# CoAP: Get started with IoT protocols

By **Rajesh Sola** - September 30, 2016

*CoAP is a software protocol that allows simple electronic devices to communicate over th Internet. It is designed for small devices with low-power sensors and actuators that need be controlled or supervised remotely, through standard Internet networks.*

Get real-time push notifications

Network protocols play a significant role in communicating between various building blocks of an IoT architecture. You might wish for an efficient protocol that connects gateways with sensor nodes as per M2M or WSN needs, and with servers or cloud platforms for Web integration and global access. HTTP may be fit for certain needs but is expensive and has its own overheads. CoAP is a simple, less expensive protocol that meets all the above needs and is affordable for those with resource constraints.

Constrained Application Protocol (CoAP) is a simple, low overhead protocol designed for environments like low-end microcontrollers and constrained networks with critical bandwidth and a high error rate such as *6LowPANs*. It is defined by the IETF open standard RFC 7252, runs on UDP by default but is not limited to it, as it can be implemented over other channels like TCP, DTLS or SMS. A typical networking stack offering CoAP is shown in Figure 1. CoAP is based on the request-response communication

model and comes with support for resource discovery, better reliability, URIs, etc. The protocol was designed for M2M needs initially, but was adapted in IoT as well, with support on gateways, high-end servers and enterprise integration.

CoAP resembles HTTP in terms of the REST model with *GET, POST, PUT* and *DELETE* methods, URIs, response codes, MIME types, etc, but one shouldn't think of it as compressed HTTP. However, CoAP can easily interface with HTTP using proxy components, where HTTP clients can talk to CoAP servers and vice versa, which enables better Web integration and the ability to meet IoT needs.

Let's have a quick overview of the protocol.

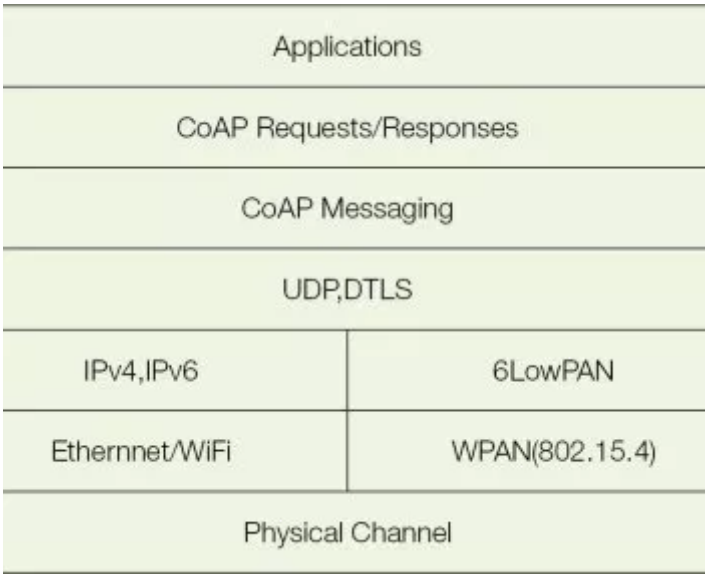


*Figure 1: Protocol stack with CoAP support*

Get real-time push notifications

**Endpoints**

A host or node participating in CoAP communication is known as an endpoint. The endpoint on which resources are defined and is a destination for requests is known as a server or, more precisely, the origin server and the endpoint from which requests are made for target resources is known as a client. Similarly, a server is the source and a client is the destination for responses. Certain endpoints like proxies act as the intermediate client and server.

**The CoAP message format**

One of the key design goals of CoAP is to avoid fragmentation at underlying layers, especially at the link layer, i.e., the whole CoAP packet should fit into a single datagram compatible with a single frame at the Ethernet or IEEE 802.15.4 layer. This is possible with a compact 4-byte binary header, optional fields and payload, as shown in Figure 2.

