Introducción a CoAP

CoAP es un protocolo de aplicación para dispositivos y redes restringidas que tiene una fuerte semejanza con el protocolo HTTP. Lo que vamos a ver en este documento son los siguientes temas:

- Una introducción general sobre el protocolo CoAP.
- En qué escenarios es conveniente que utilices el protocolo.
- Los modelos de interacción y los tipos de respuestas sincrónicas y asincrónicas.
- Las características más importantes sobre los tipos de mensajes CoAP.
- Los métodos que admite el protocolo.
- La descripción sobre cómo están formados los mensajes.
- Las características que permiten correr CoAP sobre UDP.
- El funcionamiento del caché para poder optimizar el uso de la red.
- Algunos aspectos importantes como el envío de bloques, observación y descubrimiento de recursos, el mapeo de CoAP y HTTP, los códigos de los mensajes y los tipos de contenido admitidos.
- Algunas implementaciones del protocolo para dispositivos restringidos como así también en diferentes lenguajes de programación.

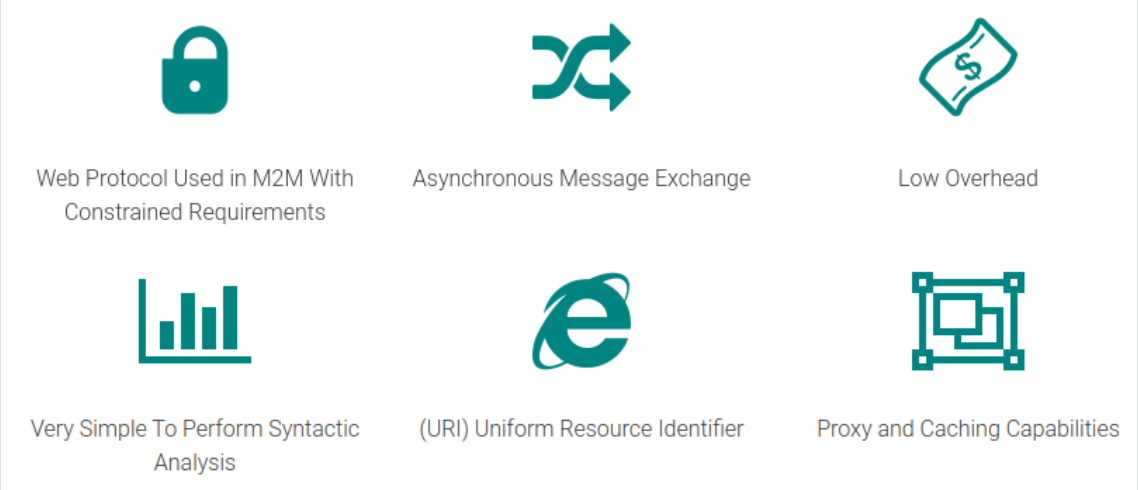
¿Qué es CoAP?

CoAP (Constrained Application Protocol) es un protocolo de software que se encuentra en el nivel de capa de aplicación del modelo OSI y está apuntado a correr en dispositivos simples, permitiendo que puedan comunicarse sobre internet. Fue diseñado especialmente para trasladar el modelo de requests y responses de HTTP a dispositivos y redes con recursos limitados. Las principales características son:

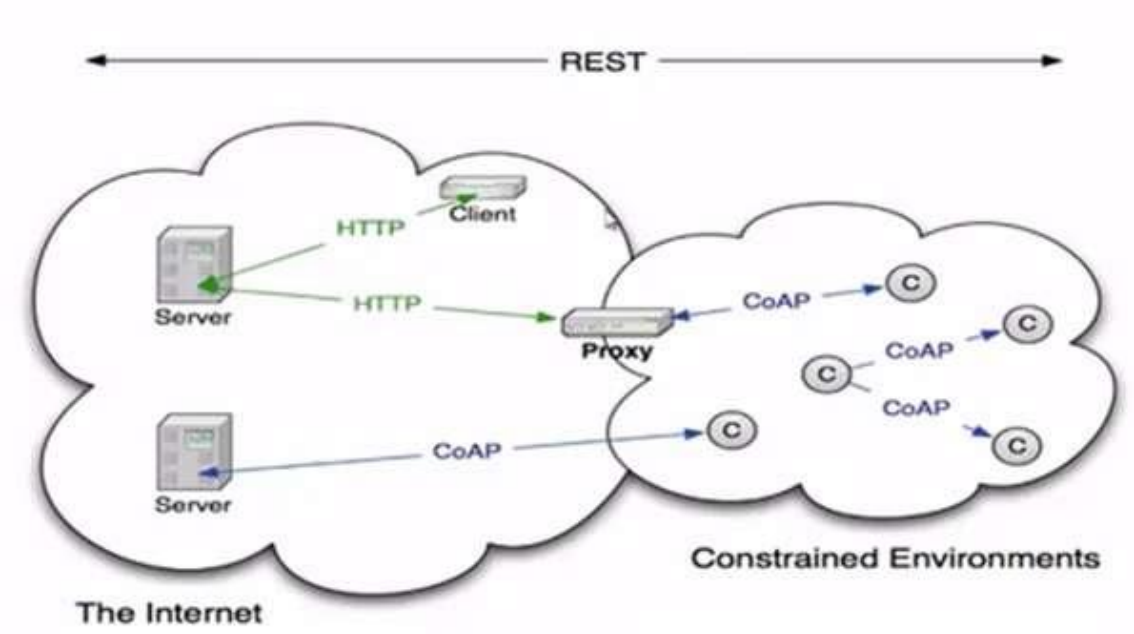
- Es un protocolo web utilizado para correr en dispositivos con recursos limitados.
- Permite el intercambio de mensajes asincrónicos, es decir que un mensaje puede llegar en cualquier momento.
- Posee una baja sobrecarga de datos asociados a cada paquete.
- Es simple de analizar con distintas herramientas, que permiten ver los tipos de mensajes enviados, sus valores y ejecutar análisis de datos.
- Es posible diseñar sus recursos de manera análoga a HTTP, tiene compatibilidad con la definición unificada de recursos (URI) y puede manejar distintos tipos de contenido.

- Al ser utilizado como puente con HTTP comparte capacidades de proxy y caché.

Haciendo foco en IoT, tiene la capacidad de observar si ocurrieron cambios en algún recurso en particular de manera nativa, y también es capaz de detectar si nuevos dispositivos se unieron a la red, pudiendo interactuar de manera automática. Veamos en esta figura algunas de sus características.



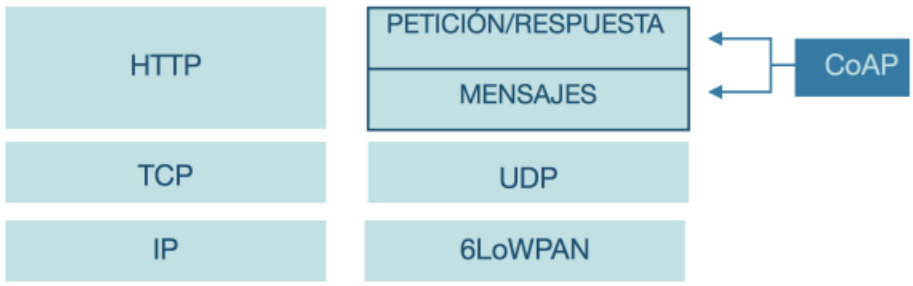
CoAP es un protocolo de aplicaciones restringidas que puede correr en hardware simple - como microcontroladores, sensores de baja potencia y dispositivos que no pueden ejecutar HTTP -, permitiendo que los dispositivos se comuniquen a través de internet.
Debido a que tiene amplias similitudes con HTTP, en muchos casos se suele realizar la comunicación desde un cliente HTTP hacia un servidor CoAP utilizando un agente en el medio como puede ser un proxy - que traduce literalmente request HTTP a CoAP y viceversa - como así también una interfaz HTTP-CoAP que no necesariamente aplica una traducción literal de métodos y códigos de error, pero que permite realizar una comunicación.
En esta figura podés ver un ejemplo de comunicación CoAP donde se muestran varios tipos de comunicaciones. Por ejemplo, es posible conectar directamente un servidor con un cliente a través de CoAP. Para comunicar un cliente HTTP con un server CoAP es necesario que primero el cliente realice un request a un server HTTP, luego este server identifique la petición e intente comunicarse con un proxy HTTP-CoAP que realiza la conversión, para posteriormente realizar un request hacia el server CoAP. Una vez que se obtiene una respuesta CoAP, el mensaje viaja en la dirección inversa hacia el cliente HTTP.



