IoT Messaging Protocols Analysis

Dragos-Alexandru Andrioaia
Computer Department
Stefan cel Mare University of Suceava
Suceava, Romania
dragos.andrioaia@ub.ro

Vasile-Gheorghita Gaitan
Computer Department
Stefan cel Mare University of Suceava
Suceava, Romania
gaitan@eed.usv.ro

***Abstract*— In recent years, Internet of Things has become a widespread concept. The concept of IoT will increasingly integrate into various fields. The technological advancement in IoT will depend on the evolution of the emerging technologies. In this paper, the authors perform an analysis of the main protocols used at the application level that are frequently encountered in IoT applications. Based on the most recent studies in the literature of specialty, four messaging protocols (MQTT, CoAP, XMPP and AMQP) were presented. At the end of the work, a comparative analysis of messaging protocols in terms of key indicators of performance, such as latency, power consumption, bandwidth and security, is presented. This study allows users to choose the right messaging protocol according to technical design requirements.**

Keywords*— *Internet of Things; IoT; MQTT; CoAP; XMPP; AMQP.

I. INTRODUCTION

The interconnection of all" things" that surround us in our daily lives through sensors and cloud computing will open the door to the next industrial revolution. IoT has the potential to change the world we live in if implemented successfully, but nonetheless, there are various problems that require thorough research to improve the quality of life [1].

Through IoT objects achieve intelligent behavior through related decisions resulting from the fact that they can communicate with each other and obtain information about each other.

From the drawn-up estimates, there is no exact knowledge of the impact that IoT will have on the population, which can be both an opportunity and a danger.

According to the Internet Society, by 2025 there will be about 100 billion IoT devices connected to the Internet and the market value will rise to over 11 trillion dollars [1].

IoT as well as the protocols involved are among the most funded topics in both the industrial and academic fields. The application layer protocols, also known as messaging protocols, are located on the last layer of the TCP/IP stack model that corresponds to the OSI session layer.

These protocols contain a series of rules that allow IoT devices to communicate with each other. Protocols define syntax, semantics and synchronization of communication [2].

In this paper, the authors present the main messaging protocols used at the application level, then conduct a comparative study in terms of performance indicators.

The main purpose of this individual analysis then comparative is to know the functionality of each messaging protocol as well as the values of performance indicators in order to choose the most appropriate protocol according to the technical design requirements.

The comparative messaging protocols use two data transmission models, request/response and publish/subscribe.

II. IOT MESSAGING PPROTOCOLS

Over the past few years, several messaging protocols that can be used in the IoT field have been proposed by researchers [3]. Of these, only five of them have a common utility in IoT applications.

The five application-wide protocols are: Message Queuing Telemetry Transport Protocol (MQTT), Constrained Application Protocol (CoAP), Advanced Message Queuing Protocol (AMQP), Xtensible Messaging and Presence Protocol (XMPP), and Hypertext Transfer Protocol (HTTP).

As the HTTP protocol is familiar in the following section, the protocols MQTT, CoAP, AMQP and XMPP will be presented.

A. Message Queuing Telemetry Transport Protocol (MQTT)

MQTT is an application-level communication protocol based on TCP/IP transfer, invented by Dr. Andy Standford-Clark and Arlen Nipper (IBM) in 1999 [4].

The standardization of this protocol was done under the supervision of Organization for the Advancement of Structured Information Standards (OASIS) in 2014. The IoT underlying principles were the minimization of network traffic and the possibility to run on devices with reduced processing capacity [5], [6].

978-1-7281-6870-8/20/$31.00 ©2020 IEEE

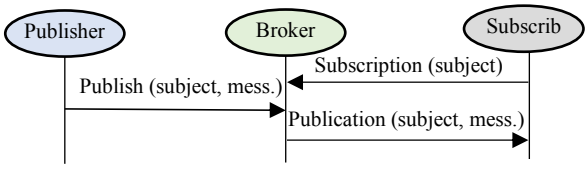

Fig. 1. The publication-subscription model of MQTT protocol.

The MQTT was designed to transmit a string of ordered bytes without errors from client to sever and vice versa. The model of dialogue used is of publish-subscribe (publish-subscribe) type, unlike the protocol HTTP, in which the exchange of data between client and server is done by request and response (request/response).

MQTT is based on a publisher-subscriber model, in which the publisher (a client) publishes data about a particular topic on a server (also called a broker), and the subscriber/subscribers (another client/other clients) subscribe to the server and receives/receive data from the server on the subject of interest (Fig. 1) [6], [7].