**Version:** 2-bit version number, currently fixed at 0x01

**Type of messages:** CoAP supports four types of messages with the following 2 bit transaction codes:

| | | | |
|---|---|---|---|
| CON | 00(0) | ACK | 10(2) |
| NON | 01(1) | RST | 11(3) |

CoAP offers optional reliability using conformable (CON) messages, where each message (request/response) should be acknowledged (ACK) by the peer endpoint. Messages will be retransmitted if a CON message is not acknowledged within the time out. A response may be combined with an ongoing acknowledgement, which is known as a piggy backed response, or the response may be sent afterwards if not available immediately, which is known as a separate response. Piggy backed responses need not be acknowledged, and the same holds for non conformable (NON) messages.

**Req/Resp code:** A 3-bit class ID and 5-bit detail in the c.dd format forms this field. The class values used for requests, success responses, client error responses and server erro responses are 0, 2, 4 or 5, respectively. The detail carries the request code or response code depending on the class value. Code 0.00 indicates an empty message.

Get real-time push notifications

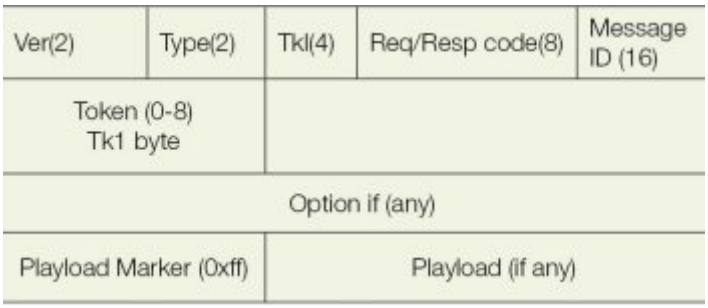


*Figure 2: CoAP message format*

**Message ID:** A 16-bit unsigned number in network byte order is used to match acknowledgement or reset messages and eliminate duplicate messages.

**Tokens:** An optional token field, which is limited to 0 to 8 bytes currently to match request responses, may be kept after the header, which is of TKL number of bytes specified in the header.

**Options:** Zero or more option fields may follow a token. A few options are Content Format, Accept, Max-Age, Etag, Uri-Path, Uri-Query, etc.

**CoAP URIs:** CoAP URIs consist of the hostname, port number, path and query string, which are specified by option fields Uri-Host, Uri-Port, Uri-Path and Uri-Query, of which Uri-Host and Uri-Port are implicit as they are part of underlying layers. Uri-Path and Uri-Query

are significant and part of the CoAP message. For example, if we request a resource with URI *coap://hostname:port/leds/red?q=state&on*, the following options are generated:

```
Option#1 Uri-Path Leds,
Option#2 Uri-Path red,
Option#3 Uri-Query q=state
Option#4 Uri-Query on.
```

CoAP servers run on UDP port no.5683 by default.

**Payload:** Followed by the optional token and zero or more option fields, the payload may start, in which case a payload marker 0xFF is placed inbetween. This indicates the end of the options and the start of the payload. The absence of the payload marker indicates an empty payload.


*Figure 3: Firefox Copper plugin*

### Request methods

CoAP has come up with a subset of the Representational State Transfer [REST] architectu of the Web, which resembles HTTP with request methods GET(1), POST(2), PUT(3) and DELETE(4), with indicated request codes. These methods can be used to create, update, query and delete the resources on the server representing events of an IoT application. F example, a server getting sensor readings directly can define suitable resources and provide data when the GET method is requested by clients, or sensor readings can be updated by using the POST method by a client holding sensor values. Besides, resources can be updated using the GET method with a query string. Similarly, clients can use the POST method to update resources which may represent actuators. However, the PUT method is preferred to POST for idempotent operations.

A *Discover* operation is added with a *GET* request on the predefined path */.well-known/core,* which returns a list of available resources along with applicable paths in the response. A ping operation is implemented by sending an empty message to which the peer responds with an empty reset (RST) message indicating the‘liveness’of the endpoint. Additionally, the *Observe* method is defined, where the notification is sent periodically or on an event basis, for a single request. If the request is a CON message, each response is of the CON type and will be acknowledged by the client. *Observe* can be initiated with an extended GET request, with the added *Observe* option set to zero. It can be cancelled by

Get real-time push notifications

sending a reset message for one of the notifications or by sending an explicit GET request with the *Observe* option set to value 1. Notifications use the Observe option with sequential values for reordering of delayed responses and the *Max-Age* option to keep freshness in the cache. The combination of resource update, *Observe* method is analogous to publish-subscribe mechanisms in other protocols like MQTT. CoAP also offers block wise transfers, which enable a large amount of data exceeding datagram limits, and eliminates fragmentation at underlying layers with better reliability.

