**CoAP Protocol**

**Step-by-Step Guide**

This article describes what is CoAP and how to use it in on IoT devices. CoAP is an IoT protocol that has interesting features specifically designed for constrained devices. There are other IoT protocols useful to build IoT solution, such as MQTT and so on.

IoT is one of the most interesting and promising technology trends. It's an ecosystem where objects, people, devices are interconnected and exchange data. In this blog, we have covered IoT from several points of view, developing IoT projects and covering several aspects related to IoT.

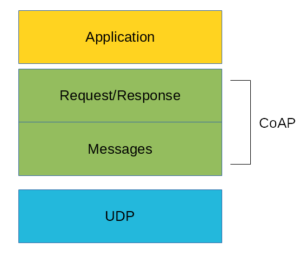
## What Is CoAP Protocol?

As said before, CoAP is an IoT protocol. CoAP stands for Constrained Application Protocol, and it is defined in [RFC 7252](https://tools.ietf.org/html/rfc7252). CoAP is a simple protocol with low overhead specifically designed for constrained devices (such as microcontrollers) and constrained networks. This protocol is used in M2M data exchange and is very similar to HTTP, even if there are important differences that we will cover laters.

The main features of CoAP protocols are:

* Web protocol used in M2M with constrained requirements
* Asynchronous message exchange
* Low overhead and very simple to parse
* URI and content-type support
* Proxy and caching capabilities

As you may notice, some features are very similar to HTTP even if CoAP must not be considered a compressed HTTP protocol because CoAP is specifically designed for IoT and in more details for M2M so it is very optimized for this task.

From the abstraction protocol layer, CoAP can be represented as:

As you can see there are two different layers that make CoAp protocol: Messages and Request/Response. The Messages layer deals with UDP and with asynchronous messages. The Request/Response layer manages request/response interaction based on request/response messages.

CoAP supports four different message types:

* Confirmable
* Non-confirmable
* Acknowledgment
* Reset

Before going deeper into the CoAp protocol, structure is useful to define some terms that we will use later:

* **Endpoint**: An entity that participates in the CoAP protocol. Usually, an Endpoint is identified with a host
* **Sender**: The entity that sends a message
* **Recipient**: The destination of a message
* **Client**: The entity that sends a request and the destination of the response
* **Server**: The entity that receives a request from a client and sends back a response to the client

## CoAP Messages Model

This is the lowest layer of CoAP. This layer deals with UDP exchanging messages between endpoints. Each CoAP message has a unique ID; this is useful to detect message duplicates. A CoAP message is built by these parts:

* A binary header
* A compact options
* Payload

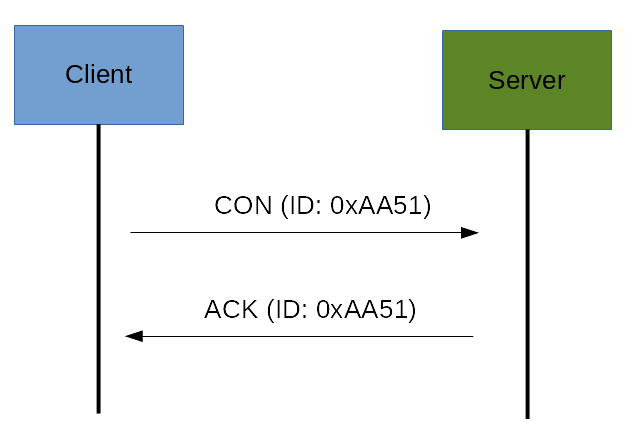
Later, we will describe the message format in more details.

As said before, the CoAP protocol uses two kinds of messages:

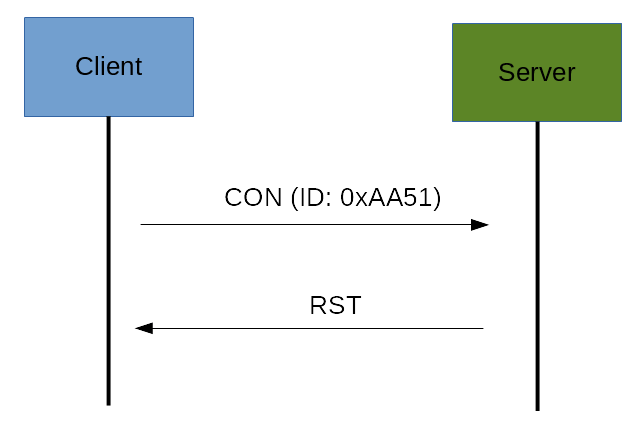
* Confirmable message
* Non-confirmable message

A confirmable message is a reliable message. When exchanging messages between two endpoints, these messages can be reliable. In CoAP, a reliable message is obtained using a Confirmable message (CON). Using this kind of message, the client can be sure that the message will arrive at the server. A Confirmable message is sent again and again until the other party sends an acknowledge message (ACK). The ACK message contains the same ID of the confirmable message (CON).

