MMQP: A Lightweight, Secure and Scalable IoT Communication Protocol

Franc Pouhela*, Sogo Pierre Sanon*, Dennis Krummacker*, Hans D. Schotten*†

*German Research Center for Artificial Intelligence (DFKI GmbH), Kaiserslautern Germany
Email: {franc.pouhela; sogo_pierre.sanon; dennis.krummacker; hans_dieter.schotten}@dfki.de
†University of Kaiserslautern (RPTU), Germany
Email: {schotten}@rptu.de

***Abstract*—The ever-evolving Internet of Things (IoT) landscape necessitates continuous advancements in communication protocols to meet evolving requirements. In response, this paper introduces the Middleware Message Queuing Protocol (MMQP), an advance, binary, lightweight and secure IoT and Machine-to-Machine (M2M) communication protocol aimed at addressing the limitations of the Message Queuing Telemetry Transport (MQTT) protocol. We present the design principles, core features, and unique advantages of MMQP in comparison to MQTT. Through practical analysis, we demonstrate MMQP's potential to enhance IoT and M2M communication.**

***Index Terms*—IoT, Protocol, Security, End-to-End, 6G**

I. Introduction

There is an ever-growing demand for efficient and continuous data collection and processing in distributed systems, notably in mobile communications networks such as 5G or the future 6G. A significant catalyst for this shift is the widespread integration of Artificial Intelligence (AI) in network operation.

A potential avenue with the keys to addressing these requirements could be a Context Management System (CoMaS) [1]. A CoMaS is a framework adept at harnessing the network context data to gain new insights into network states (context) through diverse AI algorithms. The newly gained knowledge is then leveraged to improve and optimize network operation. The efficacy of such a system hinges on its seamless and efficient acquisition and distribution of contextual data, a task exacerbated by the proliferation of IoT devices. The need for flexible, scalable and efficient communication solution arises.

MQTT, renowned as a lightweight messaging protocol, finds extensive application in both IoT and M2M scenarios where scalability and efficiency are required. Indeed, MQTT presently stands as the de facto protocol for IoT communication. However, upon closer examination of MQTT, we identified areas with potential for improvement, spanning various levels of the protocol. These shortcomings manifest in aspects such as packet structure, topic management, delivery patterns, security measures, efficiency etc.

To overcome these limitations, we designed MMQP, a similar protocol that enhances efficiency, scalability, security, and flexibility. Similar to MQTT, MMQP is a lightweight, binary protocol for IoT and M2M communication, which also follows the publisher-subscriber communication model. Our goal in this paper is to provide a clear understanding of the design principles, core features, and unique advantages of MMQP in comparison to MQTT.

The remainder of this paper is structured as follows: Section II presents a short review of related works. Section III introduces MQTT and outlines its limitations. Following this, Section IV introduces MMQP, detailing its components and features. Lastly, Section V concludes the study with a look at the potential future work.

II. Related Work

Various research efforts [2], [3], [4] have focused on addressing the limitations of MQTT and exploring alternative solutions. For instance, [5] conducted a comprehensive survey on IoT messaging protocols, where they highlighted the strengths and weaknesses of MQTT and compared it with other protocols such as Advanced Message Queuing Protocol (AMQP) and Constrained Application Protocol (CoAP), etc.

In a prior study detailed in [6], we demonstrated the advantages in terms of power efficiency gained by adopting Entity-Component-System (ECS) as the underlying architecture for implementing the broker application.

Most of the efforts invested in improving MQTT was mainly focused on security and less on performance. [7] thoroughly evaluates MQTT's security, highlighting the limitations in term of performance and scalability and [8], [9] addresses these limitations using respectively ARIA Cipher 256 Algorithm Cryptography and Elliptic Curve Cryptography.

III. MQTT Overview and Limitations

Developed in the late 1990s, MQTT [10] has since gained widespread adoption, becoming a cornerstone in the implementation of IoT applications. At its core, MQTT operates on a publish-subscribe messaging model, offering a decoupled, asynchronous, event-driven approach to message exchange. In this communication approach, participants play two distinct roles: publishers and subscribers. Publishers are responsible for generating messages and spreading them to designated topics, while subscribers express interest in specific topics and receive relevant messages in a seamless and decoupled manner.