CoAP Architecture

Si bien HTTP y CoAP comparten mucho en su estructura, HTTP se ejecuta sobre la capa de transporte TCP, mientras que CoAP lo hace a través de UDP adaptando el modelo de requests y responses. Así mismo, CoAP incluye otras características como multicast - es decir la capacidad de enviar un mensaje a múltiples destinatarios -, baja sobrecarga de datos en cada paquete y simplicidad en la ejecución, que son muy importantes para muchas aplicaciones IoT.

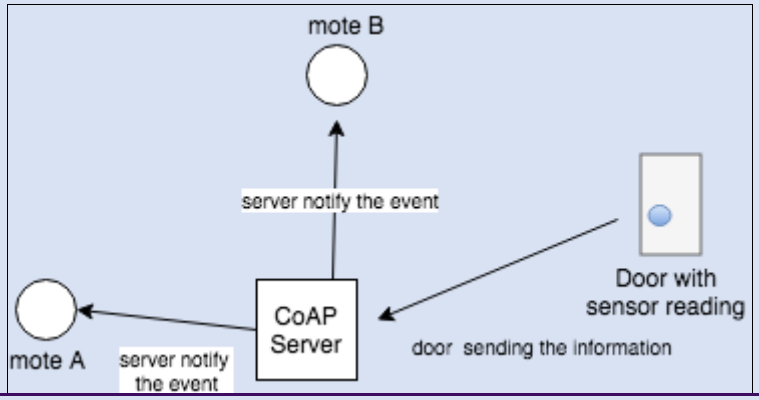
Para realizar el intercambio de mensajes, CoAP está dividido en dos subcapas diferentes. Por un lado podemos encontrar la capa Message, que se ocupa de realizar la interfaz con UDP y de recibir y enviar mensajes asincrónicos. Por otro lado, la capa Request/Response gestiona la interacción de las peticiones y sus respuestas. En esta figura podemos ver una comparación de las capas que utilizan CoAP y HTTP para que entiendas sus diferencias.



Los escenarios donde podrías implementar CoAP van desde comunicación entre dispositivos en una misma red restringida - por ejemplo en redes de baja potencia a través de 6LoWPAN -, entre dispositivos de una red conectados a internet, hasta entre dispositivos en diferentes redes restringidas, ambas unidas por internet.

El tamaño de los paquetes en CoAP es mucho menor que su contraparte en HTTP. Mantiene la misma arquitectura cliente-servidor y soporta únicamente un subset de métodos que son GET, PUT, POST y DELETE, así como también un subset de

posibles códigos de respuesta. En CoAP, además, es posible realizar requests añadiendo la función de observar, que permite al cliente recibir actualizaciones sobre un determinado recurso que haya solicitado previamente. En la figura siguiente podés apreciar un esquema de comunicación utilizando la funcionalidad de observar.



> Para la funcionalidad de observar propuesta por CoAP, podrías encontrar una analogía con la implementación de WebSockets en HTTP o bien la suscripción a un topic usando MQTT.

Además de lo que ya nombramos, cuando se ejecutan requests CoAP es posible negociar el tipo de contenido (Content-Type) que se quiere obtener en una respuesta. Estos pueden ser formatos de texto plano, CSV, XML, así como también formatos de imagen, audio, video y otros mensajes propios de cada aplicación como JSON por ejemplo.

Otra característica interesante es que se pueden solicitar diferentes niveles de calidad de servicio para el envío de los mensajes, que aplica tanto a los requests como a los responses. De esta manera, es posible marcar los mensajes como confirmables, donde se espera que el destinatario avise sobre la recepción de un mensaje, como así también mensajes no confirmables, donde el mensaje se envía sin garantizar que llegue correctamente a destino.

> Nuevamente, para buscar una analogía con esta funcionalidad, podés pensar en la calidad de servicio del protocolo MQTT, donde los mensajes pueden enviarse sin chequear, enviarse al menos una vez, o enviarse exactamente una vez.

¿Cuándo conviene usar CoAP?

Ahora que ya entendés de manera general de qué se trata el protocolo, veamos algunos casos donde te puede convenir utilizarlo.

No se puede implementar HTTP

En muchos proyectos IoT es posible que cuentes con hardware que no puede ejecutar HTTP, ya que la complejidad de implementación no resulta viable por cuestiones de memoria, tamaño del programa o capacidad de procesamiento.

En tales casos, ejecutar CoAP puede ser prácticamente lo mismo que HTTP pero con una demanda de recursos mucho menor. Así mismo, si ya contás con una aplicación HTTP funcionando, realizar la migración no debería resultar costosa, ya que podrías ejecutar el método GET para leer y POST, PUT y DELETE para modificaciones en los recursos.

Consumo de energía

En muchas aplicaciones es necesario que los dispositivos remotos se alimenten de baterías. En estos casos, el consumo de energía se debe mantener al mínimo posible, y ejecutar CoAP puede mejorar el rendimiento en comparación con implementar HTTP sobre TCP/IP.

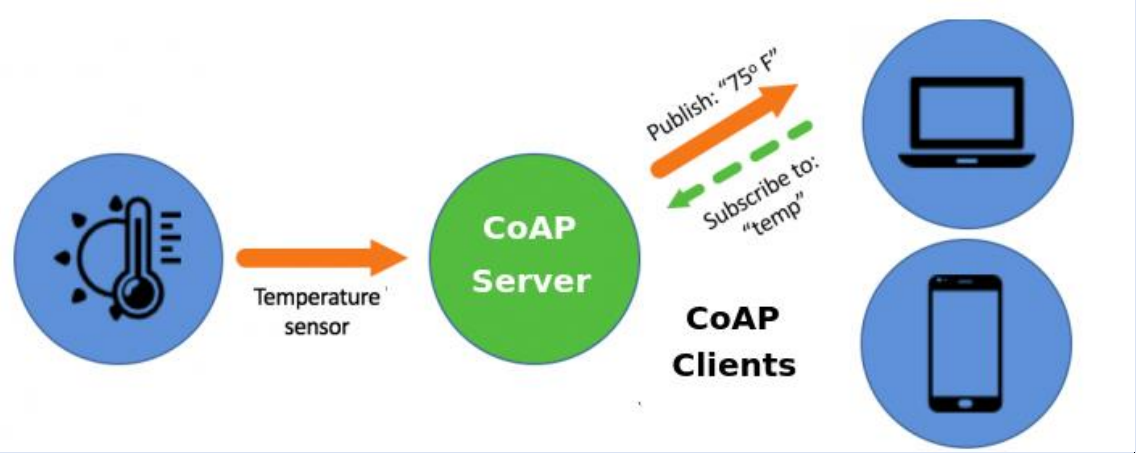
La capa de transporte UDP que usa CoAP ahorra ancho de banda y hace que el protocolo sea más eficiente. Así mismo, si se usa 6LoWPAN como capa de direccionamiento, el consumo podría verse más beneficiado aún, por lo cual, el uso de CoAP en estos casos puede resultar una opción más que viable.

