Internet of Things (IoT) Protocols: A Brief Exploration of MQTT and CoAP

Danish Bilal Ansari
Department of Computer Science and Information Technology
Virtual University of Pakistan

Atteeq-Ur-Rehman
Department of Computer Science and Information Technology
Virtual University of Pakistan

Rizwan Ali Mughal
Department of Computer Sciences
Comsats Institute of Information Technology

ABSTRACT
The concept of the Internet of Things emerged a long time ago, having enormous development in sensing devices and every-day-objects connected to the internet. With current internet infrastructure, wireless communication plays a vital role in IoT devices allowing them to transmit messages. Therefore, the vitality of these messages lies in authentication. Numerous key management techniques have also been introduced to provide a secured transmission over the internet. In the context of IoT, many protocols have been devised for authenticated and secured transmission, including XMPP, AMQP, and LWM2M. Addition to above, MQTT and CoAP are also extensively used protocols in most M2M communication. This survey paper is an exploration of these protocols and also exemplify the comparison between them.

Keywords
Internet of Things, IOT, MQTT, COAP, Messaging Format of MQTT and CoAP, Security in MQTT and CoAP

1. INTRODUCTION
The term Internet of Things (IoT) was coined in 1999 by Kevin Ashton through Auto-ID Center at MIT and market-related publications [1]. He presented the idea of tagging everyday objects with identifiers. Although this can be done by QR or Barcodes etc. but with the revolution in technology, the term IoT becomes more enriched with sensors and actuators allowing interaction with internal or external states. Physical devices like cars, watches, air-conditioners, and clothes etc. will gather data and share it with other devices so that appropriate measurements can be taken.

With the huge growth in IoT, Gartner [2] predicts that by 2017 8.4 billion IoT devices will be in use, with a 31 percent increase from 2016, and by 2020 it will reach 20.4 billion, while the spending on these services will reach about $2 trillion by 2017.

IoT devices range from large to small scale where most devices are network-depended while restricted in terms of power consumption and resources. So, development of such devices must be cost-effective and proficient to share data. Connectivity in IoT data comprises of wide range of protocols when developing IoT applications. These applications attain huge amount of data from various devices. Application nature is heavily dependent on the selection of Protocol Different protocols including AMQP, XMPP, and LWM2M exists for home automation, vehicle-to-vehicle communication, and wearable devices, with two expertized protocols for Machine-to-Machine (M2M) communication includes Message Queue Telemetry Transport (MQTT) and Constrained Application Protocol (CoAP).

The paper is arranged as follows. Section 2 describes the architecture of MQTT and CoAP. Section 3 focuses on the message format of MQTT and CoAP. Section 4 addresses security in MQTT and CoAP, and finally, Section 5 concludes the paper.

2. ARCHITECTURE OF MQTT AND COAP
As described earlier, various other protocols exist while MQTT and CoAP are extensively used in M2M communication. These protocols are described as follows:

2.1 Message Queue Telemetry Transport (MQTT)
Message Queue Telemetry Transport (MQTT) was developed by IBM in 1999 [3], which is a publish/subscribe protocol. It runs over TCP/IP. The client acting as publisher/subscriber connects to a server which is a message broker for receiving notifications. MQTT is used for remote locations where limited network bandwidth is required [4]. The architecture of MQTT is shown in Figure 1.

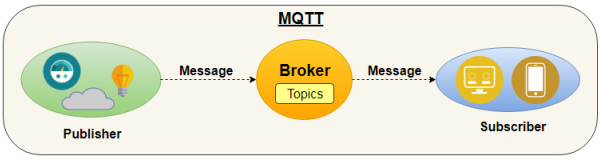

Fig 1: MQTT Architecture

Different clients can subscribe to a central broker for accumulating topics created by the devices in MQTT which employs a message exchange model. This message exchange model is resource proficient and does not follow a specific data structure. A client receives a message from the broker, which it subscribed to, about a topic when a message is posted by the device. The purpose of the broker is to confirm message delivery by simplifying management and facilitating IoT devices connected through the network. MQTT is well suited for machine-to-machine communication following a lightweight messaging protocol [5]. It also provides security even if the connection breaks off for the transmitted messages by resolving glitches with untrustworthy connections.

