado modificaciones para adaptar el cliente al nodo MCUnode v3.

Eclipse wakaaama registra los recursos obligatorios en LwM2M. Además, se han definido los siguientes objetos:

ID	Nombre	Operaciones	Instancias	Obligatorio	Tipo	Rango o enumeración	Unidades	Descripción

0	Estado	R – Lectura W – Escritura	Único	Obligatorio	Boolean,			Estado del led

-Led: objeto que representa el estado del led del MCUnode. Este objeto solo contiene un recurso.

Tabla 5. Definición del objeto LED

En una segunda implemetación, se han implementado 100 objetos con 2 cadenas, una de lectura y otra de escritura/lectura para realizar la comparación de uso de memoria respecto al protocolo CORECONF. Los 100 objetos son iguales cambiando el ID:

ID	Nombre	Operaciones	Instancias	Obligatorio	Tipo	Rango o enumeración	Unidades	Descripción
1001	Test1RO	R – Lectura	Único	Obligatorio	String			Test solo lectura
1001	Test1RW	R – Lectura W - Escritura	Único	Obligatorio	String			Test lectura y escritura

Tabla 6. Definición de los objetos genéricos de lectura y lectura/escritura

6.3.4 Pruebas

Para comprobar la correcta implementación, se han realizado pruebas de GET, PUT y POST y se han capturado los paquetes intercambiados entre cliente y servidor utilizando Wireshark.

