Constrained Application Protocol (CoAP) for theIoT

Method · May 2018

DOI: 10.13140/RG.2.2.33265.17766

Alabbas Alhaj Ali
Frankfurt University of Applied Sciences
8 PUBLICATIONS **7** CITATIONS

Constrained Application Protocol (CoAP) for the IoT

Alabbas Alhaj Ali, *HIS Member*

Abstract—The Internet of Things (IoT) is the network of physical devices, vehicles, home appliances, and other items embedded with electronics, software, sensors, actuators, and connectivity which enables these objects to connect and exchange data [3], [6], [7]. However, many small appliances in IoT are unable to establish a effective communication with constrained resources. That's why the need for a lightweight protocol was urgent. So Internet Engineering Task Force (IETF) has developed the Constrained Application Protocol (CoAP). This paper summarizes CoAP Protocol corresponding security protocol DTLS.

Index Terms—IoT, CoAP, Constrained Application Protocol, RFC 7252.

1 Introduction

The CoAP is an application layer protocol developed by the IETF CoRE Working Group. It is designed for constrained environments. Based on a REST style architecture, the protocol considers the various objects in the network as resources. A Unique Universal Resource Identifier (URI) is assigned to each resource. The protocol uses the corresponding URI to operate the different resources.

CoAP Protocol was designed to look like and be compatible with Hypertext Transfer Protocol (HTTP). This makes the protocol looks more like a traditional website-based business which provides the ability to be compatibility with an existing system that is web service-based [8].

In this paper we are going to examiner the application layer CoAP protocol by comparing it with HTTP, its advantages are presented. In section 2 we are going to explain in details the different models of the protocol such message layer model, request/response layer model and message format. Section 3 shortly describe the HTTP-Proxy to CoAP, resource discovery mechanism and multicasting. In section 4 we introduce the security mechanism using Datagram Transport Layer Security (DTLS) which provide the CoAP protocol with security elements like integrity, authentication, and confidentiality.

1.1 Application Layer Protocol

CoAP has been developed as a specialized web transfer protocol for use with constrained nodes and constrained (e.g., low-power, lossy) networks [2]. The design goal of CoAP has been to keep message overhead small, thus limiting the need for fragmentation. The communication in CoAP is similar to the client/server HTTP. but HTTP has high computation complexity, low data rate, and high energy consumption. while both CoAP and HTTP protocols work in the application layer or ISO model the model of CoAP looks more compatible and lightweight. Figure 1 shows CoAP stack.

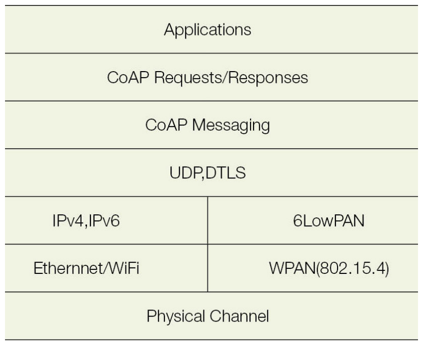

Fig. 1. CoAP Protocol Stack.

CoAP is similar to HTTP in terms of the REST model with GET, POST, PUT and DELETE methods, URIs assigned to each resource and services, MIME types, etc. However, HTTP is based on TCP protocol using point to point (p2p) where CoAP runs on UDP by default but is not limited to it, as it can be implemented over other channels like TCP, DTLS or SMS. UDP provides CoAP with IP multicast which satisfies group communication for IoT.

HTTP used under unconstrained network and CoAP used under constrained network [1]. 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.

1.1.1 CoAP Features

CoAP has the following main features [1], [2]:

- Web protocol fulfilling M2M requirements in constrained environments.
- UDP binding with optional reliability supporting uni-cast and multicast requests.
- Low header overhead and parsing complexity.
- Synchronous message exchanges.
- URI and Content-type support.
- Simple proxy and caching capabilities.

- *M. Alhaj Ali is student of master High Integrity System at Frankfurt University of Applied Sciences.*
E-mail: weeamha@gmai.com

Manuscript received May 19, 2018.

- A stateless HTTP mapping, allowing proxies to be built providing access to CoAP resources via HTTP in a uniform way or for HTTP simple interfaces to be realized alternatively over CoAP.
- Security binding to Datagram Transport Layer Security (DTLS).