2.2 Constrained Application Protocol (CoAP)

Constrained Application Protocol (CoAP) was introduced for lightweight RESTful interfacing by IETF Constrained RESTful Environment working group known as CORE [7]. REST architecture is used as a communication between HTTP client and server because of its lightweightness and is easier to consume [6], but for lightweight IoT applications, it could result in excessive power utilization and overhead. CoAP is best suited for low-energy consumption sensors to utilize RESTful services within their power limitations.

CoAP is a protocol that allows communication with wider internet using similar protocols for constrained IoT devices, which are termed as "nodes". It is best suited for devices that are on same or different constrained network. CoAP can be considered as a substitute to HTTP as it is built over UDP rather than TCP which is a common practice in HTTP. CoAP uses the Efficient XML Interchanges (EXI) format which is space effective as compared with XML/HTML [8] which is a binary format. The architecture of CoAP is shown in Figure 2.

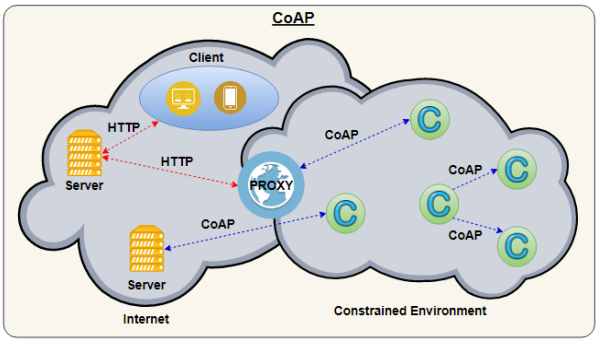

Fig 2: CoAP Architecture

CoAP comprises of two layers named as messaging and request/response. For redundancy and consistency of any message, the messaging layer is responsible while connectivity and communication handling is request/response layer job. CoAP also facilitates multicast messaging as well as support asynchronous exchange of messages. These layers are interpreted as follows:

2.2.1 Message Layer

Four type of messages are supported by CoAP as follows [9]:

- Confirmable (CON)
- Non-Confirmable (NON)
- Acknowledgment (ACK)
- Reset (RST)

2.2.1.1 Reliable Message

The reliable messaging mode is considered when a message is marked as confirmable. A confirmable message is sent out by a default timeout and exponential back-off, until the recipient reply with the acknowledgment message [9]. The received acknowledgment message contains the same id. The reset message will replace acknowledgment if a reply is unsuccessful by the recipient. The reliable messaging mode is represented in Figure 3.

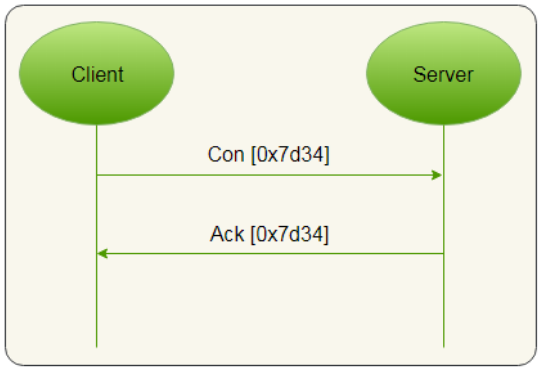

Fig 3: Reliable Message

2.2.1.2 Unreliable Message

Any message not involving as reliable will be considered as a non-confirmable message. These messages also contain a message id for redundancy identification. However, they are not an acknowledgment. They also send reset message when the recipient is unable to respond a non-confirmable message. The unreliable messaging mode is represented in Figure 4.

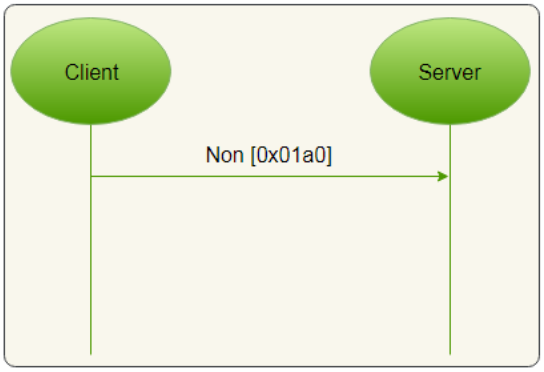

Fig 4: Unreliable Message

2.2.2 Request/Response Layer

Messages in request/response layer can be described as follows:

- Piggy-Backed
- Separate
- Non-Confirmable

2.2.2.1 Piggy-Backed Message

When any response is sent immediately by the server after receiving a confirmable or non-confirmable message, then the piggyback mechanism of CoAP is employed. Generally, the message is termed as an acknowledge message. A successful and failure message resulting in acknowledgment response is shown in Figure 5.