The MQTT broker is responsible for distributing messages and can be positioned in the cloud. The broker performs the filtering of the messages, he decides what message he will send to a certain subscriber according to the subject, content type and access permission [4].

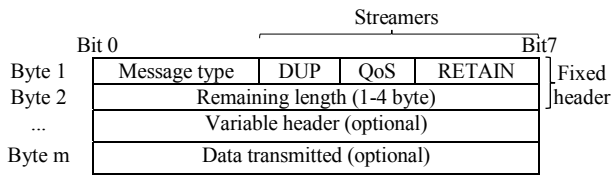

Fig. 2. MQTT message format.

The MQTT message packages as can be seen in Fig. 2, contain three parts: a mandatory header with fixed length of 2 bytes (present in all packages), a variable header (present only in certain packages) and transmitted data (only in some cases). The first byte in the fixed header identifies the type of control message (CONNECT -1, PUBLISH – 3, etc.), as well as a series of control bits, specific for each message. The second byte indicates the number of bytes left in the message, i.e. the length of variable header and the transmitted data [4], [5].

Of all 14 types of message packages available, only 5 of them may carry information: CONNECT, PUBLISH, SUBSCRIBE, UNSUBSCRIBE and SUBACK.

A protocol derived from MQTT is MQTT-S protocol. The latter has been designed for IoT sensor networks and uses the UDP protocol at the transport level [5].

Therefore, the MQTT protocol is an ideal messaging protocol for IoT communications.

B. Constrained Application Protocol (CoAP)

It was proposed by the CoRE working group of the Internet Engineering Task Force (IEFT), being standardized in 2014 [5]. It is an application-level software protocol designed for IoT devices that have limited memory space and computing power, but also low power consumption [3].

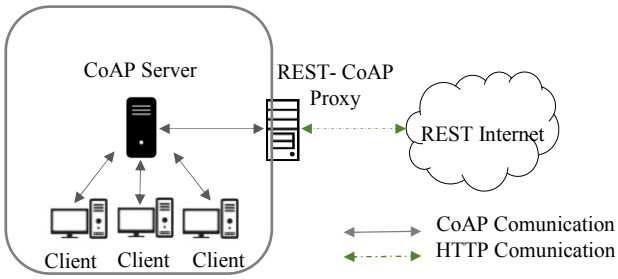

Fig. 3 The means of communication in CoAP [5].

CoAP is a protocol that transfers information between client and server and integrates existing web technologies REST and HTTP (Fig. 3). It is based on request and response messages similar to HTTP protocol, but the standard recommends the use of UDP on the transport level [5], [8].

The COAP protocol contains a mechanism for verifying the order in which the packages arrived as well as for handling network errors implemented through messages. It is used by social media platforms and mobile networks [4].

CoAP can be structured on two layers: the messaging substrate - detects duplications and provides packet loss communication on the UDP transport layer and request/response sublayer - it handles REST communications [8], [9].

Messages are transmitted by request and responses. Each message contains a message ID used to detect duplicate messages. CoAP uses four types of messages: message that needs confirmation – it requires a confirmation response from the receiver to avoid package loss (secure messages Fig. 4.a); message that does not require confirmation – the data is transmitted without waiting for a confirmation message from the receiver (unsafe messages Fig. 4.b); confirmation (acknowledgment) – recognizes the sent message that needs confirmation; Reset – reset is used instead of confirmation if sent messages that require confirmation or not cannot be processed.

It also uses four response modes: confirmable, non-confirmable, piggy packed responses and separate response [4], [8].

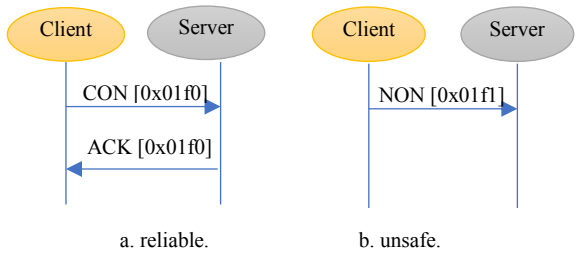

Fig. 4. Message transmission [3]:

If, as a result in transmitting a message that requires confirmation, there is no received confirmation, then the message will be retransmitted. The message sent back for confirmation contains the message ID that requires confirmation [4].

CoAP, as well as HTTP uses methods such as: GET, PUT, POST și DELETE [9], [10].

CoAP messages (Fig. 5) are binary encoded and use a fixed 4-bytes header. The message header contains a series of information regarding: protocol version (VER); type of message transmitted (T); the number of bits of the token field (TKL); request code (1-10) or response code (40-255) (Cod); as well as the message number used to verify that the message is repeated or that it was received correctly (message ID) [4].