Observación de recursos

En algunas aplicaciones es necesario que puedas saber cuando un recurso en particular ha cambiado de estado. Este recurso puede ser, por ejemplo, el valor de un sensor de puerta o la monitorización de un sensor de temperatura.

Tal como vimos anteriormente, la característica de observar puede ser aplicable con otros protocolos, como por ejemplo utilizar la suscripción a un topic con MQTT o el uso de WebSockets en una comunicación a través de HTTP. Más allá que sea posible con otros métodos - y encadenando con los puntos anteriores de incapacidad para correr HTTP o bien optimizar el consumo de energía sin usar TCP como capa de transporte -, CoAP permite una observación de los recursos a través de un protocolo más simple que HTTP y MQTT, y optimizando de mejor manera el consumo de energía para esos casos.

En la figura a continuación podemos ver un diagrama de suscripción a un recurso en particular. El servidor CoAP realiza la lectura de un sensor de temperatura periódicamente, y si hay clientes que previamente hayan solicitado recibir notificaciones de esta variable, en cuanto haya un cambio de valor, se les enviarán las actualizaciones de manera automática.



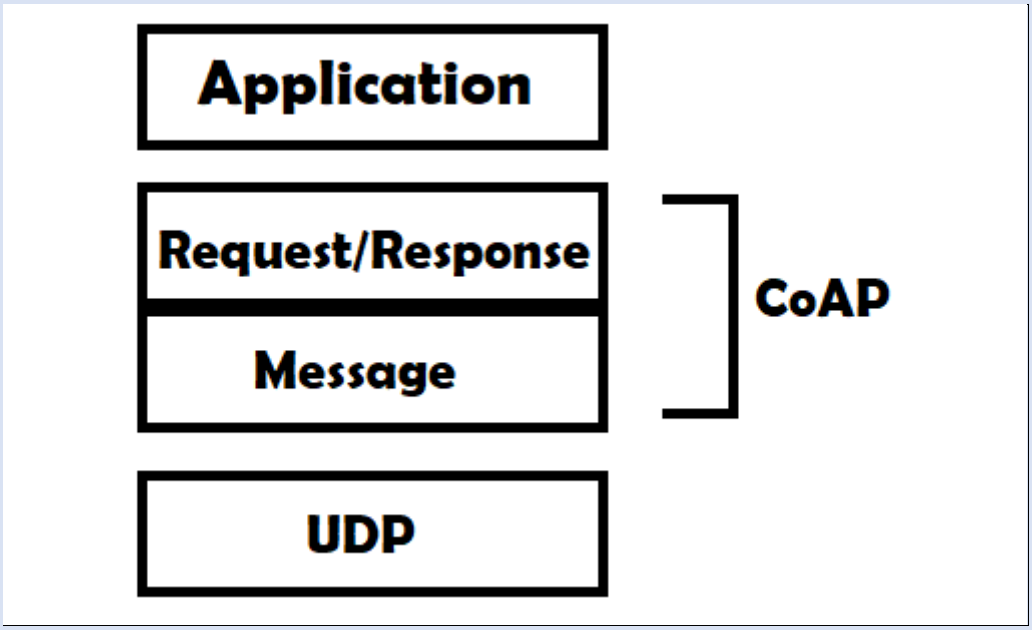
La observación de un recurso también se podría realizar si los clientes realizan requests periódicos al servidor. Esto se conoce como polling. Aunque este método es muy ineficiente para realizar la tarea. Por esa razón, la funcionalidad observe de CoAP resulta muy conveniente.

Modelos de interacción

El modelo de interacción de CoAP es similar al modelo cliente/servidor de HTTP. Sin embargo, en muchas ocasiones en CoAP, un dispositivo puede actuar como servidor y cliente. Un intercambio CoAP es equivalente al de HTTP y es enviado por un cliente para solicitar una acción - usando un método GET, POST, PUT o DELETE - en un recurso - identificado por una URI - correspondiente a un servidor en particular. Al recibir una solicitud o request, el servidor CoAP la procesa y envía una respuesta al cliente especificando un código de respuesta y una representación del contenido si corresponde.

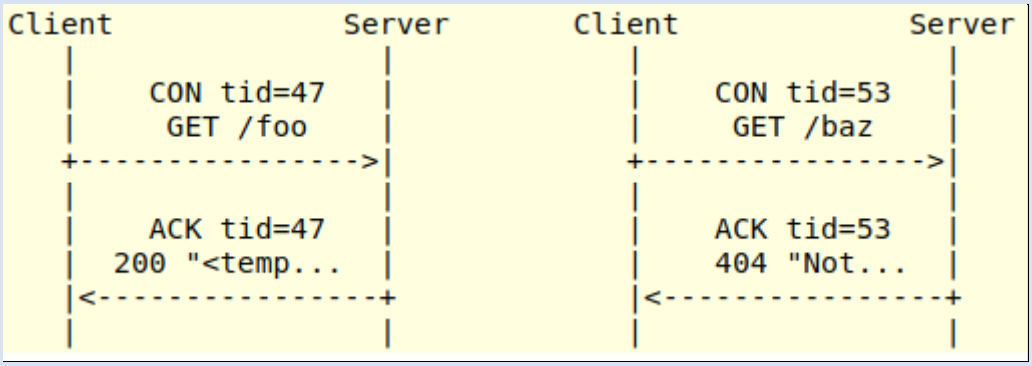
A diferencia de HTTP, CoAP se ocupa de estos intercambios de forma asincrónica a través de UDP. Esto se logra mediante distintos tipos de mensajes identificables con un ID y que tienen una calidad de servicio asociada. Las transacciones solicitud/respuesta son transparentes para el usuario, y es el protocolo CoAP quien se encarga de manejar el asincronismo.

Reforzando los conceptos que vimos anteriormente, CoAP utiliza un enfoque de dos capas para funcionar. Una capa transaccional llamada Message que es utilizada para tratar con UDP y la naturaleza asincrónica de las interacciones, y la capa Request/Response que se encarga de las interacciones utilizando distintos métodos. En la figura siguiente podés ver la la representación de este esquema.



Respuesta sincrónica

La interacción más común entre las capas Request/Response y Message funciona enviando una solicitud en un mensaje CoAP confirmable y esperando un mensaje de acuse de recibo (ACK). En esta imagen podemos ver un ejemplo de dos posibles interacciones para un método GET, donde en una existe una respuesta, y en la otra el recurso no fue encontrado.