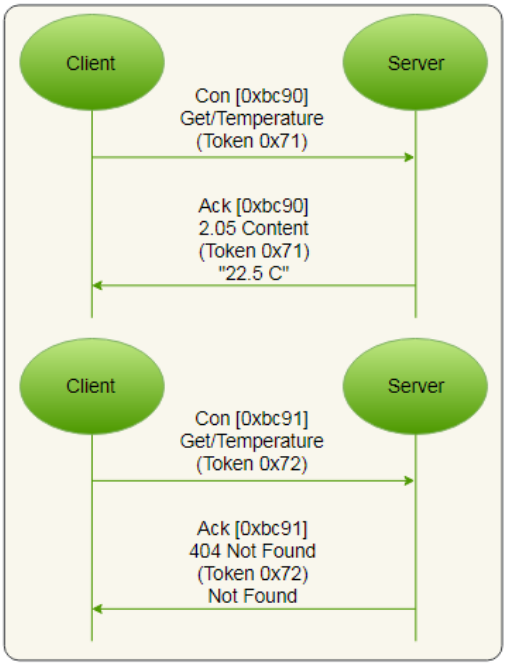

Fig 5: Successful and Failure Piggyback Messages

2.2.2.2 Separate

In the separate method of a message, the server sends an empty message rather than an acknowledgment. The purpose is to stop the client from resending the message. This message generally takes some time for delivery. The server will send a confirmable message when it is ready. A message with a separate response is shown in Figure 6.

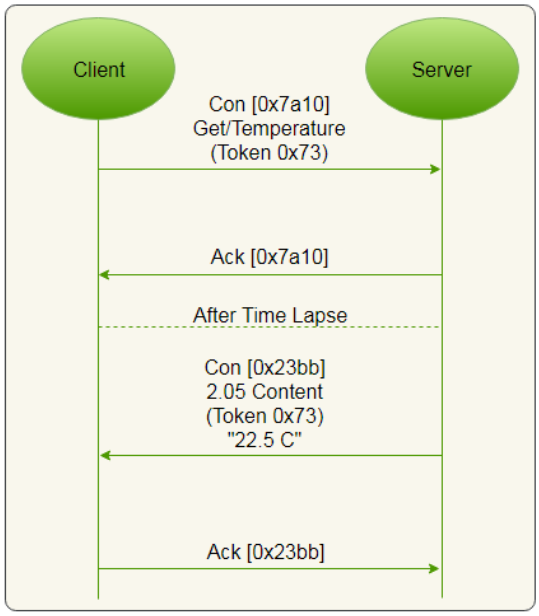

Fig 6: Message with separate response

2.2.2.3 Non-Confirmable

When a non-confirmable message is sent, the response could be non-confirmable. However, the server could also send a confirmable message. Non-confirmable request/response message is shown in Figure 7.

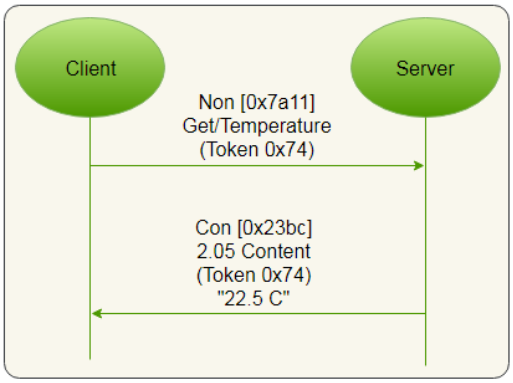

Fig 7: Non-confirmable request/response message

3. MESSAGE FORMAT OF MQTT AND COAP

Messaging format with MQTT and CoAP can be described as follows:

3.1 MQTT Message

For a connection to establish, both client and broker should have TCP/IP stack. The connection, with MQTT CONNECT, is initiated by a broker after receiving a command message from the client and will answer with CONNACK. If a spam message is requested or takes too much time, the connection will be closed. The MQTT CONNECT Message is shown in Figure 8 [10].