2 CoAP Structure

On design, the CoAP protocol structure the most important key was to avoid message fragmentation so that the CoAP package could be fit in one single frame at the Ethernet or IEEE 802.15.4 layer. As is shown in Figure 1 the CoAP Message Layer is designed to deal with UDP and asynchronous switching and request/response layer handle the communication method.

2.1 Message Layer

The CoAP messaging model is based on the exchange of messages over UDP between endpoints [2].

Message Layer supports 4 types message: CON (confirmable), NON (non-confirmable), ACK (Acknowledgement), RST (Reset) [2]. The protocol provides reliability throw the message as Confirmable (CON) this type of message usually re-transmitted be a default timeout and decreasing counting time exponentially until the recipient sends an Acknowledgement message (ACK) with the same Message-ID [1], [2], [8]. Figure 2 shows a reliable message transport.

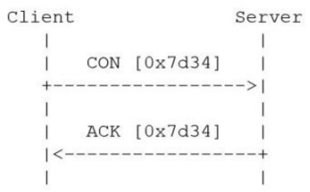

Fig. 2. Reliable message transmission.

The unreliability of message transport is carried throw NON-type message which does not require acknowledgment, but this message contains ID for supervising in case of re-transmission.

2.2 Message Format

Figure 3 shows CoAP message format which contains a fixed-length 4-byte header that may be followed by compact binary options and a payload. This message format is shared by requests and responses.

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

Ver	T	OC	Code	MessageID
Token (if any, TKL bytes)...				
Options (if any)...				
Payload (if any)...				

Fig. 3. CoAP message format [2].

The message header bits are defined as follows:

- (Ver): 2-bit unsigned integer. Indicates the CoAP version number.
- (T): 2-bit unsigned integer. Indicates if this message is of type Confirmable (0), Non-confirmable (1), Acknowledgement (2), or Reset (3).
- Code: 8-bit unsigned integer, the Request message (1-10) or Response message (40-255).
- Message-ID: 16-bit unsigned integer in network byte order, Identifier for matching messages responses.
- Token (TKL): 4-bit unsigned integer. Indicates the length of the variable-length Token field (0-8 bytes).
- Options: Zero or more optional fields may follow a token. A few options like Content Format, Accept, Max-Age, Etag, Uri-Path, Uri-Query, etc

2.3 Request/Response Layer

The request/response could happen in two ways Piggy-backed or Separate response. Figure 4 shows the Piggy-backed where the client sends the request using CON type or NON-type message and receives response ACK message immediately.

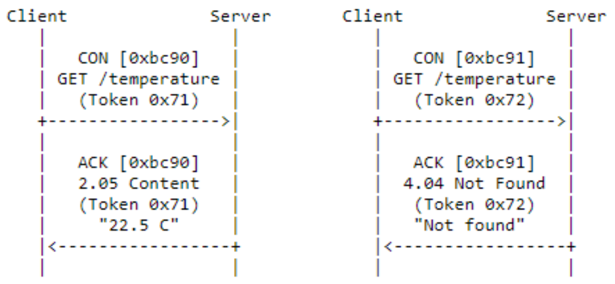

Fig. 4. Piggy-backed request/response transmission [2].

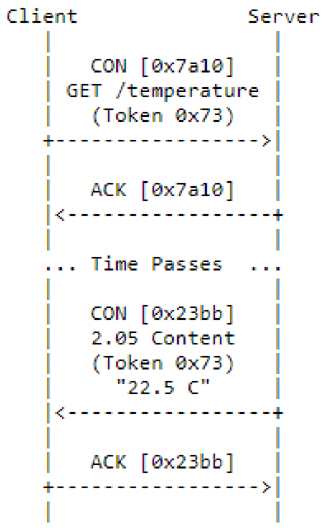

Fig. 5. Separate request/response transmission [2].

In the other hand with Separate response message type shown in Figure 5. when the receiver receives a CON type message but not able to respond to this request immediately, it will send an empty ACK message immediately. When the response is ready, it will send a new CON to client and client reply a confirmable message with acknowledgment.

2.3.1 *URI scheme*

CoAP has its own URI scheme, which is very similar to HTTP scheme but differs in a few details. Figure 6 shown CoAP URIs scheme.

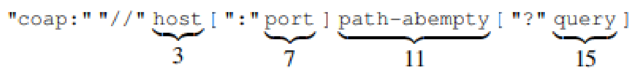

Fig. 6. CoAP URIs scheme [4].