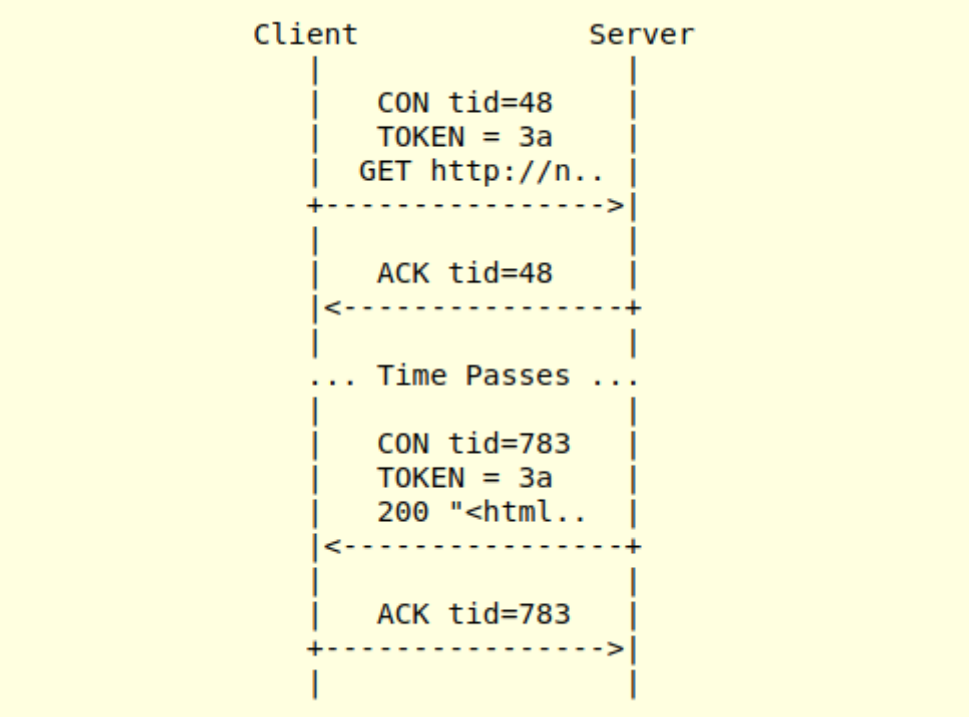


Cuando el servidor le envía al cliente una respuesta de un mensaje confirmable, éste espera que el cliente, al recibir la respuesta, le envíe un mensaje ACK avisando que recibió correctamente el mensaje. Para los casos de ejemplo anteriores, no es necesario que el servidor le responda por separado un ACK cuando recibe la respuesta, ya que el mensaje ACK puede ir incluido en la propia respuesta.
La relación entre el mensaje confirmable (CON) y el mensaje de acuse de recibo (ACK) se indica mediante el ID de transacción. Estos IDs son únicos por cada transacción y cumplen la función de vincular un mensaje confirmable con su correspondiente respuesta.

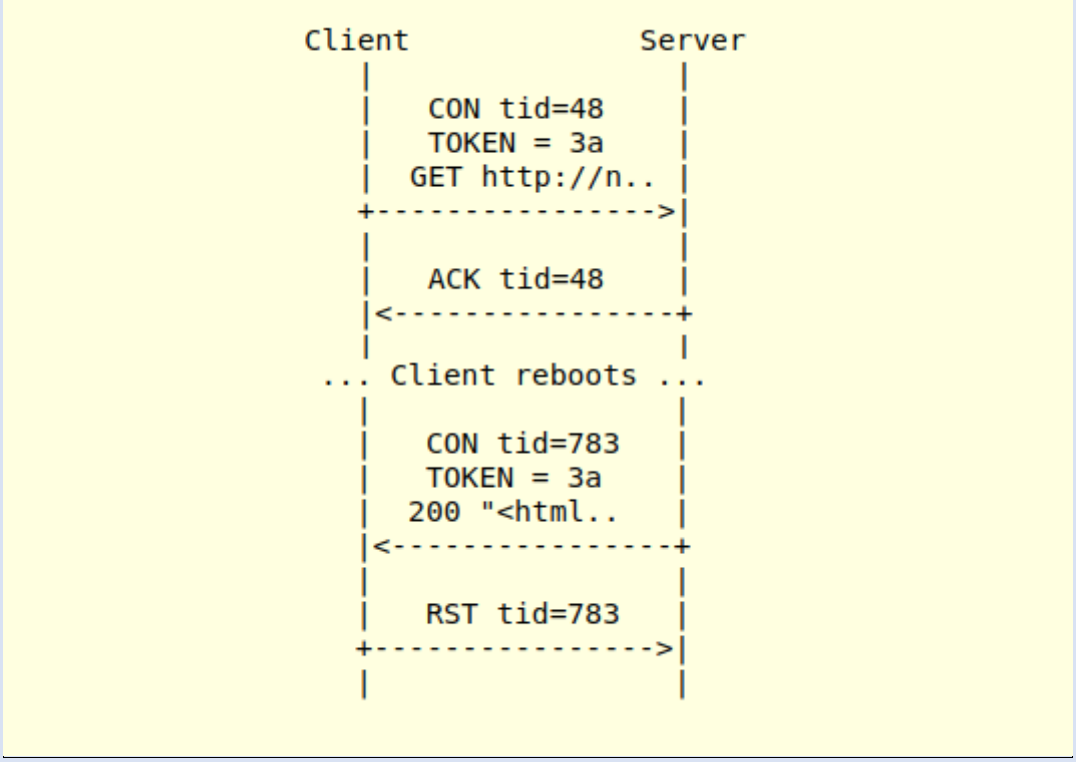
Respuesta asincrónica

El caso de respuesta sincrónica representado es el más simple, pero en otras ocasiones es necesario implementar una respuesta asincrónica. Por ejemplo, un

cliente puede solicitar a un servidor CoAP los datos de un sensor, pero éste puede no contar aún la lectura del mismo.
Si el request fue a través de un mensaje confirmable, y el cliente no recibe una respuesta en un determinado tiempo, vuelve a retransmitir la petición. Para manejar este caso, CoAP implementa un mecanismo para avisarle al cliente que recibió correctamente la solicitud enviando un ACK asociado con el ID del request, pero la respuesta la enviará en otro momento; particularmente cuando tenga los datos del sensor.
Cuando el servidor finalmente ha obtenido los datos solicitados y está listo para enviar la respuesta, inicia una transacción al cliente. Esta nueva transacción tiene su propio ID, por lo que no hay un acoplamiento automático de la respuesta a la solicitud. Para asegurar el mensaje, se vuelve a enviar como mensaje confirmable y el cliente debe responder con un ACK, asociando el nuevo ID enviado por el servidor. Opcionalmente se puede incluir un Token que vincule la solicitud y la respuesta asincrónica como vemos en esta imagen.



Adicionalmente podemos considerar una situación de falla especial. Es posible que al recibir una respuesta a un request del pasado, el cliente ya no sea consciente de que envió una solicitud, por ejemplo, si no tiene almacenamiento estable o si se reinició en ese lapso de tiempo. Para estos casos se puede devolver un mensaje del tipo Reset que anula la transacción y que está representado en esta figura.



Tipos de mensajes

Las transacciones CoAP utilizan cuatro tipos de mensajes diferentes que vamos a ver en esta sección.

> Es posible que encuentres información similar a la nombrada en los modelos de interacción, pero en este caso estamos haciendo foco puntualmente en cómo opera cada tipo de mensajes, independientemente de si la respuesta es sincrónica o asincrónica.

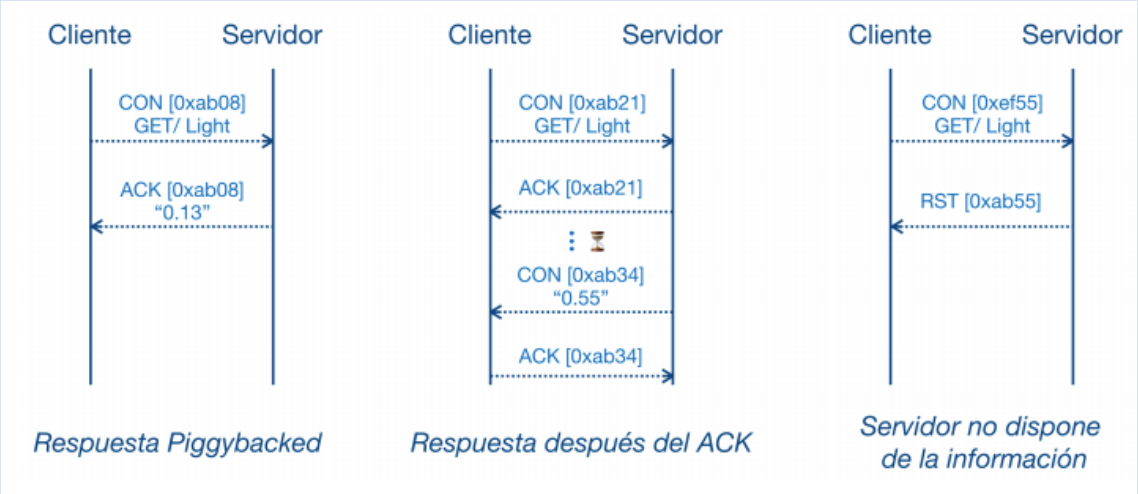
Mensajes confirmables (CON)

Este tipo de mensajes es cuando se necesita conseguir un mecanismo seguro, ya que garantiza que el mensaje llegue al destinatario. Los mensajes del tipo Confirmable constan de temporizador y backoff. Tras la transmisión de un mensaje CON se espera recibir un ACK, y podés encontrar tres situaciones posibles con este tipo de mensaje:

- 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.
- 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.