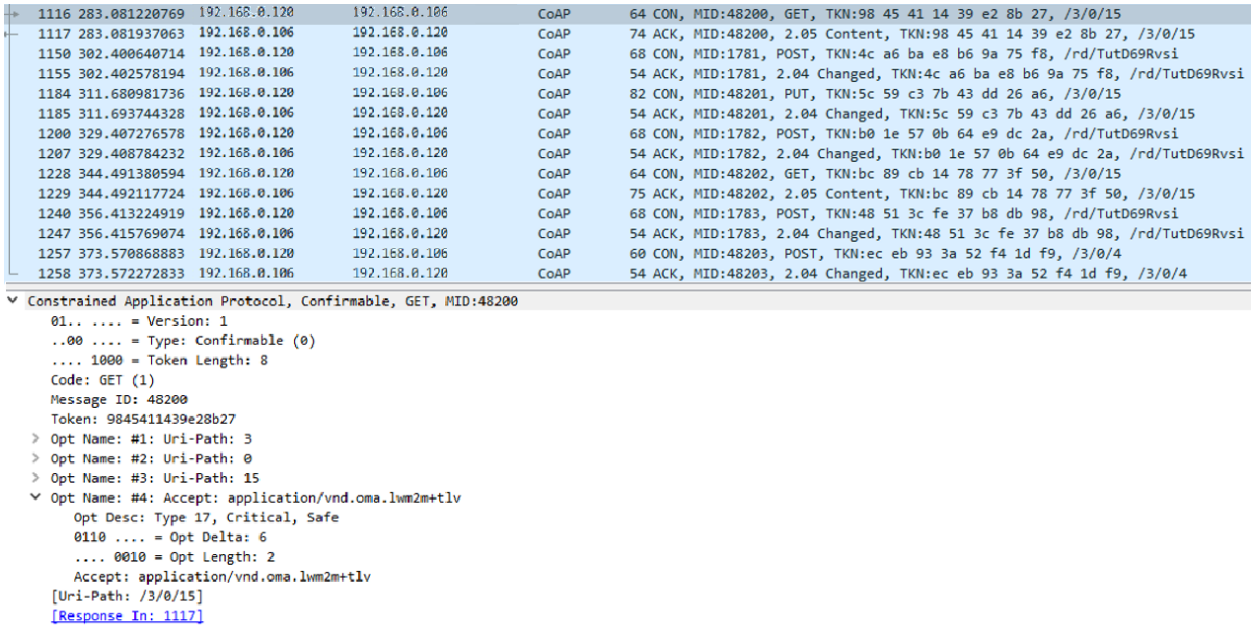

Figura 20. Ejemplo de petición GET

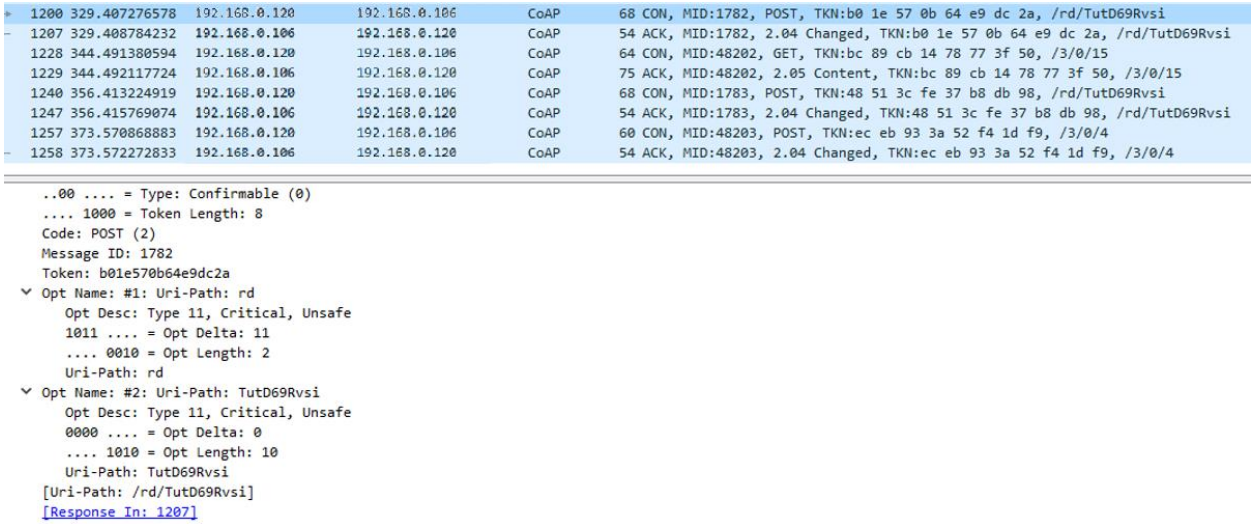

Figura 21. Ejemplo de petición POST

7 COMPARATIVA Y CONCLUSIONES

Now this is not the end, it is not even the beginning of the end. But it is, perhaps, the end of the beginning

-Winston Churchill-

En este capitulo se va a realizar la comparativa entre LwM2M y CORECONF. Esta comparativa se realizará a partir de las pruebas realizadas a ambas implementaciones, teniendo en cuenta tres aspectos: tamaño de los paquetes para el envío de una misma información, uso de memoria de programa en las implementaciones y uso de memoria para datos. Teniendo en cuenta la base teórica, también se realizará una comparativa de las opciones, ventajas y desventajas que ofrecen ambos protocolos en su implementación. Por últimos se numerarán una serie de conclusiones obtenidas de estas comparativa.

7.1 Tamaño de los paquetes

Para realizar la comparativa entre los tamaños de paquetes en LwM2M y CORECONF, se han utilizado un total de 7 peticiones:

- 3 Peticiones de modificación al objeto de localización (tipo String) con los literales: “Madrid”, “Europa/Madrid” y “Los Palacios y Villafranca/Sevilla/Andalucia/España”. Las peticiones son iPATCH para CORECONF y PUT para LwM2M
- 3 Peticiones de obtención de datos al objeto de localización (tipo String) que contenía los literales anteriores. En CORECONF las peticiones son FETCH y en LwM2M GET
- 1 Petición Post para ejecutar una llamada a procedimiento remoto (reboot).

LwM2M no concreta el tipo de codificación que debe utilizar el payload de COAP, por lo que para las pruebas se han decidido usar dos de los más frecuentes: JSON y TLV. Por su parte, el payload de CORECONF será CBOR.