We can see that the scheme for both protocols is very similar. However, the significant difference between two protocols is the decomposition of URI into individual components and their integration in the header [4]. HTTP always stores the absolute path and even includes the queries for some request methods (eg GET or HEAD), whereas CoAP always saves all path pieces and queries separately in individual options. The same applies to host addresses and port numbers, They are also stored separately from each other, unlike HTTP. The respective URI always fit into the initial 4-bit field when assembling the header.

2.3.2 *Request methods*

CoAP resembles HTTP with request methods of GET, POST, PUT, and DELETE. They have the same properties of safe (only retrieval) and idempotent (you can invoke it multiple times with the same effects) as HTTP [2], [4], [5]. These methods can be used to create, update, query and delete the resources on the server representing events of an IoT application.

In additional CoAP define a new method the Observe method. This method is used to implement a subscription concept build-in with the protocol. This method is not in HTTP protocol and has been constructed over HTTP in many ways. Observe method is simply a GET request with the option called observe which till the server that this client want to get every update in this resource. The observe is define be Taken and every time a change happens to the resource the server sends a new response to the client with the same Taken [4], [5].

2.3.3 *Response codes*

There are three ways to send responses: Piggy-backed, Separate and Non-confirmable. A piggy-backed response is a response that is delivered with an acknowledgment message. Sometimes it is not possible (eg due to long processing time) to send the answer immediately, so first an empty acknowledgment and later the answer is sent in a separate confirmable message. Finally, if the request was received with a non-confirmable message, the response must also be sent as non-confirmable [2], [4], [7].

Response codes are similar to HTTP. For instance, 2.xx indicates success, 4.xx indicates client error and 5.xx indicates server error. Although some response codes match the HTTP status codes (eg, 4.04 and 404 ”Not Found”), others have different codes (2.05 ”Content” is equivalent to 200 ”OK”, but 2.05 is only used in response to GET) or are not represented at all (see Table 1).

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

TABLE 1
CoAP Response codes [2], [4]

3 Cross-Protocol and Resources

In addition to the two-layered interaction model, CoAP has additional features such as HTTP Proxying, Multicasting and Service and Resource Discovery.

3.1 HTTP Proxy

The similarity between CoAP and HTTP protocol offer a convenient way to implement proxy between this two protocols. That is how to devices implements the two protocols can easily connected throw the Proxy. There are two types of cross-protocol proxying:

CoAP-HTTP Proxying: allows access to the resources of an HTTP server for CoAP clients. Since CoAP methods are equivalent to HTTP methods, the construction of HTTP, TCP and optionally TLS can be done easily [4].

HTTP-CoAP Proxying: allows access to the resources of a CoAP server for HTTP clients. To send an HTTP request to the proxy, the client must specify the absolute path to the resource including the schema (coap / coaps) in the method invocation. Once the proxy has received the message, it will request the specified CoAP resource. All HTTP methods from specification RFC 2616, with the exception of OPTIONS, TRACE and CONNECT, can also be converted into CoAP [2], [4].

3.2 Resource Discovery

In CoAP, a server can be discovered through (knowing or) learning a URI that references a resource in the namespace of the server. Alternatively, client can send a CoAP multicast request to all nodes using the ”All CoAP Nodes” address [4].

The CoAP default port number 5683 MUST be supported by a server that offers resources for resource discovery. The discovery of resources offered by a CoAP endpoint is extremely important in machine-to-machine applications where there are no humans in the loop and static interfaces result in fragility [2], [9].

4 Security in CoAP

There are three main elements when considering security, namely integrity, authentication, and confidentiality.

HTTP is secured using Transport Layer Security (TLS) over TCP [2], [9]. Since TLS relies on reliable transport-layer transmission, TLS cannot be readily used with CoAP. First, the integrity checking of data is sequential, That is if a particular message is not received, the integrity check of the next message is based on the wrong sequence number and will therefore fail. On the other hand, the TLS handshake (authentication and key exchange) cannot take place at all if the messages are not exchanged reliably, that is, in a disorderly or lossy manner [4], [9].

CoAP is secured using Datagram TLS (DTLS) over UDP [2]. DTLS in application layer protect end-to-end communication. DTLS also avoids cryptography overhead problems that occur in lower layer security protocols. DTLS solves two problems reordering and packet loss. DTLS is TLS with added features to deal with the unreliable nature of the UDP transport. Figure 7 shows the abstract layering of DTLS-Secured CoAP.

```
+----------------------+
|      Application     |
+----------------------+
+----------------------+
|  Requests/Responses  |
|----------------------|  CoAP
|       Messages       |
+----------------------+
+----------------------+
|         DTLS         |
+----------------------+
+----------------------+
|          UDP         |
+----------------------+
```

Fig. 7. Abstract Layering of DTLS-Secured CoAP [2].