	Bit 0		Bit 8	Bit 16	Bit 31
Byte 1	VER	T	TKL	Cod	Message ID
Byte 2	Request identifier (token)				
…	Options				
Byte m	Data transmitted (loading)				

Fig. 5 Messages CoAP format [4], [5].

Request identifier (token) allows the distinction between request and response. The options and the loading are optional fields. A typical CoAP message can have between 10 and 20 bytes. [4].

C. Xtensible Messaging and Presence Protocol (XMPP)

XMPP originally known as Jabber, was developed in 1999 and standardized in 2004 by the IETF. The protocol is open-source and its extensions are currently being developed. Originally designed for instant messaging services, it was later used in IoT applications [6].

XMPP is based on the XML standard and uses the TCP protocol transport layer. Accepts both the publication/subscription architecture specific to the MQTT protocol, as well as the request/response. [10].

It has been designed for almost real-time communications and therefore message latency is reduced. XMPP is a client-server protocol [6].

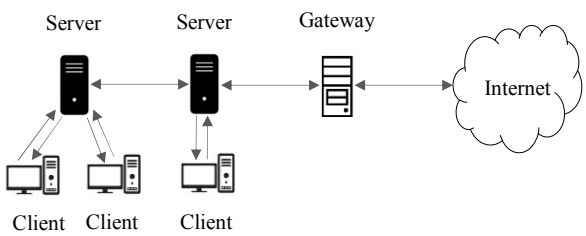

Fig. 6. Means of communication in XMPP [6].

Data transfer from one client to another is done through servers and also for foreign domains through the gateway (Fig. 6.) [6].

XMPP clients communicate with each other through XMPP addresses that have a format similar to e-mail addresses. Each client has a personal id called Jabber ID (JID), the *user@domain* form.

Once the connection is established, message exchanges are made through two XML streams, the first is transmitted from client to server and the second from server to client. This transmission from client to server and from server to client is called stanza. These stanzas XML can be divided in three classes in XMPP: message, presence and iq (information/query) (Fig. 7) [8].

```
<stream>
    <presence>
      <show/>
    </presence>
    <message to=_ '>
      <body/>
    </message>
    <iq to=_ '>
      <query/>
    </iq>
</stream>
```

Fig. 7. XMPP stanza structure [8].

Stanzas XML:

<message/> - can be regarded as a „store and push" mechanism in which an entity publishes information to another, such as transferring messages between two terminals. Each message is assigned an id and the sender and receiver of the message have been defined as attributes inside the tag <message/> [8], [11].

<presence /> - can be viewed as a broadcast or publish-subscribe mechanism, whereby multiple entities receive information about a particular entity, for example: availability of the entity (online, offline) or another relevant feature [8], [11].

<iq/> - is a request-response mechanism between any two entities similar to HTTP that allows entities to formulate request and receive responses between each other [8], [11].

All stanzas primitives must be in the <stream /> block, which represents an XML stream.

D. Advanced Message Queuing Protocol (AMQP)

John O'Hara, an employee of JPMorgan Chase bank, designed AMQP in 2003. A binary, open-source protocol, it uses the TCP protocol on the transport layer. AMQP guarantees that it has three levels of delivery: at-most-once, at-least-once and exactly once delivery [12], [13].

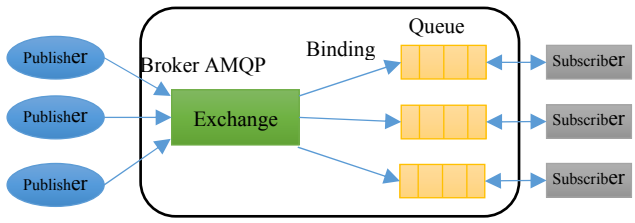

Fig. 8. The editor subscriber model in AMQP [3], [5], [12].

AMQP also supports the point-to-point communication model as well as the publish/subscribe communication model.

In the AMQP model, customers send and receive messages through a message broker. Functionally, an AMQP broker has three components: exchange, queue and binding Fig. 8 [12].

Exchange, receive messages sent from the publisher application and direct them to "message queues" [5], [12].

Message queue stores messages until they can be safely processed by the subscriber's client application [5], [12].

The link establishes the relationship between the message queue and the exchange. The routing between exchange and message queues is based on certain predefined rules and conditions [5], [12].