Paquete	LwM2M con codificación JSON (Bytes)	LwM2M con codificación TLV (Bytes)	CORECONF (Bytes)
Petición para modificar el objeto Location a valor “Madrid”	114	75	66
Petición para modificar el objeto Location a valor “Europa/Madrid”	121	83	77
Petición para modificar el objeto Location a valor “Los Palacios y Villafranca/Sevilla/Andalucia/España”	159	122	117
Respuesta a la modificación	54	54	54

Tabla 7. Comparativa del tamaño de los paquetes en peticiones de modificación

En las peticiones de modificación podemos observar como LwM2M necesita una trama un 73% más grande con respecto a CORECONF cuando utiliza la codificación JSON para peticiones con payload corto y va reduciendo la diferencia hasta un 36% en la prueba de mayor payload. La diferencia es mucho menor al utilizar TLV, necesitando un 14% más en peticiones cortas y descendiendo hasta un 4% en la petición más larga.

La diferencia respecto a LwM2M con codificación JSON se debe al uso de la codificación CBOR por parte de CORECONF.

Respecto a LwM2M con codificación TLV, la principal diferencia se encuentra en los bytes dedicados a identificar el recurso. Al utilizar el método PUT, LwM2M identifica el recurso como parte de su URI, y esos bytes por tanto no van codificados con TLV. En cambio CORECONF al utilizar iPATCH identifica el recurso dentro de su payload y aprovecha también la codificación CBOR.

Por el lado de las respuestas, al aportar solo el status CHANGED, hacen que sean exactamente iguales y no haya diferencias.

Paquete	LwM2M con codificación JSON (Bytes)	LwM2M con codificación TLV (Bytes)	CORECONF (Bytes)
Petición para obtener el objeto Location	66	66	62
Respuesta “Madrid”	107	66	70
Respuesta “Europa/Madrid”	114	74	77
Respuesta “Los Palacios y Villafranca/Sevilla/Andalucia/España”	153	114	117

Tabla 8. Comparativa del tamaño de los paquetes en peticiones de obtención de datos

En las peticiones para obtener el valor del objeto se observa una diferencia de 4 bytes entre CORECONF y cualquier opción de LwM2M (un 6,4%). La diferencia viene de nuevo debido a la forma de identificar a los objetos. El uso del método FETCH por parte de CORECONF hace que el SID utilizado para identificar el objeto vaya codificado en CBOR, frente a las peticiones de LwM2M que utilizan GET y el recurso por tanto va en la URI sin codificar.

Analizando las respuestas, en este caso todas tienen payload, por lo que las diferencias existentes se deben tanto por la identificación del objeto como por las codificaciones.

De nuevo utilizar LwM2M con JSON requiere de más bytes que si se utiliza CORECONF. En objetos cuyo valor sean cadenas cortas, la diferencia está entorno al 53% y va disminuyendo hasta el 30% en la prueba de mayor longitud.

Al codificar con TLV, las respuestas son de menor longitud que en CORECONF. Las codificaciones en TLV y CBOR son similares y la diferencia viene principalmente del formato que utilizan los protocolos en sus respuestas. En LwM2M la respuesta al acceder a un recurso está formada únicamente por el valor de ese recurso mientras que en CORECONF es un mapa SID:valor, lo cual añade los bytes extra de diferencia. Estos bytes extras son más notorios cuando la cadena es más corta

Paquete	LwM2M con	LwM2M con	CORECONF

	codificación JSON (Bytes)	codificación TLV (Bytes)	(Bytes)
Petición Post rpc reboot	64	64	62
Respuesta Post rpc reboot	54	54	54

Tabla 9. Comparativa del tamaño de los paquetes en peticiones POST

En la llamada a procedimiento remoto las diferencias en bytes se deben a la identificación de los objetos. En las peticiones LwM2M hace uso de 2 bytes más (un 3% más) mientras que en las respuestas necesitan los mismos bytes al no tener payload.

En general, se observa como CORECONF cumple con uno de sus principales objetivos, reducir el tamaño de las tramas necesarias para la gestión con respecto a LwM2M. Esto lo consigue utilizando CBOR para su codificación y evitando utilizar la identificación de los recursos en la URI de la petición, donde irían sin codificar.