A. MQTT Communication Pattern

This paper primarily focuses on the lates version of MQTT version 5. Figure 1 shows MQTT's flow of interactions. Initially, the publisher establishes a connection with the broker by sending a CONNECT packet, to which the broker responds with a CONNACK packet acknowledging the successful connection. Subsequently, the subscriber subscribes to a specific topic using the SUBSCRIBE packet. Meanwhile, the publisher sends a message to the broker using a PUBLISH packet, which can be queued for further processing depending on the Quality of Service (QoS). The broker then forwards the relevant message to the subscriber.

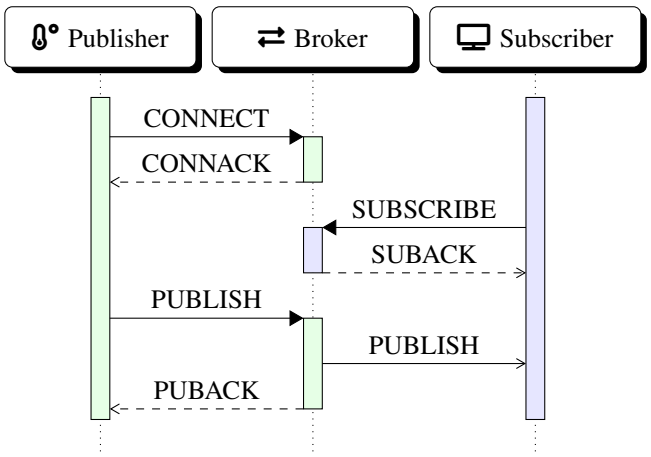

Fig. 1. MQTT Message Exchange

B. MQTT Control Packets

The table in Figure 2 displays all MQTT control packets. Generally, these packets consist of three primary components: a fixed header, a variable header, and the payload.

Packets	ID	Description	Flow
CONNECT	1	Client connection request	Client to Server
CONNACK	2	CONREQ acknowledgment	Server to Client
PUBLISH	3	Client publish request	Bidirectionnal
PUBACK	4	PUBLISH acknowledgment	Bidirectionnal
PUBREC	5	Publish received (QoS 2)	Bidirectionnal
PUBREL	6	Publish release (QoS 2)	Bidirectionnal
PUBCOMP	7	Publish complete (QoS 2)	Bidirectionnal
SUBSCRIBE	8	Client subscribe request	Client to Server
SUBACK	9	SUBSCRIBE ack...	Server to Client
UNSUBSCRIBE	10	Client unsubscribe request	Client to Server
UNSUBACK	11	UNSUBSCRIBE ack...	Server to Client
PINGREQ	12	Ping request	Client to Server
PINGRESP	13	Ping response	Server to Client
DISCONNECT	14	Client is disconnecting	Bidirectionnal
AUTH	15	Authentication	Bidirectionnal

Fig. 2. MQTT Control Packets

MQTT's capacity to include additional packet types is constrained due to its allocation of all available bits for control packet types. As illustrated in Figure 3, MQTT assigns 4 bits in the control byte, to represent packet type. Expanding the packet types would require significant protocol modifications, potentially disrupting current implementations.

1) MQTT Fixed Header: Each control packet has a fixed header (Figure 3). The packet type is represented using the four first bits of the packet's first byte. The subsequent four bits are used for various flags, depending on the packet type. The following byte(s) represent the packet's remaining length, which varies according to the packet size.