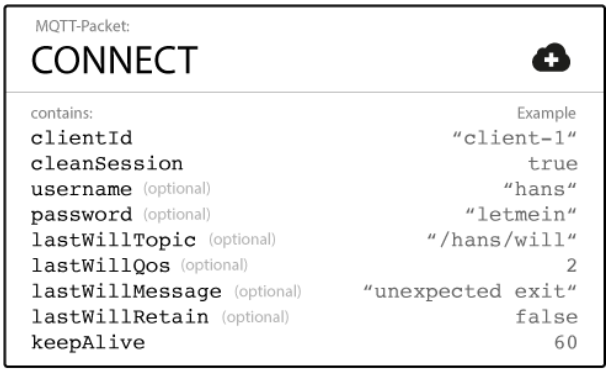

Fig 8: MQTT CONNECT

The client id is a unique identifier which is used for connecting to the broker by a client. The broker uses it to uniquely identify the client and its state. The clean session helps broker in establishing the connection in persistent and or non-persistent mode. In persistent mode, all subscription and messages will be gathered by the broker.

Authorization and Authentication of a client are managed by username and password. Will message is used in the verification of MQTT connection messages. It helps in the notification of other clients when a client disconnects. Keep

alive helps in identifying the status of both, broker and client. After a time interval, a PING Request is send by the client and broker respond back with a PING Response.

As stated earlier, the broker responds back with a CONNACK Message. This contains session present flag and a connection acknowledge flag. The MQTT CONNACK Message is shown in Figure 9 [10].

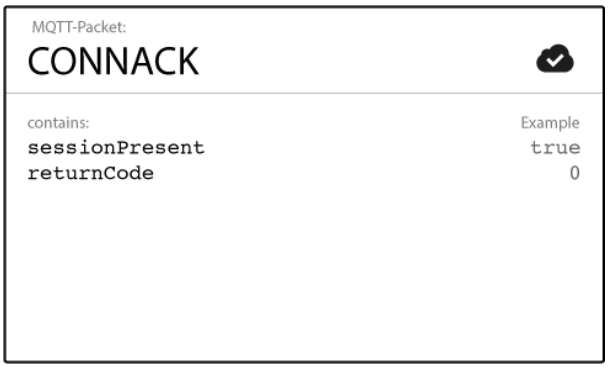

Fig 9: MQTT CONNACK

The session present flag helps in identifying if the client has a persistent session or not from earlier connections. The other flag is known as connect acknowledge flag or return code, which indicates whether a connection is successful or not. Table 1 represents the return codes from MQTT CONNACK:

Table 1. Return codes from MQTT CONNACK

Return Code	Response
0	Connection is accepted
1	Connection is refused, because of unacceptable version
2	Connection is refused, because of identifier rejected
3	Connection is refused, because of unavailability of server
4	Connection is refused, because of the wrong username or password
5	Connection is refused, because of the no authorization

Message format in MQTT is case sensitive. Naming sensors and actuators should follow a categorized and consistent order. For instance, if light, O_2, and air sensors are in the basement of an office, the correct way of naming would be:

myOffice/basement/sensor/light

myOffice/basement/sensor/O_2

myOffice/basement/sensor/air

In addition to above, wildcards can also be used.

3.2 CoAP Message

CoAP is transmitted over UDP by default. It follows a RESTful architecture which makes it lightweight to run on constrained devices. It uses a simple binary format for messaging, which is a 4-byte header along with payload option [11]. The message format for CoAP consisting of 4-byte is shown in Table 2.

Table 2. CoAP consisting of 4-bytes header

Version (V)	Type (T)	Token Length (TKL)	Code	Message ID
Token (If any)				
Options (If any)				
Payload (If any)				

The CoAP follows an architecture similar to HTTP client/server. A client usually sends a request to the server containing a method code e.g. GET, PUT, POST or DELETE. After receiving a request, the server responds back with a payload and a response code. Table 2 shows that the CoAP messages contain a binary header base with a 4-byte. A CoAP message contains following fields:

3.2.1 Vector (V):

This is a 2-bit unsigned integer indicating the version number for CoAP. Generally, it is set to one.