- El servidor no dispone de la información, entonces responde con un mensaje de tipo RST.

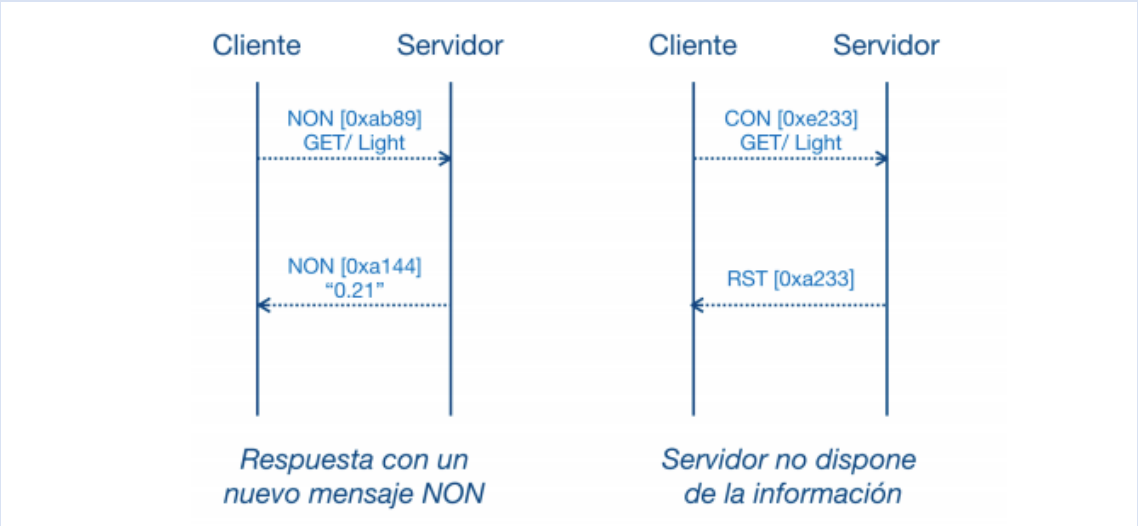
Veamos en la siguiente figura la representación de cada una de estas situaciones.



Mensajes non-confirmable (NON)

Los mensajes non confirmable (NON) no requieren una confirmación por parte del receptor, y la información solicitada será enviada en otro mensaje del tipo NON. Estos mensajes son poco fiables y se utilizan para enviar información no crítica.

Aunque este tipo de mensajes no sean confiables, igualmente tienen una identificación única. Al igual que con transmisiones de mensajes CON, existe el caso de que el servidor no pueda responder con la información solicitada; en tal caso se responderá con un mensaje RST. Podemos ver un ejemplo de comunicación utilizando este tipo de mensaje en la figura siguiente.



Mensajes de acuse de recibo (ACK)

Un mensaje de acuse de recibo reconoce que llegó un mensaje confirmable específico identificado por su ID de transacción. Al igual que con todos los tipos de mensajes en sí, puede tener una carga útil - por ejemplo, para los casos de respuesta Piggybacked - y algunas opciones para proporcionar más detalles al respecto.

Mensajes de reinicio (RST)

Un mensaje de reinicio indica que se recibió un mensaje específico, pero falta información contexto para procesarlo correctamente. Esta condición generalmente se produce cuando el nodo receptor se ha reiniciado y no ha persistido de manera adecuada la petición enviada anteriormente, o bien cuando un servidor realiza la cancelación de una transacción.

ID de transacción

El ID de transacción es un número entero mantenido por un nodo CoAP para todos los mensajes confirmables o no confirmables que envía. Cada nodo CoAP mantiene una variable única de ID de transacción que cambia cada vez que se envía un nuevo mensaje confirmable o no confirmable, independientemente de la dirección o el puerto de destino.

El ID de transacción se utiliza para hacer coincidir un acuse de recibo ACK con una solicitud pendiente, para transmitir y también descartar mensajes duplicados. El ID de transacción debe ser aleatorio y no debe reutilizarse dentro de la ventana de transmisión potencial, calculada de la siguiente forma: RESPONSE_TIMEOUT * (2 ^ MAX_RETRANSMIT - 1).

Métodos

CoAP admite los métodos GET, POST, PUT y DELETE, que se pueden mapear fácilmente con HTTP, y en esta sección vamos a describir brevemente el comportamiento de cada uno.

GET

El método GET obtiene la información del recurso identificado por el URI del request. En caso de éxito, el servidor devuelve un código 200 (OK) como respuesta. Opcionalmente, la respuesta a un GET se puede almacenar en caché si cumple con los requisitos adecuados.

POST

El método POST se utiliza para crear un nuevo recurso en el servidor. Si el recurso se ha creado correctamente, el código de respuesta debería ser 201 (Creado), incluyendo el URI del nuevo recurso. Si el POST se ejecuta correctamente, pero no se crea ningún nuevo recurso en el servidor, debería devolverse un código de respuesta 200 (OK). Las respuestas a este método no se pueden almacenar en caché ya que van a ser distintas cada vez que se haga un llamado.

PUT

El método PUT se utiliza para actualizar el recurso identificado por el URI del request. Si existe un recurso en ese URI, el cuerpo del mensaje debe contener las modificaciones del recurso, y debería devolverse una respuesta 200 (OK).

Si no existe ningún recurso en la URI del request, el servidor puede crear uno nuevo con ese URI devolviendo un código 201 (Creado) como respuesta. Si el servidor no

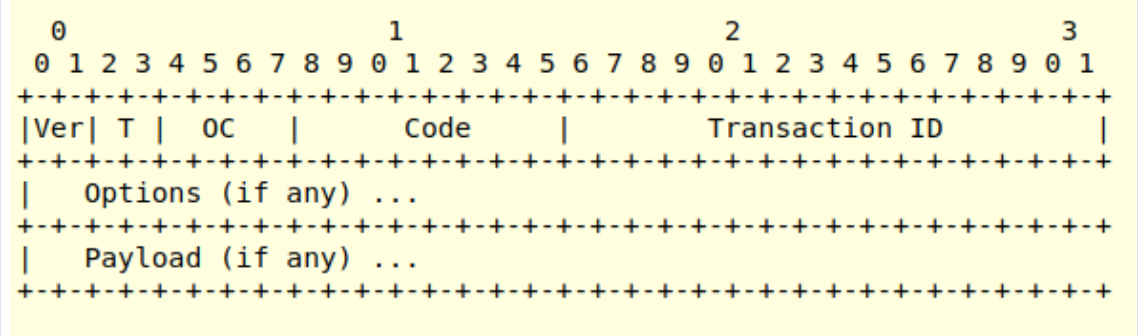
pudo crear el recurso, entonces debería enviar un código de error apropiado. Del mismo modo que con el método POST, las respuestas a este método no se pueden almacenar en caché.