Bits	7	6	5	4	3	2	1	0
Byte 1	Packet Type (4 Bits)				Flags (4 Bits)			
Byte 2 ...	Remaining Length (1 - 4 Bytes)							

Fig. 3. MQTT Packet Fixed Header

MQTT uses the Variable Length Encoding (VLE) algorithm to the packet's remaining length. This technique represents a variable-length value using a series of bytes, where each byte uses 7-bits to represent data and the eight bit serves as a continuation flag. The algorithm is as follows:

$$\text{Encoded Byte}_i = \begin{cases} \text{Byte}_i \,\&\, 127 & \text{if Byte}_{i+1} \text{ exists} \\ \text{Byte}_i & \text{if Byte}_{i+1} \text{ does not exist} \end{cases}$$

Packets with a remaining length between 0 and 127 bytes will use one byte to encode the length. Similarly, packets with a remaining length between 128 and 16383 bytes will use two bytes, etc. The maximum size that can be encoded in MQTT using VLE depends on the number of bytes used for encoding and follows the formula: $L_r = \sum_{i=0}^{n} 127 \times 128^i$ Where n is the number of bytes used for encoding. In practice, 4 bytes are generally used to encode the size in MQTT, making the maximum size limited to around 254MB.

While VLE offers benefits such as minimizing overhead for smaller packet lengths, it also introduces some downsides. Implementing variable length encoding introduces complexity in encoding and decoding logic, potentially increasing computational overhead, particularly in resource-constrained environments. This complexity extends to error handling, where incorrect byte sequences or buffer overflows may occur, necessitating robust error detection and recovery mechanisms. Furthermore, ensuring interoperability with other MQTT clients and brokers becomes crucial, as any deviations in encoding or decoding logic can lead to communication issues and interoperability challenges.

2) Variable Header: Some MQTT packets have a variable header which resides between the fixed header and the payload. The content of this variable header varies depending on the packet type. For instance, the PUBLISH packet can contain a packet identifier for acknowledgement.

Figure 4 describes a PUBLISH packet. In MQTT. The two packet flags, DUP and RET, respectively stand for DUPLI-

Bits	7	6	5	4	3	2	1	0
Byte 1	PUBLISH				DUP	QoS		RET
Byte 2	Remaining Length							
Byte 3	Topic Length MSB (0)							
Byte 4	Topic Length LSB (3)							
[...]	Topic ("room1/light")							
Byte 15	Packet Identifier MSB (0)							
Byte 16	Packet Identifier MSB (2)							
[...]	Payload							

Fig. 4. PUBLISH Control Packet

CATE and RETAIN. The DUPLICATE flag informs the broker that the currently sent packet is a duplicate. This can occur when the broker fails to acknowledge a packet sent by the publisher. The RETAIN flag instructs the broker to retain the last message sent on a particular topic and deliver it to any new subscribers immediately upon connection.

The retain feature lacks fine-grained control. Firstly, it's not possible to specify retention properties on a per-topic basis such as a message expiry duration. This limitation can be problematic in applications where message freshness is critical, as retained messages may become outdated over time. Additionally, MQTT lacks retain modes for scenarios requiring the retention of multiple messages using First-In-First-Out (FIFO) or Last-In-First-Out (LIFO) patterns. These modes could provide additional flexibility in delivering retained messages on specific topics.

C. Quality of Service Levels

MQTT supports three levels of QoS for message delivery.

QoS 0 (At most once): Also known as best effort delivery, ensures the delivery without any acknowledgment.

QoS 1 (At least once): Here, the sender publishes the message, and the receiver acknowledges its receipt. If the acknowledgment is not received, the publisher may resend the message. (see Figure 1).

QoS 2 (Exactly once): This QoS level guarantees that the message is delivered exactly once. It involves a two-step process as depicted in Figure 5.

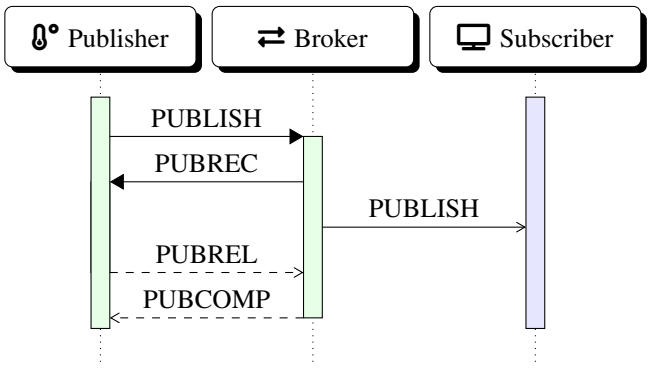

Fig. 5. Publishing with QoS level 2

D. Topic Management

In MQTT, topics serve as crucial identifiers for organizing and disseminating messages. Managed hierarchically by brokers, topics are represented as strings with multiple levels separated by forward slashes "/". When a client publishes a message, the broker evaluates the topic hierarchy to determine relevant subscribers. The broker forwards the message to subscribers interested in topics matching the published topic or its parent levels. For instance, publishing a message on *home/room1/light* triggers the broker to also deliver the message to subscribers interested in *home/room1*, or *home*.