Unlike TLS, the DTLS messages are explicitly numbered. TLS relies on the correct order of messages backed up only by TCP. The explicit numbering of messages now also allows the securing of the handshake: DTLS transmits packets again (re-transmission) if no response has been received after a certain time (timeout). Furthermore, DTLS records can be both fragmented and reduplicated.

CoAP device is provided with the security information that it needs, including keying materials and access control lists [2]. CoAP defines four security modes in which a CoAP device operates, with NoSec and RawPublicKey mandatory to implement:

NoSec: DTLS is not used. Alternatively, CoAP can be used with IPsec [2], [4].

PreSharedKey (PSK): DTLS is enabled, Symmetric keys are distributed to the nodes in advance. and each key includes a list of which nodes it can be used to communicate with as described [2]

RawPublicKey (RPK): DTLS is enabled, An asymmetric key pair is used, but without a certificate. The public keys of a device are stored, for example, in the firmware ”raw” (without X.509v3 structure) and can be updated. The device also has an identity calculated from the public key and a list of identities of the nodes it can communicate with [2], [4].

Certificate: DTLS is enabled, An asymmetric key pair with X.509 certificates of an X.509v3 is used. Certificate issued by a certification authority (CA). The device also has a list of root trust anchors that can be used for validating a certificate [2], [4].

Similar to HTTPs, CoAP defines the CoAPs URI schema. This differs from CoAP only in the scheme specification. CoAP and CoAPs are to be regarded as stand-alone servers, even if the host specification of URIs matches. The default port for CoAPs is not yet defined [2].

5 Conclusion

This work gave an overview of the new application protocol CoAP. CoAP offers the benefits of a REST architecture designed specifically for low-power loss wireless networks.

The IETF CoRE Working Group is developing the protocol to allow resource access similar to HTTP for small devices in M2M communication. Such devices become part of the Internet of Things.

CoAP header including options is composed: Instead of coding the fields as strings, they are stored in binary format, ie as sparingly as possible. CoAP's two-layered interaction model was presented: Transmission reliability is secured on the message layer, and the functionality of the REST architecture is implemented on the request-response layer.

CoAP messages can be made tap-proof using encryption protocol DTLS. The slim structure in relation to header fields of a CoAP message has a convincing effect. From the functionality of CoAP, M2M is clearly visible as the main guideline. The real advantages and disadvantages in relation to other application protocols such as HTTP will only become clear in the future with a further dissemination and implementation of CoAP.

Acknowledgments

Technology has made it easier for the students to learn with new devices, but nothing can come close to the experience of being touched by a great people. Thank you Dr.-Ing. Elpiniki Tsakalaki.

References

[1] Prof. Raj Jain. *Constrained Application Protocol for Internet of Things* Xi Chen, chen857 (at) wustl.edu.

[2] Z. Shelby, K. Hartke, C. Bormann, *Constrained Application Protocol (CoAP)*, Universitaet Bremen TZI. [June 2014]

[3] Brown, Eric (13 September 2016). *Who Needs the Internet of Things?*. Linux.com. Retrieved 23 October 2016.

[4] Roman Trapickin (2013-08-1), *Constrained Application Protocol (CoAP): Einfhrung und berblick*, Fakultt fr Informatik, Technische Universitt Mnchen.

[5] Z. Shelby, *Constrained Application Protocol (CoAP) Tutorial* youtube video, https://www.youtube.com/watch?v=4bSr5x5gKvA

[6] Brown, Eric (20 September 2016). *21 Open Source Projects for IoT*. Linux.com. Retrieved 23 October 2016.

[7] *Internet of Things Global Standards Initiative*. ITU. Retrieved 26 June 2015.

[8] Jonathan Fries (04 May 2017). *Why are IoT developers confused by MQTT and CoAP?* techtarget.com. Retrieved 28 March 2017.

[9] Mohammed Riyadh Abdmeziem, Djamel Tandjaoui, and Imed Romdhani, *Architect the Internet of Things: State of the Art*. university of Sciences and Technology Houari Boumedienne.