7.2 Memoria de programa (Flash) y memoria RAM

Para realizar la comparativa del uso de memoria se utilizarán los datos obtenidos al realizar la compilación en el IDE PlatformIO. Se han realizado 2 implementaciones para cada uno de los protocolos, una con los recursos mínimos, simulando un dispositivo del que solo nos interesa controlar un objeto y otro con 100 objetos conteniendo cada uno dos cadenas, una de solo lectura y otra de lectura y escritura.

Sketch	Memoria Flash (bytes)	Memoria RAM (bytes)
Sketch vacío	255680	26772
Implementación CORECONF básica	295033	31260
Implementación CORECONF con 100 objetos	324785	42660
Implementación LwM2M básica	316965	34488
Implementación LwM2M con 100 objetos	341493	42147

Tabla 10. Comparativa memoria Flash y RAM

En primer lugar, vemos cómo una implementación básica de CORECONF necesita 39353 bytes de memoria flash por encima de un sketch vacío, mientras que la implementación básica de LwM2M necesita 61285 bytes extra. Este resultado era esperable debido a la existencia de objetos de implementación obligatoria en LwM2M que no existen en CORECONF, además de la posibilidad de modificar el payload entre TLV y JSON, lo que requiere librerías adicionales. Todo esto provoca que en la implementación realizada se necesita un 55,7% de memoria más que en CORECONF con respecto al sketch vacío.

Al implementar 100 objetos, las diferencias disminuyen, necesitándose solo ún 37,5% más, e incluso la diferencia en bytes es menos, por lo que la implementación de objetos LwM2M es más eficiente que la realizada para CORECONF.

Por otro lado, si comparamos la memoria RAM utilizada, obtenemos resultados análogos. Para el caso de la implementación básica, LwM2M necesita más memoria RAM, debido a que presenta objetos obligatorios. Sin

embargo, en la implementación de LwM2M la memoria reservada para los objetos de tipo string se reservan de forma dinámica lo que reduce la RAM utilizada respecto a la implementación de CORECONF, en la que se han fijado el tamaño de los strings.

7.3 Conclusiones

A tenor de lo expuesto tanto en la base teórica como en los resultados obtenidos en las pruebas de rendimiento realizadas, se listan una serie de conclusiones a modo de resumen en la comparativa entre LwM2M y CORECONF:

- CORECONF necesita menos recursos para ser implementado, especialmente en entornos en los que hay que controlar pocos objetos.
- Debido a la utilización de SIDs, la utilización de CBOR como codificación y el uso de los métodos iPATCH y FETCH frente a GET/PUT/POST/DELETE, CORECONF es más eficiente en el envío de información.
- LwM2M define objetos de seguridad de obligada implementación, mientras que CORECONF no lo hace.
- CORECONF utiliza UDP como capa de transporte y DTLS para securización, mientras que LwM2M permita utilizar UDP con DTLS, TCP con TLS o SMS.
- LwM2M controla el registro y desregistro de dispositivos con el servidor de arranque, mientras que CORECONF no proporciona ninguna forma de registro.
- LwM2M cuenta con objetos de gestión definidos por la OMA y por empresas que han solicitado el alta de objetos a la OMNA. CORECONF cuenta con SIDs definidos por el IETF y registrados por IANA y aunque sus objetos provienen de definiciones YANG, la gran mayoría de módulos YANG existentes a día de hoy no tienen SIDs asociados por no utilizarse en entornos restringidos.

REFERENCIAS

[1] Vailshery, L. S. (2023, Julio 27). *statista*. Retrieved from https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/

[2] M. Veillete, Ed.; Trillilant Networks Inc.; P. van der Stok, Ed. (2023). *draft-ietf-core-comi-16 - CoAP Management Interface (CORECONF).* Retrieved from https://datatracker.ietf.org/doc/draft-ietf-core-comi/16/

[3] Sinche, S., Raposo, D., Armando, N., Rodrigues, A., Boavida, F., Pereira, V., & Silva, J. S. (2019). A survey of IoT management protocols and frameworks. *IEEE Communications Surveys & Tutorials*, *22*(2), 1168-1190.