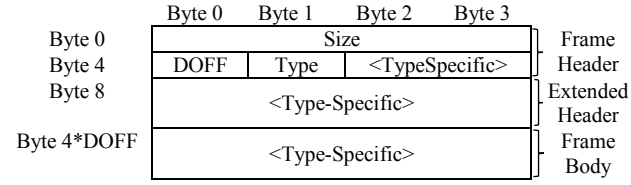

Fig. 9. AMQP frame format [12].

The transport layer provides the extension points required for the messaging layer. In this layer, communications are frame-oriented. An AMQP frame can be divided into three parts: the frame header, the extended header, and the frame body. The overall structure of the AMQP frames is shown in Fig. 9. The top four bytes show the size of the frame. Date Offset (DOFF) gives the body position inside the frame. The Type field indicates the format and purpose of the frame.

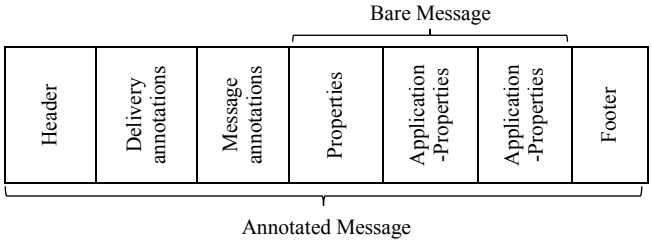

Fig. 10. The AMQP messages format [5].

On the messaging layer, the AMQP standard defines two types of messages: massages sent by the sender and messages received by the receiver. Fig. 10 is presented the format of the AMQP message. The message received by the receiver contains the message sent by the sender but also contains other information generated by the messaging infrastructure [5].

The header transmits delivery parameters, including durability, priority, lifetime, first link, and number of previous attempts used to deliver the message. The Message/Delivery Annotations section sends messages/delivery information that is not included in the Properties section. The Properties section contains information such as: ID, source, destination, subject, content type, expiration time, time to create this message, etc. Application properties structure the Data Applications section. Data in the Data Applications section is used for filtering or routing. Footer provides hash messages, signatures, and encryption details [5].

III. Comparative Analysis of Messaging Protocols Used in IoT

This chapter contains a comparison of the main messaging protocols (MQTT, CoAP, XMPP, AMQP) commonly used in IoT applications, Tab. I. MQTT, has the message header of 2 bytes. This makes it ideal for networks whose bandwidth is limited. The API requires minimal processing on devices, so power consumption is low. MQTT broker cannot store multiple messages in a message queue, only the last message received in the message broker [14].

AMQP is used in IoT applications that require the transfer of a large amount of data. The power consumption, processing power and memory size requirements for a device using the AMQP messaging protocol are relatively high.

COAP uses the UDP protocol at the transport level. This causes the transfer speed between devices to be high, but it only makes it used within a subnet (global networks use TCP protocol transport). Due to the fact that COAP has a condensed header and the size of the packages is small, the COAP messaging protocol is more efficient than HTTP [1].

XMPP is a protocol with a decentralized architecture similar with the e-mail protocols such as SMTP, which can be implemented on a large scale.

TABLE I. COMPARATIVE ANALYSIS OF MESSAGING PROTOCOLS USED IN IOT

No.	Criterion/ Protocol	MQTT	CoAP	XMPP	AMQP
1.	Year	1999	2010	2004	2003
2.	Publish/ Subscribe Model	Yes	Yes	Yes	Yes
3.	Request/ Response Model	No	Yes	Yes	No
4.	Transport Protocol	TCP	UDP	TCP	TCP
5.	Encoding Format	Binary	Binary	Character	Binary
6.	Header Size (bytes)	2	4	-	8
7.	Bandwidth needed	Low	Low	Low	High
8.	Proxy Support	Partial	Yes	Yes	Yes
9.	Security	TLS/SSL	DTLS, IPSec	SASL and TLS	TLS/SSL, IPSec, SASL
10.	Support to QoS	Yes	Yes	No	Yes
11.	Platform IoT	Microsoft Azure IoT Suite, IBM Watson IoT, ThingWorx, Amazon Web Services IoT.	ThingWorx	ThingWorx	Microsoft Azure IoT Suite

Choosing a messaging protocol must be based on several factors. The most important factors are computing capacity, communication capacity, energy consumption and bandwidth.

IV. CONCLUSIONS

IoT is the field that allows "things" to communicate with each other over the Internet, store and recover data, and interact with users. It will continue to change our lives as emerging technologies continue to evolve.