One drawback of MQTT is its limited support for topic manipulation. It primarily relies on SUBSCRIBE and UNSUBSCRIBE packets for topic requests, lacking a direct mechanism for adding additional topic requests. This limitation can affect the flexibility and dynamism of MQTT-based systems, particularly in scenarios requiring dynamic topic management based on changing requirements. For instance, in scenarios where a publisher desires more control over message forwarding or retention on specific topics, etc.

1) Topic Alias: In MQTT v5.0, the Topic Alias feature optimizes bandwidth usage and reduces the size of published messages by substituting commonly used topics with shorter aliases. Clients negotiate topic aliases with the broker during connection establishment, specifying a maximum alias they intend to use. The broker responds with its own maximum, which may differ. Adhering to the broker's maximum is crucial to avoid issues. When publishing a message for the first time on a topic, the client includes both the alias and the topic string. Upon receiving this initial message, the broker maps the alias to the corresponding topic. Subsequently, the client can use the alias alone, omitting the topic string.

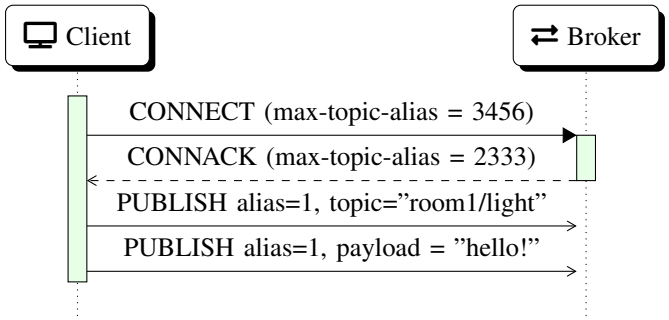

Fig. 6. Topic Alias Negotiation

A topic such as *Germany/Kaiserslautern/BismarckStreet/TrafficLight2* has 51 characters. In contrast, using a 4-bytes topic alias means that 47 extra bytes are wasted when using the topic string. Considering a scenario with 10,000 devices exchanging messages at a rate of 10Hz, the total extra bytes used would be approximately 5 MB/s, equivalent to around 13 terabytes in a month. The plot depicted in Figure 7 provides a visual representation.

Topic aliases offer significant improvements in bandwidth efficiency, yet certain limitations warrant attention. Firstly, the

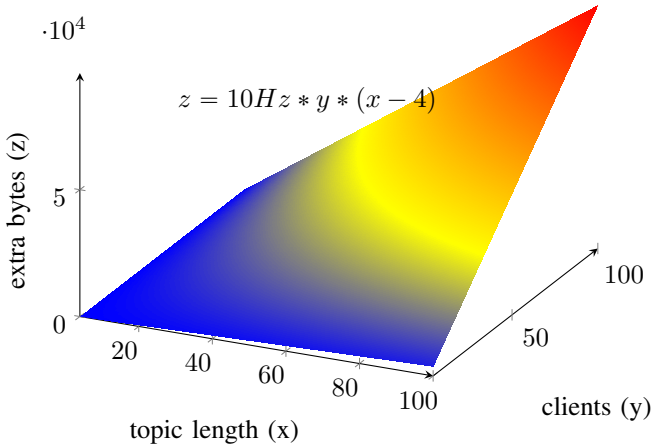

Fig. 7. Topic's extra bandwidth usage

client must negotiate the topic alias range during connection handshake. The client must adhere to the range provided by the broker, which may not align with its preferences and the client's use of topic aliases is restricted by the maximum alias provided by the broker. Secondly, the variable nature of the topic alias range complicates the implementation of client applications. Given the benefits of topic alias, addressing these limitations should be a priority in protocol design.