DELETE

El método DELETE solicita que se elimine el recurso identificado por el URI del request. En caso que se pueda eliminar correctamente el recurso, el servidor debería devolver un código 200 como respuesta. Así mismo, con este método las respuestas no se pueden almacenar en caché.

Formato de mensajes

Las transacciones en CoAP usan un formato de encabezado base que puede ir seguido de distintas opciones de manera optativa. Los bytes posteriores al encabezado del paquete se consideran el payload del mensaje, si corresponde. La longitud del payload del mensaje está regida por la longitud de un datagrama UDP. El header se utiliza en todas las transacciones de CoAP, ya sea para requests o responses con cualquiera de los tipos de mensajes y con cualquier método CoAP. En la imagen siguiente podés ver la descripción de un header CoAP.



- **Ver (versión)**: este campo contiene 2 bits que indican la versión CoAP utilizada.
- **T (type)**: este campo contiene 2 bits que indican el tipo de mensaje. Puede tomar cuatro valores, que son Confirmable, Non-confirmable, Acknowledgement y Reset.
- **OC (option count)**: este campo sirve para hacer recuento de opciones Son 4 bits que indican si hay opciones después del header base. Si se establece en 0, el payload del mensaje sigue inmediatamente al header base. Si es mayor que cero, el campo indica el número de opciones que seguirán inmediatamente al encabezado base.
- **Code**: 8 bits que indican el método o el código de respuesta de un mensaje. El valor 0 indica que no hay código. Los valores del 1-10 se utilizan para los códigos de método CoAP, los valores 11-39 se reservan

para uso futuro y los valores 40-255 se utilizan para los códigos de respuesta.
- **Transaction ID**: 16 bits que indican el ID del mensaje. Permite evitar duplicidad y facilitar mensajes del tipo Acknowledgement y Reset.

A continuación del header, pueden definirse las opciones y a continuación el payload del mensaje. El payload del mensaje estará atado a la aplicación en particular. Veamos algunas opciones que se pueden incluir en un mensaje.

- **Content-Type**: La opción de tipo de contenido indica el tipo de medio de internet del cuerpo del mensaje, que está definido por una tabla. Se debe incluir una opción de identificador de tipo de contenido si hay payload en un mensaje CoAP. Si no existe esta opción, se asume el formato text/plain.
- **Max Age:** La opción Max-age indica la antigüedad en segundos del recurso para su uso en el control de caché. El valor de la opción es un entero variable entre 8 y 32 bits. Se asume un valor predeterminado de 60 segundos en ausencia de esta opción.
- **Token**: La opción Token es una secuencia de 1-2 bytes que se usa para hacer coincidir request y un response que pueden enviarse de manera asincrónica. El token es inicialmente generado por un cliente cuando ejecuta un request. Este valor es conservado por el servidor, que al momento de devolver la respuesta aplica el token en el header de respuesta.

Caché

Los dispositivos CoAP generalmente están limitados por el ancho de banda y la potencia de procesamiento. Para optimizar el rendimiento de la transferencia de datos bajo estas restricciones, se utilizan funciones de almacenamiento en caché consistentes con HTTP.
La vida del caché - es decir por cuánto tiempo se puede mantener - se controla mediante la opción de encabezado Max-Age; la actualización del caché y el control de versiones de un recurso se controlan a través de la opción de encabezado Etag; y los proxies entre un cliente y endpoint pueden participar en el proceso de almacenamiento en caché en nombre de los nodos para evitar tráfico innecesario en la red restringida.

Control de caché

Cuando un endpoint responde a una solicitud GET, debe especificar la opción de encabezado Max-Age. Max-Age determina la vida útil de la caché del recurso en

segundos. Los recursos que cambian rápidamente tendrán una vida de caché corta, y los recursos que cambian con poca frecuencia deben especificar una vida de caché larga. La opción por defecto son 60 segundos. Si se quiere deshabilitar la opción de caché se debe especificar el valor 0.
Cuando un cliente lee la respuesta de una solicitud GET, debe almacenar en caché la representación del recurso durante la vida útil de la caché según lo especificado por el encabezado Max-Age. Durante el tiempo de duración, el cliente debe usar su versión en caché y evitar realizar GET adicionales hacia el recurso.
En general, es el endpoint del servidor el responsable de determinar la antigüedad de la caché, aunque en algunos casos, es posible que el cliente desee determinar su propia tolerancia para la obsolescencia de la caché. En este caso, el cliente puede especificar la opción Max-Age en el header del request GET, y si el Max-Age soportado por el cliente es menor que el del endpoint, el servidor - o proxy - debe actualizar su valor de Max-Age para el response en cuestión.

Refresh de caché

Después de la expiración del caché, los clientes y los servidores proxy pueden actualizar su representación de un recurso. La actualización de la caché se logra mediante un request GET que devolverá el estado actual del recurso.

Proxy

Un proxy es una entidad CoAP que atiende requests almacenadas en caché en nombre de otras entidades CoAP. Cualquier nodo en una red CoAP puede actuar como un proxy, aunque en general suele ser el nodo entre la red restringida e internet quien implementa la funcionalidad. El uso de un proxy es necesario en los siguientes escenarios:

- Cuando hay clientes externos a la red restringida que desean acceder a la información de los nodos. En estos casos, los clientes externos deben acceder al proxy en vez de a los servidores CoAP directamente.
- Cuando se desee acceder a los recursos de los server CoAP dentro de la misma red restringida, pero se debe ajustar el rendimiento de la red de manera más óptima.
- Cuando los clientes de los dispositivos desean acceder a la información de los recursos, pero los dispositivos se encuentran en modo bajo consumo.

Otros aspectos

Hasta el momento vimos de manera general las características más importantes del protocolo, pero aún quedan muchos otros aspectos de interés. En esta sección vamos a describir algunos.

Retransmisión de mensajes

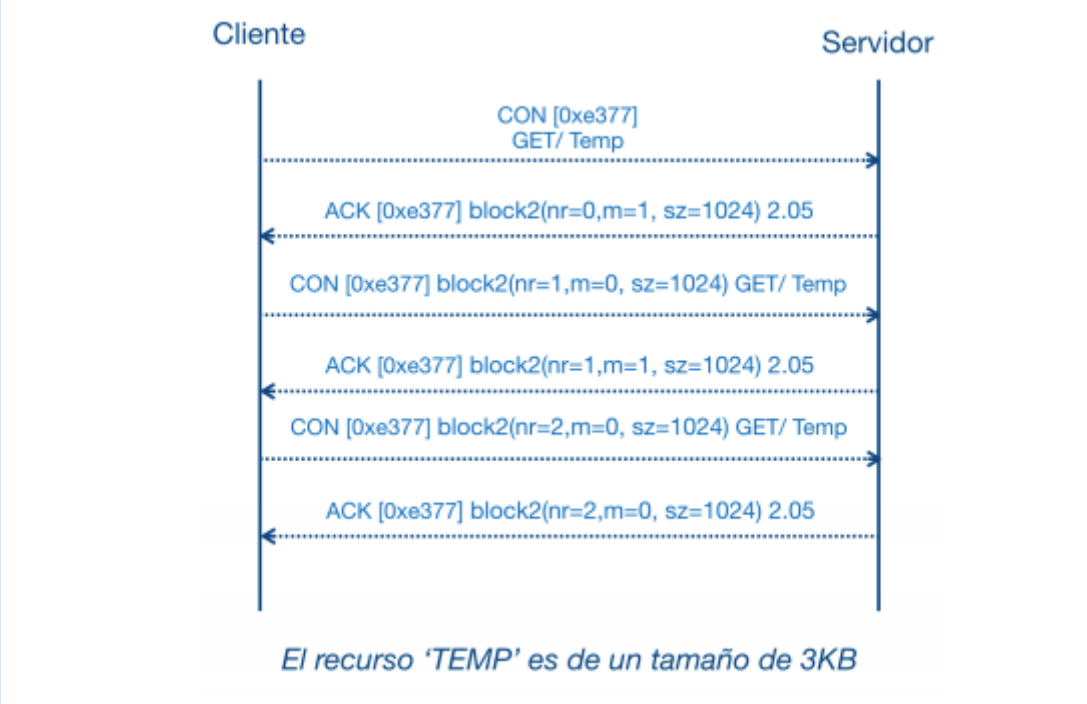
Una entidad CoAP realiza un seguimiento de los mensajes confirmables abiertos que envió y que están esperando una respuesta. Cada mensaje incluye al menos una dirección IP y un puerto de destino, el payload del mensaje, un contador de retransmisión y un tiempo de espera.
El seguimiento de los mensajes se realiza a través de un registro con un tiempo de espera inicial predeterminado - identificado como RESPONSE_TIMEOUT - y estableciendo el contador de retransmisión en 0. Cuando se recibe un ACK coincidente con el mensaje el registro se elimina, ya que la operación se encuentra confirmada.
Cuando se alcanza el valor de RESPONSE_TIMEOUT sin haber llegado al máximo de retransmisiones - identificadas por el valor MAX_RETRANSMIT -, el mensaje original se retransmite al destino sin modificación, el contador de retransmisión se incrementa y el tiempo de espera se duplica.
Si el contador de retransmisiones llega a MAX_RETRANSMIT, el registro abierto para ese mensaje se elimina y se informa que ha ocurrido un error.