3.2.2 Type (T)

This is also a 2-bit unsigned integer which indicates the type of message. These are confirmable, non-confirmable, acknowledged and reset.

3.2.3 Token Length (TKL)

This indicates the length of the token. For this, 4-bit unsigned integers are used. 0-8 bits are used for indicating token length while 9-15 are reserved.

3.2.4 Code

The code is used for specifying the request or response code. It is an 8-bit unsigned integer. 1-31 bits are used for request while 64-191 bits are used for the response code. Generally, the code field for request could be GET, PUT, POST or DELETE, while the response is just a response code.

3.2.5 Message ID

To identify the redundancy of any message and the matching response, the Message ID is used. This is a 16-bit unsigned integer.

3.2.6 Token

Token length field utilizes 0-8 bytes. All requests and responses can be associated with a Token value.

3.2.7 Options

Options are affixed at end of a message, which can also be followed by another option and so on. The option can also contain a payload.

3.2.8 Payload

The payload is generally attached at the end of the UDP datagram, which is depended on the datagram size. If the length is zero, then it indicates the non-existence of payload.

4. SECURITY IN MQTT AND COAP

As security is considered a compromise between securable and highly usable application, it is of more importance among IoT. From application to network and protocols, security is an

essential factor for protecting IoT applications. Security in terms of MQTT and CoAP protocols are as follows:

4.1 Security in MQTT

MQTT security can be implemented on various layers. Different layers help in the prevention of numerous attacks. There is not any consolidated mechanism for having a secured MQTT architecture, rather than developing on already available standards. Security on the various level in MQTT can be implemented as follows:

4.1.1 Network Layer

VPN or secured network can be a source of security in Network layer among client and broker. It is helpful in offering a secure and reliable connection. VPN's are helpful in securing such gateway applications where devices communicating with gateways are on one end while VPN with broker resides on another end.

4.1.2 Transport Layer

Encryption is used in encoding data in such way that it is only accessible to authorized users and unauthorized users cannot view it [12]. Encryption can also be taken into account for securing the transport layer. The transport encryption can be managed by implementing TLS (Transport Layer Security)/DTLS (Datagram Transport Layer Security). For TCP, TLS is used, while DTLS is used for UDP [13]. As TLS is securable in transport layer, it is not feasible among resource-limited devices, mainly because of the packet overhead.

4.1.3 Application Layer

Authentication is essential towards implementing security in both layers, Transport, and Application. The TLS helps in authenticating both client and server using a certificate in the transport layer. On the other hand, authentication can be administered by providing username and password at the application layer.

A unique identifier is assigned to every client for authenticating to the broker. The unique identifier is used for connecting to the broker in MQTT CONNECT. This identifier generally contains 65535 characters, where 23 characters are cancelled out. It is considered a good practice to include the MAC address or a serial number of the device for the user id, or the user id should be at least 36 characters long. The authentication assessment is performed by the MQTT broker after providing username and password.

Authorization is yet another possibility of securing application layer, in which access rights are assigned to a specific resource. In MQTT, Access Control List (ACL) is used for authorization and is implemented on broker side [13]. The ACL allows access rights and permitted operations to be granted to a specific process. All username and passwords belonging to a user and it's published or subscribed topics are stored at ACL in MQTT.

Another authorization technique used in the perspective of MQTT security is Role Based Access Control (RBAC). In this technique, access rights against a certain resource depend on the role assigned to a user. Maintaining users with the permission become easier by assigning roles. Web services and various plugins can be used for enhancing authorization aspects.

4.2 Security in CoAP

Datagram Transport Layer Security (DTLS) is one mechanism used by CoAP on top of UDP instead of TCP to manage changes encountered in HTTP. So, Single-to-Multipoint communications can be managed by this method in CoAP. In constrained environments, DTLS can be used along with minimum configurations for secure CoAP messages.

Authentication, confidentiality, key management, and data integrity are some crucial factors managed by DTLS [14]. It also helps in the various cryptographic techniques. DTLS is considerably the most secure protocol for channel security because of its facilitation in providing authentication, protecting application data and key exchange with key management and algorithms [15]. Along with DTLS, four modes of security are used by CoAP in numerous applications. In terms of authentication and key management, these modes differ from each other as described follows:

4.2.1 NoSec

In this mode of CoAP, no security is provided, and hence all messages transferred are deprived of security.