**Response codes**

Most response codes are similar to HTTP. For instance, 2.xx indicates success, 4.xx indicates client error and 5.xx indicates server error. Here are some frequently encountered response codes:

| | |
|---|---|
| 2.01 | Created |
| 2.02 | Deleted |
| 2.04 | Changed |
| 2.05 | Content |
| 4.04 | Not found (resource) |
| 4.05 | Method not allowed |

This is the summary of the CoAP protocol and message format. You can refer to the listed RFC and references for more details like message ID rules, token generation, options, etc. Now, let's go ahead with some hands-on stuff. The Eclipse IoT project offers Californium core library, demo apps and add-ons to bootstrap with the protocol.

Get real-time push notifications

**Building the Eclipse Californium core demo apps**

Set up the Maven build using the recent stable version of binaries. Check out the Git repository *github.com/eclipse/californium.git* with a recent stable branch, say 1.0.4, and switch into a cloned directory, as follows:

```
mvn clean install
```

*# in case of failure at scandium-core module comment the following line in pom.xml under modules as follows*

```
<!--    <module>scandium-core</module> -->
```

*# Or rerun the command as follows*

```
mvn clean installed -rf : californium-core
```

*# collect californium-core-1.0.4.jar from californium-core/target and element-connector-1.0.4.jar from element-connector/target to a new work directory for building your own server, client apps without maven build support.*

*element-connector* is a Java socket abstraction for UDP, DTLS, TCP, etc., which provides a connector interface implemented by UDPConnector currently, and DTLS by Scandium plugin. Support for TCP, SMS can be added in future with this abstraction layer.

Get real-time push notifications

*Figure 4: coap request node is NodeRED*

**A tour of demo apps and quick test with the Firefox Copper plugin**

Switch to demo apps/run and launch the following demos as follows:

```
java -jar cf-helloworld-server-1.0.4.jar
```

This provides a single resource at path */helloWorld* supporting the GET method.
Open Firefox, install the Copper plugin using *https://addons.mozilla.org/en-US/firefox/addon/copper-270430/*. Type the URL *coap://localhost*, and hit *Ping* to check how alive the server is and hit Discover to retrieve hosted resources. Click on the *HelloWorld* resource and request the GET method, which returns the response with the payload *Hello World!* You will also notice that other methods are not allowed, which is indicated by the response code 4.05.
You can run another demo, which provides various resources supporting REST methods and also the Observe method and block wise transfer, as follows:

```
java -jar cf-plugtest-server-1.0.4.jar
```

You can also run *cf-helloworld-client-1.0.4.jar* and *cf-plugtest-client-1.0.4.jar* for testing the respective servers.

### Working with CoAP servers with custom resources

To run a server with custom resources, you can use the example code at *github.com/rajeshsola/iot-examples/coap/demo-server,* which defines the following resources with supported operations. In this demo, the resources represent dummy information in order to run without specific hardware.

```
/Temperature                GET
/RtcData                    Observe, GET
/Leds                       subpath for red, green,blue
/Leds/xxx                   PUT, GET
updateme!                   PUT, POST, GET
removeme!                   DELETE
```

Execute the *make* command with *build*, run targets to build and run the demo server. You can test this server using the Firefox Copper plugin as shown in Figure 3.

### NodeRED and CoAP-cli support

NodeRED is a visual wiring tool for prototyping IoT solutions and networking services. It comes with an add-on *node-red-contrib-coap* and provides a *coap-request* node for maki requests to the CoAP server. This node takes input or gives output in the form of a payloa property of a JavaScript object. Currently, this node supports CON messages.