E. Topic Wildcards

In MQTT, topic wildcards, such as "+" (single-level) and "#" (multi-level), enable subscribers to efficiently subscribe to multiple topics simultaneously. The "+" wildcard matches any single level in a topic hierarchy, while the "#" wildcard matches all levels. For instance, subscribing to *home/+/light* means we will receive messages published on *home/room1/light*, *home/room2/light*, etc. Respectively, subscribing to *home/#* means we will receive messages published on *home*, *home/room1/light* etc.

F. Last Will Testament

The Last Will Testament (LWT) allows a client to specify a message that will be published by the broker on its behalf when the client unexpectedly disconnects. during the connection handshake using the CONNECT packet (see Figure 8), the client can specify a last will message along with a topic and QoS level to publish it.

There are some potential downsides to point out regarding the CONNECT packet. Firstly, the utilization of connect flags could be optimized for greater efficiency. Currently, more bits are allocated than necessary for the LWT. Streamlining this process to use only one bit would reduce overhead and enhance protocol efficiency, while also potentially enabling Initial Will Message (IWM) support. Introducing IWM functionality, where a device transmits its online status to the broker upon connection or resumption, could significantly improve functionality, particularly in request/response scenarios requiring knowledge of a responder's online status.

Bits	7	6	5	4	3	2	1	0
Byte 1	CONNECT				0	0	0	0
Byte 2	Remaining Length (1 - 4 Bytes)							
Byte 3	Protocol Length MSB							
Byte 4	Protocol Length LSB							
[...]	Protocol ID ("MQTT")							
Byte 9	Protocol Version							
Byte 10	User Name	Pass-word	Will Retain	Will QoS		Will Flag	Clean Start	0
Byte 11	Keep Alive Interval MSB							
Byte 12	Keep Alive Interval LSB							
[...]	Payload							

Fig. 8. CONNECT Packet Layout

G. Session Keep Alive

The Keep Alive feature in MQTT ensures the health and availability of clients by enabling them to periodically communicate with the broker. It involves a time-based mechanism where the client and broker exchange control packets at regular intervals known as the Keep Alive Interval. This ensures that both parties remain aware of each other's connectivity status. The keep alive interval is provided to the broker via the CONNECT packet (see Figure 8) during connection establishment, which is a time duration in seconds.

This interval indicates how often the client will send a ping request, PINGREQ to the broker to signal its continued presence. Upon receiving this packet, the broker must respond with a ping response PINGRESP. This is only done if no prior activity between the client and the broker has been registered. If the broker fails to respond to the ping request of the client, this last can assume connection loss and take specific actions.

In a traditional client-server mindset, the server is the one providing services to clients. Having the broker initiate the Keep Alive handshake makes the implementation of the client less complex. Clients can focus on consuming desire messages and handling data without needing to manage the connection state. This simplification streamlines client development and reduces the risk of implementation errors.

H. Request/Response Pattern

MQTT was not made to support request/response pattern. This feature was later added in version 5. This involves a client subscribed to a response topic sending a request and waiting for a corresponding response from another client subscribed to that request topic. The sequence diagram depicted in Figure 9 illustrates how this typically works in MQTT. Both the Requester and Responder initiate connection by assigning the request/response information identifier within the CONNECT packet. This helps the client to solicit the server to include response information in the CONNACK packet. This response information serves as a distinct segment of the response topic, facilitating the server's permission verification process.

The requester and the responder proceeds by respectively subscribing to the desired response/request topic *resp-topic*,

req-topic. The requester proceeds by sending a request message to the broker on the request topic. Upon receiving the request, the Broker forwards it to the Responder by publishing the message to the request topic. The responder processes the request and publishes its response back to the broker using the response topic. Finally, the broker then forwards the response message to the requester using the response topic.