4.2.2 PreSharedKey

Those sensing devices already programmed with cryptographic keys can be secured using this mode. These cryptographic keys in such devices help in communication with other devices. Such devices, not sustaining public key cryptography generally use this mode. In multiple devices, a single key is utilized with one key for each destination.

4.2.3 RawPublicKey

On public keys, the sensing devices that requires authentication generally adopt this security mode. However, these devices do not follow a public key structure. The public key helps in the identification of devices. This identity can be used in interacting with nodes of public keys. This security mode is considered essential in the implementation of CoAP.

4.2.4 Certificates

This security mode is also helpful for sensing devices requiring authentication on public keys but also useful in those that follow public key structure for validation. However, this mode assures the accessibility of security structure.

5. CONCLUSION

This paper focuses on the two highly utilized application protocols for IoT, MQTT, and CoAP. With many advancements in IoT applications, devices become more efficient in terms of sensors and actuators. With the involvement of networking infrastructure, many protocols schemes were devised. AMQP, XMPP, and LWM2M are named few. Along with them, MQTT and CoAP protocols are extensively used application protocol. First, we describe the architecture of MQTT and CoAP in this paper, and also show how the transmission of the message takes place. Then we present the messaging format for these protocols along with format examples. Messaging in MQTT follows a categorized and consistent order, while in CoAP, the messages follow a Restful architecture. This architecture is similar to HTTP client/server. A client usually sends a request to the server containing a method code e.g. GET, PUT, POST or DELETE. The next section then addresses the security implementation of various layers in IoT using MQTT and CoAP. In MQTT, the security can be implemented at various layers including

network, transport, and application. On the other hand, the security in CoAP follows a DTLS mechanism used on top of UDP. Four mdoes NoSec, PreSharedKey, RawPublicKey, and Certificates are used for security in CoAP. We believe that this survey paper provides the reader with the insight into IoT protocols. It will also benefit the research community with its intellectual contribution towards research and development.

6. REFERENCES

[1] https://en.wikipedia.org/wiki/Internet_of_things

[2] https://www.gartner.com/newsroom/id/3598917

[3] D. Locke, "MQ telemetry transport (MQTT) v3.1 protocol specification" IBM developer Works Technical Library 2010, http://www.ibm.com/developerworks/webservices/library/wsmqtt/index.html.

[4] https://en.wikipedia.org/wiki/MQTT

[5] Chen, Whei-Jen and Gupta, Rahul and Lampkin, Valerie and Robertson, Dale M. and Subrahmanyam, Nagesh, "Responsive Mobile User Experience Using MQTT and IBM MessageSight" IBM Corp., 2014

[6] Cristian Mateos, Andres Pablo Flores, Alejandra Cechich and Alejandro Zunino, "RESTful Service Composition at a Glance: a Survey", 2015

[7] https://en.wikipedia.org/wiki/Constrained_Application_Protocol

[8] Pallavi Sethi and Smruti R. Sarangi, "Internet of Things: Architectures, Protocols, and Applications"

[9] Z. Shelby, ARM, K. Hartke, C. Bormann, "The Constrained Application Protocol (CoAP)", https://tools.ietf.org/html/rfc7252

[10] https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment

[11] Isam Ishaq, Jeroen Hoebeke, Ingrid Moerman and Piet Demeester, "IETF Standardization in the Field of the Internet of Things (IoT): A Survey", 2013

[12] https://en.wikipedia.org/wiki/Encryption

[13] Sotirios Katsikeas, Konstantinos Fysarakis, Andreas Miaoudakis, Amaury Van Bemten, Ioannis Askoxylakis, Ioannis Papaefstathiou and Anargyros Plemenos, "Lightweight & Secure Industrial IoT Communications via the MQ Telemetry Transport Protocol", 2017

[14] Thamer Alghamdi, A. Lasebae and Mahdi Aiash, "Security Analysis of the Constrained Application Protocol in the Internet of Things", 2013

[15] Ajit A. Chavan, Mininath K. Nighot, "Secure CoAP Using Enhanced DTLS for Internet of Things", 2014.