The npm package *coap-cli* provides a simple command line tool for making various reque with the following syntax:

```
coap <request> <options> URI
```

A few options are: -p for payload, -o for observe, -c for CON and -n for NON.

```
e.g.:     coap post coap://localhost/updateme! -p hello     #for POST operation
coap -o coap://localhost/RtcData #for observe         operations
```

Other implementations are:

- libcoap and freecoap in C, suitable for porting on the lwip stack
- nCoAP in Java
- npm packages coap, node-coap, coap-cli, etc, in Node.js
- CoAPthon, aiocoap, txthings, etc, in Python
- ESP8266 libraries

Stay tuned to *http://coap.technology/impls.html* for frequent updates.

### Packet sniffing using Wireshark

Wireshark comes with support for dissecting CoAP messages, and the final version of the protocol is well supported from version 1.12 onwards. For this, install Wireshark from the package manager or build it from source. Then run Wireshark with suitable privileges and

Get real-time push notifications

start capturing by selecting the interface on which the message flow exists. Type *coap* in the filter box and hit *Enter* to eliminate other packets. In each message, you can check various header elements, tokens, options and payloads, if any. For example, request the URL *coap://localhost/Temperature* using the *GET* method, and check if it is of the CON or NON type, message ID, Uri-Path option, etc. You can also find a piggy backed response with the same message ID of request, content format as *text/plain*, response code 2.05 and suitable payload. Now, select the NON type from behaviour drop-down in the Copper plugin and try the same request. You will find an independent NON message with a different ID as the response. You can dissect the message flow in Ping, discover operations and other request methods.

You can analyse capture files in the *pcapng* format recorded from Wireshark or an equivalent, using the *coap.me* pcap interpreter. It also provides a crawler client, which crawls though any public CoAP server using *coap.me.*

Get real-time push notifications

*Figure 5: Protocol debugging with Wireshark*

*Figure 6: Spitfrefox Android app*

**Android apps**

Any IoT application is not complete without a mobile app, and Android has good support for

this. A prebuilt application *Aneska* from Playstore is available. You can develop custom apps by building code available from Eclipse Californium demo apps (*cf-android*) or Spitfirefox from *github.com/okleine* using Android Studio or the Gradle build system.

Connect your mobile to the same network of the server over Wi-Fi or by using USB tethering, and test the resources by entering the server IP in these apps.

**Bridging with HTTP**

Since CoAP supports a subset of the REST model similar to HTTP, it is easy to integrate both protocols using proxy endpoints. An HTTP-CoAP proxy is used to request resources on a CoAP server from the HTTP client and reverse proxy is used to request resources on the HTTP server from CoAP clients. Eclipse Californium comes with a proxy library and an example *cf-proxy* for this.

By running *cf-proxy,* you can talk to the CoAP server from the HTTP client as follows: *http://phostname:8080/proxy/coap://chostname:5683/resource,* where *phostname* represents the proxy and *chostname* represents the origin server.

Another implementation is the npm package ponte, a current Eclipse IoT project that bridges MQTT, CoAP and HTTP protocols. To try this, install the *ponte, bunyan* packages using npm and run the following command:

```
ponte -v | bunyan
```

Using this bridge, identical URIs for the resource named abcd are *coap://hostname:5683/r/abcd* and *http://hostname:3000/resources/abcd*

Get real-time push notifications

**Public servers and cloud platforms**

Compared to other protocols, currently, very few cloud platforms support CoAP like *thethings.io,* ThingMQ, etc. A few public servers with test resources to get started with clients are listed on *coap://californium.eclipse.org/ or coap://vs0.inf.ethz.ch/, coap://coap.me/.*

I hope this article helps you to bootstrap the CoAP protocol, and its tools and libraries. Please follow the given pointers for detailed coverage of the protocol and more hands-on information.

**Share this:**

Google Facebook Twitter More

**Rajesh Sola**

The author is a faculty member at C-DAC's advanced computing training school, Pune and an evangelist in the embedded systems and IoT domains. He loves teaching and open source. You can reach him at rajeshsola@gmail.com

Get real-time push notifications