[4] Parmigiani, A., & Dettmar, U. (2021, November). Comparison and Evaluation of LwM2M and MQTT in Low-Power Wide-Area Networks. In *2021 IEEE International Conference on Internet of Things and Intelligence Systems (IoTaIS)* (pp. 8-14). IEEE.

[5] Saif, D., & Matrawy, A. (2023). Mass Configuration of Heterogeneous IIoT Nodes: A Proposal and Experimental Evaluation. *Authorea Preprints*.

[6] Bhat, M. G., Bhattacharjee, S., Gündoğan, C., Alexandris, K., & Gogolev, A. (2023). CORECONF, NETCONF, and RESTCONF: Benchmarking Network Orchestration in Constrained IIoT Devices. *IEEE Internet of Things Journal*.

[7] Fielding, R. T. (2000). *REST: Architectural Styles and the Design of Network-based Software rchitectures.* Doctoral dissertation Unpublished doctoral dissertation , University of California, Irvine .

[8] Fielding, R. T., & Taylor, R. N. (2002). Principled design of the modern web architecture. *ACM Transactions on Internet Technology (TOIT)*, *2*(2), 115-150.

[9] Pautasso, C., & Wilde, E. (2010, April). RESTful web services: principles, patterns, emerging technologies. In *Proceedings of the 19th international conference on World wide web* (pp. 1359-1360).

[10] Webber, J., Parastatidis, S., & Robinson, I. (2010). *REST in practice: Hypermedia and systems architecture*. " O'Reilly Media, Inc.".

[11] Bjorklun, M; Tail-f Systems. (2016). RFC 7950 - The YANG 1.1 Data Modeling Language. Obtenido de https://tools.ietf.org/html/rfc7950

[12] Xu, H., & Xiao, D. (2008, November). Data modeling for NETCONF-based network management: XML schema or YANG. In *2008 11th IEEE International Conference on Communication Technology* (pp. 561-564). IEEE.

[13] Cui, H., Zhang, B., Li, G., Gao, X., & Li, Y. (2009, February). Contrast analysis of netconf modeling languages: Xml schema, relax ng and yang. In *2009 International Conference on Communication Software and Networks* (pp. 322-326). IEEE.

[14] Schönwälder, J. (2012, July). Network configuration management with NETCONF and YANG. In *84th IETF Meeting* (Vol. 29).

[15] Shelby, Z., Hartke, K. y Bormann, C. (2014). RFC 7252 - The Constrained Application Protocol (CoAP). Obtenido de https://tools.ietf.org/html/rfc7252

[16] Shelby, Z. (2014). Coap: The web of things protocol. *ARM IoT Tutorial*, *3*.

[17] Veillete, M.; Trillilant Networks Inc.; Pelov, A. (2018). draft-ietf-core-yang-cbor-07 - CBOR Encoding of Data Modeled with YANG. Obtenido de https://tools.ietf.org/html/draft-ietf-core-yang-cbor-07

[18] Josefsson, S. (2006). RFC 4648 – The Base16, Base32, and Base64 Data Encodings. Obtenido de https://datatracker.ietf.org/doc/html/rfc4648

[19] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., & Wilton, R. (2018). RFC 8342: Network Management Datastore Architecture (NMDA).

[20] Selander, G., Mattsson, J., Palombini, F., & Seitz, L. (2019). RFC 8613: Object Security for Constrained

RESTful Environments (OSCORE).

[21] Open Mobile Alliance. (2019). *LwM2M v1.1.* Obtenido de http://openmobilealliance.org/release/LightweightM2M/Lightweight_Machine_to_Machine-v1_1-OMASpecworks.pdf

[22] Rao, S., Chendanda, D., Deshpande, C., & Lakkundi, V. (8 de 2015). Implementing LWM2M in constrained IoT devices. *2015 IEEE Conference on Wireless Sensors (ICWiSe)*.