Envío de bloques

Debido a la naturaleza restringida de las redes y dispositivos que ejecutan CoAP, para los casos donde es necesario enviar una elevada cantidad de datos se debe hacer uso de la funcionalidad Block-Wise Transfers. Básicamente se divide un bloque de información en bloques más pequeños, y entre las principales características de la funcionalidad están:

- Bloques de un tamaño definido entre 16 bytes y 1 kilobyte.
- Todos los bloques utilizan el mismo tipo 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.

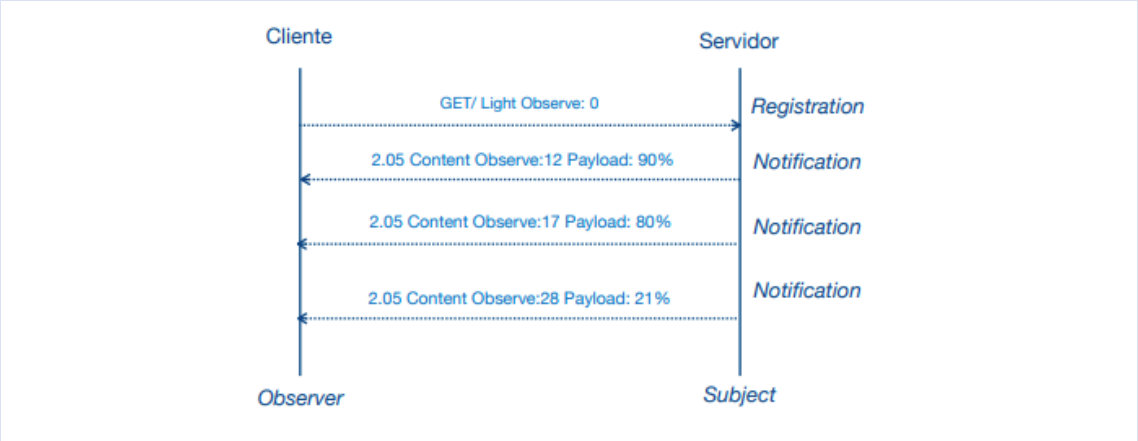
Al igual que con los tipos de mensaje, existen diferentes diagramas dependiendo de la negociación entre los dispositivos, y en la siguiente figura podemos ver el caso más simple.



El recurso 'TEMP' es de un tamaño de 3KB

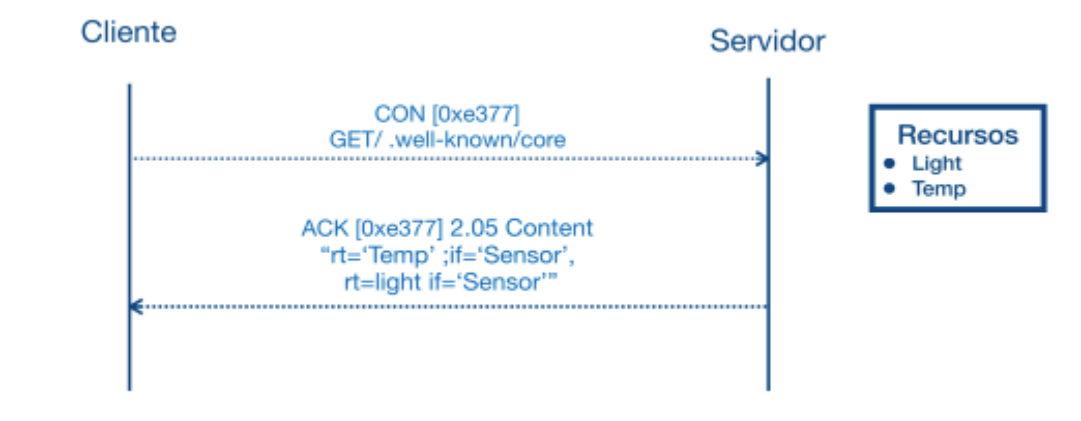
Observación de recursos

Un cliente CoAP puede acceder a los recursos que el servidor ofrezca ejecutando los requests que vimos anteriormente. Sin embargo, si una entidad desea recibir el valor de un recurso cuando éste sufra una modificación, es necesario el uso de la observación de recursos. En esta funcionalidad se define un sujeto, un observador, el mensaje de registro y los mensajes de notificación. Veamos un ejemplo en este diagrama.



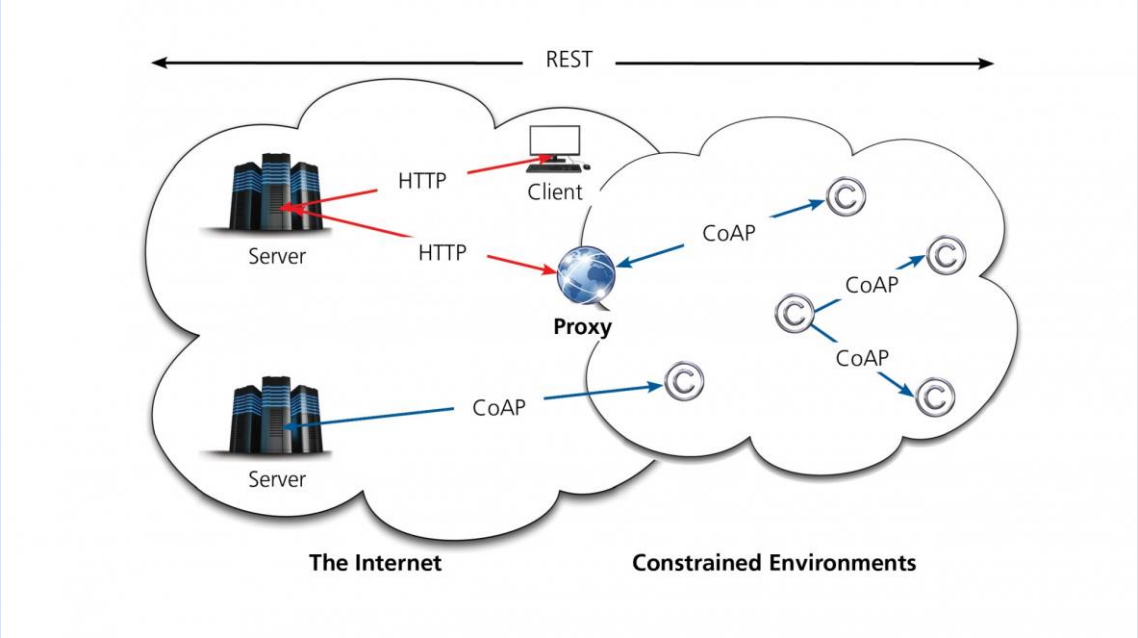
Descubrimiento de recursos

El descubrimiento de los recursos que puede ofrecer un servidor CoAP es una funcionalidad muy importante en entornos M2M, ya que permite la detección automática de las posibilidades que ofrece cada entidad. Para estos casos, existe una petición especial conocida como /.well-known que el cliente debe realizar. En el diagrama siguiente hay una descripción de esta funcionalidad.