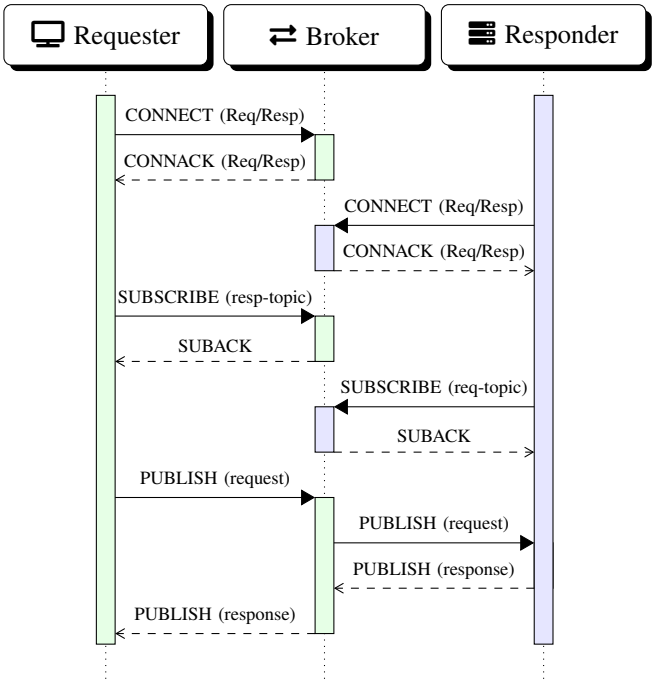

Fig. 9. MQTT Request/Response Exchange

While the MQTT's request/response pattern provides a means for applications with such requirement to achieve their purposes, it also has several drawbacks that cannot be neglected. First, the negotiation process and the need for the requester to subscribe to a response topic before sending a request is not the most flexible approach. As the number of clients and the frequency of request/response interactions increase, the broker may experience scalability challenges. Managing numerous topics for requests and responses, along with the associated message handling overhead, can strain broker resources and impact system performance. Best-effort message delivery model means that there are no guarantees of message delivery or order of arrival. In request/response scenarios where reliability is crucial, additional mechanisms such as QoS levels and message acknowledgment may be necessary to ensure message delivery and integrity.

Another limitation is that MQTT brokers usually restrict the topics that clients can publish and subscribe to. The requester can specify a random response topic, but cannot guarantee that it has permission to subscribe to that topic, nor can it guarantee that the responder has permission to publish messages on it. Finally, Implementing the request/response pattern of MQTT requires additional logic to manage topics for requests and responses, as well as mechanisms for correlating requests with their corresponding responses. This complexity can lead to more intricate client and broker implementations and potentially increase the risk of failures.

I. Security/Privacy

MQTT provides some security features to protect the confidentiality, integrity, and availability of data transmitted over the network. The main security mechanisms in MQTT include: **Access Control Lists (ACLs)**: Brokers can implement ACLs to enforce fine-grained access control policies. ACLs allow brokers to specify which clients are allowed to publish or subscribe to specific topics. **Authentication**: MQTT can supports various authentication methods, including username/-password and client certificate. **Transport Layer Security (TLS)**: MQTT supports TLS encryption, which encrypts the communication channel between clients and brokers to prevent eavesdropping and tampering by malicious actors.

One major drawback of TLS is its heaviness in terms of computation. TLS encryption involves complex cryptographic algorithms and computations that makes it difficult and almost impossible for resource constraint devices to use it for encryption. Moreover, TLS's encryption capabilities primarily focus on securing the communication channel between clients and brokers. MQTT therefore does not natively support end-to-end encryption, which means that messages may be decrypted by the broker before being forwarded to subscribers, potentially exposing them to insider threats.

IV. Introducing MMQP

As depicted in Figure 10, MMQP adheres to the same communication model as MQTT. The only notable difference thus far lies in the nomenclature of the packets. The SUBSCRIBE packet for topic subscription is replaced by a more versatile packet called TOPREQ (topic request), facilitating a broader spectrum of potential topic requests. This will be further elucidated as we delve into topic management in MMQP.