ANEXO A: CLIENTE CORECONF

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "src/cbor.h"

#define IPATCH 7
#define FETCH 5
#define POST 2

char* devuelvePeticion(int tipo, int argc, char* argv[]){
    char name[120];
    char* peticion = (char*)malloc(100*sizeof(char));
    CborEncoder encoder;
    CborEncoder array_encoder;
    uint8_t buf[200];
    size_t buflen;
    int len_peticion;
    int len_sid;
    char sid[6];
    const char *nombreArchivo = "peticion.bin";


    //Añadir el resto del comando
    sprintf(peticion, " ");
    cbor_encoder_init(&encoder, buf, 100, 0);
```

```
    cbor_encoder_create_array(&encoder, &array_encoder,
CborIndefiniteLength);

    int isid = 1;

    for(isid=2;isid<argc-1;isid++){

        if(strchr(argv[isid],'/')!=NULL){

            strcpy(name,strrchr(argv[isid],'/')+1);

            if(name!=NULL && strcmp(name,"")!=0){

                printf("%s\n",name);

                len_peticion=strlen(argv[isid]);

                int len_name=strlen(name);

                len_sid=len_peticion-len_name-1;

                printf("len_peticion %d y len_sid %d y len_name
%d\n",len_peticion,len_sid, len_name);

                strncpy(sid, argv[isid], len_sid);

                printf("%s\n",sid);

                //Tenemos que codificar un mapa

                CborEncoder parameterMap;

                cbor_encoder_create_map(&array_encoder, &parameterMap,
1);

                cbor_encode_uint(&parameterMap, atoi(sid));

                cbor_encode_text_string(&parameterMap, name,
strlen(name));

                cbor_encoder_close_container(&array_encoder,
&parameterMap);

                strcpy(name, "");

            }

        }else{

            if(tipo==IPATCH){

                CborEncoder parameterMap;

                cbor_encoder_create_map(&array_encoder, &parameterMap,
1);
```

```
                cbor_encode_uint(&parameterMap, atoi(sid));
                cbor_encode_null(&parameterMap);
                cbor_encoder_close_container(&array_encoder,
&parameterMap);
            }else if(tipo == FETCH || tipo == POST){
                cbor_encode_uint(&array_encoder, atoi(argv[isid]));
            }

        }

    }

    cbor_encoder_close_container(&encoder, &array_encoder);
    buflen=cbor_encoder_get_buffer_size(&encoder,buf);
    //Guardamos el contenido en un archivo
    FILE *archivo = fopen(nombreArchivo, "wb");

    if (archivo == NULL) {
        fprintf(stderr, "No se pudo abrir el archivo para
escritura.\n");
        return "";  // Terminar el programa
    }

    // Escribir el contenido de buf en el archivo
    size_t elementosEscritos = fwrite(buf, sizeof(uint8_t), buflen,
archivo);
    // Cerrar el archivo
    fclose(archivo);

    printf("Datos guardados exitosamente en %s.\n", nombreArchivo);
```

```
    strcat(peticion, nombreArchivo);
    strcat(peticion, " -T cafe2024 coap://");
    //strcat(peticion, " coap://");
    strcat(peticion, argv[argc-1]);
    strcat(peticion, "/c");
    printf("saliendo.\n");
    return peticion;
}

static void indent(int nestingLevel)
{
    while (nestingLevel--)
        puts("  ");
}

static void dumpbytes(const uint8_t *buf, size_t len)
{
    while (len--)
        printf("%02X ", *buf++);
}
static CborError dumprecursive(CborValue *it, int nestingLevel)
{
    while (!cbor_value_at_end(it)) {
        CborError err;
        CborType type = cbor_value_get_type(it);