Mapeo CoAP-HTTP

El mapeo entre CoAP y HTTP es utilizado en varios escenarios, donde dos posibles resultan característicos. Por un lado, cuando se necesita implementar una interfaz web HTTP para utilizarla sobre CoAP, y por otro, implementar un proxy HTTP-CoAP. En la siguiente figura podemos ver un caso utilizando un proxy para llegar desde un cliente web HTTP a una red CoAP.



Tal como vimos, los métodos que implementa CoAP son GET, POST, PUT & DELETE. Esto permite hacer un mapeo directo entre los dos protocolos en la mayoría de los casos. El mapeo de CoAP a HTTP es sencillo, se realiza una conversión directa de los métodos 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.

Códigos

CoAP utiliza un subconjunto de los códigos de estado HTTP definidos, además de algunos códigos de estado específicos de CoAP. Para reducir el tamaño de los headers, el código tiene 8 bits, y en la siguiente tabla podemos ver el mapeo correspondiente.

Code	HTTP Name
40	100 Continue
80	200 OK
81	201 Created
124	304 Not Modified
160	400 Bad Request
164	404 Not Found
165	405 Method Not Allowed
175	415 Unsupported Media Type

Code	HTTP Name
200	500 Internal Server Error
202	502 Bad Gateway
203	503 Service Unavailable
204	504 Gateway Timeout
240	Token Option required by server
241	Uri-Authority Option required by server
242	Critical Option not supported

Tipos de contenido (Content-Type)

Los tipos de contenido en HTTP se identifican con un string, como por ejemplo application/json. Esta cadena se compone de un tipo de nivel superior - aplicación en este caso - y un subtipo, como por ejemplo json.
Para minimizar el tamaño de los headers - y evitar la sobrecarga -, en vez de usar strings, CoAP codifica los mensajes en un único byte que es un subconjunto de los que se pueden manejar con HTTP. Veamos en esta tabla las equivalencias de los tipos de contenido.

Internet media type	Identifier
text/plain (UTF-8)	0
text/xml (UTF-8)	1
text/csv (UTF-8)	2
text/html (UTF-8)	3
image/gif	21
image/jpeg	22
image/png	23
image/tiff	24
audio/raw	25
video/raw	26
application/link-format	40

Internet media type	Identifier
application/xml	41
application/octet-stream	42
application/rdf+xml	43
application/soap+xml	44
application/atom+xml	45
application/xmpp+xml	46
application/exi	47
application/x-bxml	48
application/fastinfoset	49
application/soap+fastinfoset	50
application/json	51

Implementaciones

El protocolo CoAP es suficientemente simple como para implementarlo desde cero, aunque esta no es una opción disponible en muchos casos. Por suerte existen implementaciones del protocolo para varias plataformas, muchas de estas implementaciones permanecen privadas, pero algunas se publican con licencias de código abierto, como Apache 2.0 o licencia MIT.

Implementaciones en sistemas restringidos

- **microcoap** [link]: Una implementación de C que se puede compilar para entornos Arduino y POSIX.
- **Lobaro CoAP** [link]: Una implementación de C con licencia MIT para dispositivos restringidos que cubre CoAP base, Observe y Block, con demostraciones para ESP8266 y ZWIR4512 (Cortex M3).
- **SMCP** [link]: Una pila CoAP basada en C, pequeña pero capaz, adecuada para dispositivos embebidos. Admite observaciones, respuestas asincrónicas a solicitudes y más. Las funciones que no se utilizan se pueden eliminar para reducir la huella (footprint).
- **libcoap** [link]: Una implementación C de CoAP que se puede usar tanto en dispositivos restringidos (que ejecutan sistemas operativos como Contiki o LWIP) como en un sistema POSIX más grande. Además, la biblioteca se ha portado a TinyOS y RIOT.
- **Erbium** [link]: Contiki es un sistema operativo ampliamente utilizado para nodos restringidos, que se emplea para investigación y desarrollo de productos. Erbium es un motor REST completo y una implementación de CoAP para Contiki.

Implementaciones de Servidores CoAP

- **Java**
    - Una implementación significativa de CoAP basada en Java es Californium.
    - nCoAP es una implementación Java del protocolo CoAP que utiliza Netty NIO.
    - Leshan es una implementación del lado del servidor OMA Lightweight M2M (LWM2M), además de Californium.
- **C/C++**
    - Varias de las implementaciones de dispositivos restringidos, como libcoap, también se pueden usar en el lado del servidor.
- **JavaScript** (node.js)
    - node-coap es una biblioteca de cliente y servidor para CoAP modelada según el módulo http.
    - CoAP-CLI es una interfaz de línea de comandos para CoAP, construida sobre node.js y node-coap.
- **Python**

- txThings es una biblioteca CoAP basada en Twisted de Python.
- aiocoap implementa CoAP de forma nativa con mecanismos de asyncio de Python 3.4 y proporciona herramientas de línea de comandos para la búsqueda de recursos y el proxy.
- CoAPthon es una biblioteca de Python para el protocolo CoAP.

> La descripción de las implementaciones que vimos en esta sección está descrita en la documentación de CoAP, podés consultar esa fuente para obtener información actualizada.

Conclusiones

La definición de CoAP abarca lo que vimos hasta ahora y muchos otros aspectos que quedaron fuera de la introducción inicial. Si te interesa el tema es recomendable que vayas a la documentación oficial del protocolo y comiences a investigar los aspectos más relevantes. Para hacer un resumen, en este artículo vimos los siguientes temas.

- Una introducción general sobre el protocolo CoAP.
- En qué escenarios es conveniente que utilices el protocolo.
- Los modelos de interacción y los tipos de respuestas sincrónicas y asincrónicas.
- Las características más importantes sobre los tipos de mensajes CoAP.
- Los métodos que admite el protocolo.
- La descripción sobre cómo están formados los mensajes.
- Las características que permiten correr CoAP sobre UDP.
- El funcionamiento del caché para poder optimizar el uso de la red.
- Otros aspectos importantes como el envío de bloques, observación y descubrimiento de recursos, el mapeo de CoAP y HTTP, los códigos de los mensajes y los tipos de contenido admitidos.
- Algunas implementaciones del protocolo para dispositivos restringidos como así también en diferentes lenguajes de programación.

Bibliografía

- IETF Tools, CoAP draft - [Disponible: Julio 2021]
- Sitio oficial de CoAP - [Disponible Julio 2021]
- Estudio de implementaciones del CoAP - [Disponible Julio 2021]
- IP vs CoAP IoT communication - [Disponible Julio 2021]
- ¿Qué es CoAP? - [Disponible Julio 2021]
- CoAP vs MQTT - [Disponible Julio 2021]
- Radiocraft, CoAP protocol - [Disponible Julio 2021]
- RF Wireless, CoAP protocol - [Disponible Julio 2021]