The picture below shows the message exchange process:

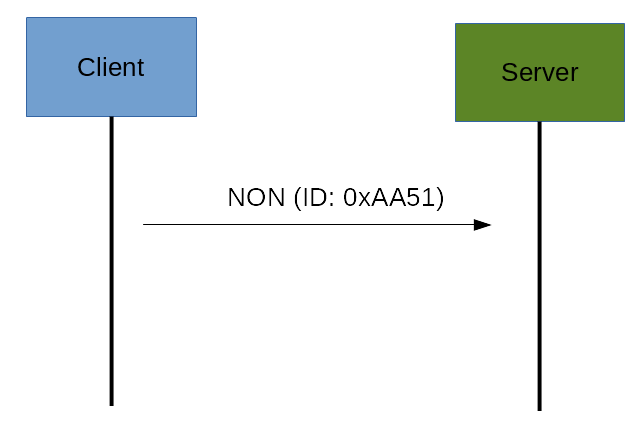


If the server has troubles managing the incoming request, it can send back a Rest message (RST) instead of the Acknowledge message (ACK):



The other message category is the Non-confirmable (NON) messages. These are messages that don’t require an Acknowledge by the server. They are unreliable messages or in other words messages that do not contain critical information that must be delivered to the server. To this category belongs messages that contain values read from sensors.

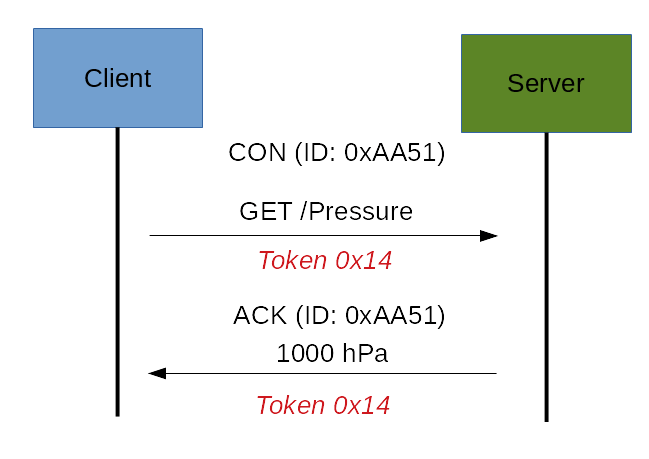
Even if these messages are unreliable, they have a unique ID.



## CoAp Request/Response Model

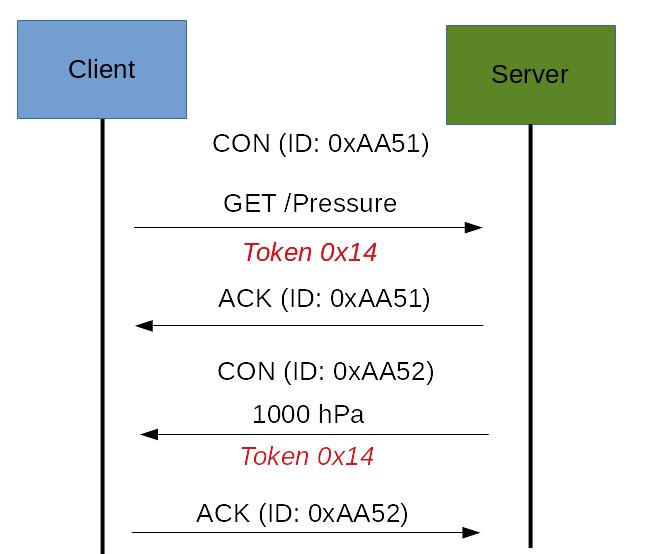
The CoAP Request/Response is the second layer in the CoAP abstraction layer. The request is sent using a Confirmable (CON) or Non-Confirmable (NON) message. There are several scenarios depending on if the server can answer immediately to the client request or the answer if not available.

If the server can answer immediately to the client request, then if the request is carried using a Confirmable message (CON), the server sends back to the client an Acknowledge message containing the response or the error code:



As you can notice in the CoAP message, there is a Token. The Token is different from the Message-ID and it is used to match the request and the response.

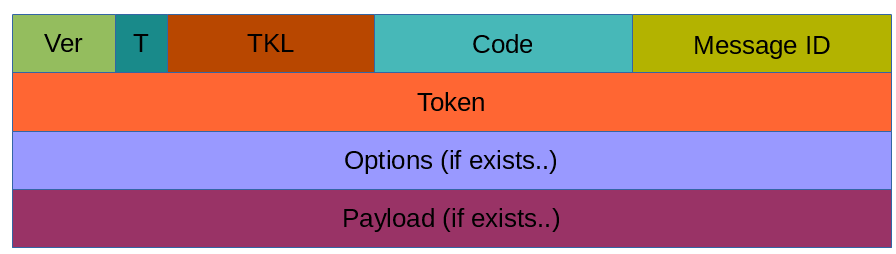
If the server can’t answer to the request coming from the client immediately, then it sends an Acknowledge message with an empty response. As soon as the response is available, then the server sends a new Confirmable message to the client containing the response. At this point, the client sends back an Acknowledge message:



If the request coming from the client is carried using a NON-confirmable message, then the server answer using a NON-confirmable message.