        indent(nestingLevel);
        switch (type) {
```

```
case CborArrayType:
case CborMapType: {
    // recursive type
    CborValue recursed;
    assert(cbor_value_is_container(it));
    puts(type == CborArrayType ? "Array[" : "Map[");
    err = cbor_value_enter_container(it, &recursed);
    if (err)
        return err;       // parse error
    err = dumprecursive(&recursed, nestingLevel + 1);
    if (err)
        return err;       // parse error
    err = cbor_value_leave_container(it, &recursed);
    if (err)
        return err;       // parse error
    indent(nestingLevel);
    puts("]");
    continue;
}

case CborIntegerType: {
    int64_t val;
    cbor_value_get_int64(it, &val);
    printf("%lld", (long long)val);
    break;
}

case CborByteStringType: {
    uint8_t *buf;
```

```
        size_t n;
        err = cbor_value_dup_byte_string(it, &buf, &n, it);
        if (err)
            return err;     // parse error
        dumpbytes(buf, n);
        puts("");
        free(buf);
        continue;
    }

    case CborTextStringType: {
        char *buf;
        size_t n;
        err = cbor_value_dup_text_string(it, &buf, &n, it);
        if (err)
            return err;     // parse error
        puts(buf);
        free(buf);
        continue;
    }

    case CborTagType: {
        CborTag tag;
        cbor_value_get_tag(it, &tag);
        printf("Tag(%lld)", (long long)tag);
        break;
    }

    case CborSimpleType: {
```

```
        uint8_t type;
        cbor_value_get_simple_type(it, &type);
        printf("simple(%u)", type);
        break;
    }

    case CborNullType:
        puts("null");
        break;

    case CborUndefinedType:
        puts("undefined");
        break;

    case CborBooleanType: {
        bool val;
        cbor_value_get_boolean(it, &val);
        puts(val ? "true" : "false");
        break;
    }

    case CborDoubleType: {
        double val;
        if (false) {
            float f;
    case CborFloatType:
            cbor_value_get_float(it, &f);
            val = f;
        } else {
```

```
                cbor_value_get_double(it, &val);
            }
            printf("%g", val);
            break;
        }
        case CborHalfFloatType: {
            uint16_t val;
            cbor_value_get_half_float(it, &val);
            printf("__f16(%04x)", val);
            break;
        }

        case CborInvalidType:
            assert(false);
            break;
        }

        err = cbor_value_advance_fixed(it);
        if (err)
            return err;
    }
    return CborNoError;
}

int main(int argc, char** argv){
char peticion [200];
char option[5][10]={"get","put","post","fetch","ipatch"};
char payload[100];
uint8_t inicio[30];
```

```
int i;
int numBytes;
char base64[4];
char ip[18];
char ruta[20];
char sid[6];
int len_peticion;
int len_sid;
char* objeto;
char* parametro;
FILE *pipe;
char buffer[256];



CborParser parser;
CborValue iterador;
     if(argc==1){
          printf("Uso: %s opcion [datos] ip/ruta \n",argv[0]);
     }else{
          if(strcmp(argv[1],option[2])==0){
               strcpy(peticion, "coap-client -m post -f");
            strcat(peticion, devuelvePeticion(POST, argc, argv));
          }else if(strcmp(argv[1],option[3])==0){
            //FETCH
            strcpy(peticion, "coap-client -m fetch -f");
            strcat(peticion, devuelvePeticion(FETCH, argc, argv));

        }else if(strcmp(argv[1],option[4])==0){
```

```
            //iPATCH
            strcpy(peticion, "coap-client -m ipatch -f");
            strcat(peticion, devuelvePeticion(IPATCH, argc, argv));

        }
        //Llamada a coap-client para ejecutar la peticion
        printf("Peticion: %s\n", peticion);
        pipe = popen(peticion, "r");

        if (pipe == NULL) {
        printf("El comando ha fallado\n" );
        exit(1);
        }
        while (fgets(buffer, sizeof(buffer), pipe) != NULL) {
        //Analizar la respuesta
        cbor_parser_init(buffer, 256, 0, &parser, &iterador);
        //Mostrarla por pantalla
        dumprecursive(&iterador, 0);

        }
          /* Cerrar el pipe */
        pclose(pipe);
    }
    exit(0);
}
```