The main messaging protocols used by IoT systems were analyzed and compared in this paper. Each protocol has a number of advantages compared to the others. The choice of the appropriate messaging protocol is based on the type of application served and the advantages we follow. A more complex application could use more messaging protocols on the application layer according to the requirements of each module.

MQTT is the most widely used messaging protocol within IoT applications due to low bandwidth and energy consumption.

COAP can be used in applications which require REST functionality being much more efficient than the HTTP protocol.

XMPP can send messages in real time to a large number of clients.

AMQP has a high security.

The comparison helps us identify which messaging protocol is best suited for many IoT applications.

ACKNOWLEDGMENT

The authors would like to acknowledge “Stefan Cel Mare” University of Suceava, for the support.

REFERENCES

[1] A. U. R. Khan, q. f. Hassan, and S. Madani, Internet of Things: Challenges, Advances, and Applications, 1 ed. New York: Chapman and Hall/CRC, 2018, p. 436.

[2] X. Perry, Designing Embedded Systems and the Internet of Things (Iot) with the Arm Mbed. New York: John Wiley & Sons Inc, 2018, p. 344.

[3] T. Salman and R. Jain, "Networking protocols and standards for internet of things," in Internet of Things and Data Analytics Handbook, H. Geng Ed.: John Wiley & Sons, 2017, pp. 215-238.

[4] D. B. Ansari, A.-U. Rehman, and R. Ali, "Internet of Things (IoT) Protocols: A Brief Exploration of MQTT and CoAP," International Journal of Computer Applications, vol. 179, no. 27, pp. 9-14, Mar. 2018, doi: 10.5120/ijca2018916438.

[5] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," IEEE Communications Surveys & Tutorials, vol. 17, no. 4, pp. 2347-2376, Jun. 2015 2015, doi: 10.1109/COMST.2015.2444095.

[6] S. Jaikar and R. Kamatchi, "A Survey of Messaging Protocols for IoT Systems," International Journal of Advanced in Management, Technology and Engineering Sciences, vol. 8, no. 2, pp. 510-514, Feb. 2018.

[7] C. Akasiadis, V. Pitsilis, and C. D. Spyropoulos, "A Multi-Protocol IoT Platform Based on Open-Source Frameworks," (in eng), Sensors (Basel), vol. 19, no. 19, p. 4217, Sep. 2019, doi: 10.3390/s19194217.

[8] M. S. Anusha, Babu, E; Sai Mahesh, Reddy, L; Vamsi, Krishna, A; Bhagyasree, B, "Performance Analysis of Data Protocols of Internet of Things: A Qualitative Review," International Journal of Pure and Applied Mathematics, vol. 115 no. 6, pp. 37-47, 2017.

[9] L. Nastase, "Security in the Internet of Things: A Survey on Application Layer Protocols," in 2017 21st International Conference on Control Systems and Computer Science (CSCS), May 2017, pp. 659-666, doi: 10.1109/CSCS.2017.101.

[10] V. Karagiannis, P. Chatzimisios, F. Vázquez-Gallego, and J. Alonso-Zarate, "A survey on application layer protocols for the Internet of Things," Trans. IoT Cloud Comput., vol. 3, no. 1, pp. 11-17, Jan. 2015, doi: 10.5281/zenodo.51613.

[11] H. Arslan, "Extensible messaging and presence protocol’s adaptation to business applications," Global Journal of Computer Sciences: Theory and Research, vol. 6, no. 1, pp. 10-17, Nov. 2016, doi: 10.18844/gjcs.v6i1.1210.

[12] P. Bhimani and G. Panchal, "Message Delivery Guarantee and Status Update of Clients Based on IoT-AMQP," in Intelligent Communication and Computational Technologies, Singapore, Y.-C. Hu, S. Tiwari, K. K. Mishra, and M. C. Trivedi, Eds., 2018, vol. 19: Springer, pp. 15-22, doi: 10.1007/978-981-10-5523-2_2.

[13] S. Vinoski, "Advanced Message Queuing Protocol," Internet Computing, IEEE, vol. 10, no. 6, pp. 87-89, Nov. 2006, doi: 10.1109/MIC.2006.116.

[14] A. Talaminos-Barroso, M. A. Estudillo-Valderrama, L. M. Roa, J. Reina-Tosina, and F. Ortega-Ruiz, "A Machine-to-Machine protocol benchmark for eHealth applications – Use case: Respiratory rehabilitation," Computer Methods and Programs in Biomedicine, vol. 129, pp. 1-11, Jun. 2016, doi: https://doi.org/10.1016/j.cmpb.2016.03.004.