## CoAp Message Format

This paragraph covers the CoAP Message format. By now, we have discussed different kinds of messages exchanged between the client and the server. Now it is time to analyze the message format. The constrained application protocol is the meat for constrained environments, and for this reason, it uses compact messages. To avoid fragmentation, a message occupies the data section of a UDP datagram. A message is made by several parts:



Where:

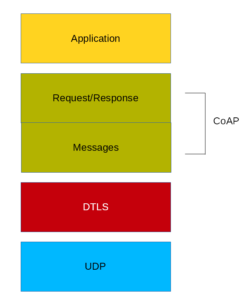
* **Ver**: It is a 2 bit unsigned integer indicating the version
* **T**: it is a 2 bit unsigned integer indicating the message type: 0 confirmable, 1 non-confirmable
* **TKL**: Token Length is the token 4 bit length
* **Code**: It is the code response (8 bit length)
* **Message ID**: It is the message ID expressed with 16 bit
* And so on.

More useful resources:

* [MQTT protocol tutorial](https://www.survivingwithandroid.com/2016/10/mqtt-protocol-tutorial.html)
* [IoT protocols overview](https://www.survivingwithandroid.com/2016/08/iot-protocols-list.html)
* [Getting Started With MQTT (DZone Refcard)](https://dzone.com/refcardz/getting-started-with-mqtt)

## CoAP Security Aspects

One important aspect when dealing with IoT protocols is the security aspects. As stated before, CoAP uses UDP to transport information. CoAP relies on UDP security aspects to protect the information. As HTTP uses TLS over TCP, CoAP uses Datagram TLS over UDP. DTLS supports RSA, AES, and so on. Anyway, we should consider that in some constrained devices some of DTLS cipher suits may not be available. It is important to notice that some cipher suites introduce some complexity and constrained devices may not have resources enough to manage it.



## CoAP Vs. MQTT

An important aspect to cover is the main differences between CoAP and MQTT. As you may know, [MQTT](https://www.survivingwithandroid.com/2016/10/mqtt-protocol-tutorial.html) is another protocol widely used in IoT. There are several differences between these two protocols. The first aspect to notice is the different paradigm used. MQTT uses a publisher-subscriber while CoAP uses a request-response paradigm. MQTT uses a central broker to dispatch messages coming from the publisher to the clients. CoAP is essentially a one-to-one protocol very similar to the HTTP protocol. Moreover, MQTT is an event-oriented protocol while CoAP is more suitable for state transfer.

Following table compares various features of COAP vs MQTT and tabulates difference between CoAP and MQTT protocols.

|  |  |  |
| --- | --- | --- |
| **Features** | **CoAP** | **MQTT** |
| Full Form | Constrained Application Protocol | Message Queue Telemetry Transport |
| Model used for communication | Request-Response, Publish-Subscribe | Publish-Subscribe |
| RESTful | Yes | No |
| Transport layer | Preferably UDP, TCP can also be used. | Preferably TCP, UDP can also be used (MQTT-S). |
| Header Size | 4 Bytes | 2 Bytes |
| Number of message types used | 4 | 16 |
| Messaging | Asynchronous & Synchronous | Asynchronous |
| Application Reliability | 2 Levels | 3 Levels |
| Security | IPSEC or DTLS | Not defined in the standard |
| Intermediaries | YES | YES (MQTT-S) |
| LLN Suitability (thousand nodes) | Excellent | Fair |
| Application success stories | Utility Field Area Networks | Extending enterprise messaging into IoT applications |

## **Comparison**

MQTT and CoAP are both useful as IoT protocols, but have fundamental differences.

MQTT is a many-to-many communication protocol for passing messages between multiple clients through a central broker. It decouples producer and consumer by letting clients publish and having the broker decide where to route and copy messages. While MQTT has some support for persistence, it does best as a communications bus for live data.

CoAP is, primarily, a one-to-one protocol for transferring state information between client and server. While it has support for observing resources, CoAP is best suited to a state transfer model, not purely event based.

MQTT clients make a long-lived outgoing TCP connection to a broker. This usually presents no problem for devices behind NAT. CoAP clients and servers both send and receive UDP packets. In NAT environments, tunnelling or port forwarding can be used to allow CoAP, or devices may first initiate a connection to the head-end as in LWM2M.

MQTT provides no support for labelling messages with types or other metadata to help clients understand it. MQTT messages can be used for any purpose, but all clients must know the message formats up-front to allow communication. CoAP, conversely, provides inbuilt support for content negotiation and discovery allowing devices to probe each other to find ways of exchanging data.

Both protocols have pros and cons, choosing the right one depends on your application.