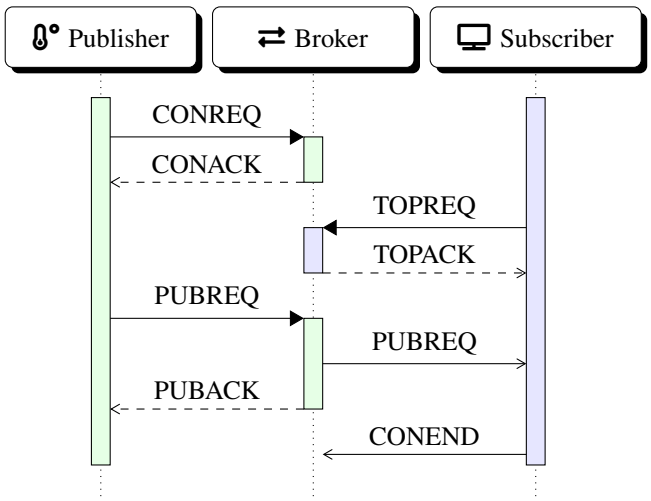

Fig. 10. MMQP Communication Pattern

A. MMQP Control Packets

The table shown in Figure 11 outlines all control packets of MMQP. Notably, there are only 10 such packets. This marks an improvement over the limitation highlighted in the MQTT section, where it was observed that MQTT exhausts all available bits for packet representation. These 10 packets are sufficient to provide the same features with more flexibility. By reorganizing the packets and their layout effectively, we managed to decrease the number of packets necessary to support all features and QoS levels of MQTT. Further elucidation on this optimization will become evident as we proceed.

Packets	ID	Description	Flow
CONREQ	1	Client connection request	Client to Server
CONACK	2	CONREQ acknowledgment	Server to Client
CONSYN	3	Keep alive packet (PING)	Bidirectionnal
CONEND	4	Client is disconnecting	Bidirectionnal
TOPREQ	5	Client topic request	Client to Server
TOPACK	6	TOPREQ acknowledgment	Bidirectionnal
PUBREQ	7	Client publish request	Bidirectionnal
PUBSYN	8	Publish control (QoS 2)	Bidirectional
PUBACK	9	PUBREQ acknowledgment	Bidirectionnal
AUTHEN	10	Authentication packet	Bidirectionnal

Fig. 11. MMQP Control Packets

B. MMQP Packets Format

Similar to MQTT, MMQP's control packets consist of a fixed header, an optional variable header, and payload. The fixed header, depicted in Figure 12, entails the control byte and the encoded remaining length. The notable distinction lies in the encoding of the remaining length. Within the control byte, 2-bits are allocated to represent the remaining length interval or size, these 2-bits are referred to as the **Remaining Length Flags (RLF)**. For a packet with a remaining length value using only 1-byte such as 120 or in general with a value within [0, 255]. The RLF would be set to 00 and a remaining length within [256, 65535] (2 bytes), the RLF would be 01 etc.

Bits	7	6	5	4	3	2	1	0
Byte 1	Control Packet				RLF		Flags	
Byte 2	Remaining Length (1 - 4 Bytes)							

Fig. 12. MMQP Fixed Header

This remaining length encoding approach is straightforward and doesn't necessitate extensive computation. By just looking at the RLF, we can promptly determine the precise number of bytes used to encode the remaining length. Additionally, the remaining length can go up to 4GB, making room for a wider range of possible use cases for this protocol.

C. Connection Establishment

The connection establishment starts with the client sending a CONREQ packet to the broker (see Figure 10). This packet's format is illustrated in Figure 13. Notably, it bears resemblance to MQTT's CONNECT control packet depicted in Figure 8. However, the disparity lies in the utilization of the session flags also known as "Connect Flags" in MQTT. Instead of allocating most bits to represent various properties of the LWT, each bit now represents a properties category.

Bits	7	6	5	4	3	2	1	0
Byte 1	CONREQ (1)				RLF		RES	0
Byte 2	Remaining Length (1 - 4 Bytes)							
Byte 3	Protocol Length (4)							
[...]	"MMQP"							
Byte 8	Protocol Version (1)							
Byte 9	AUTH	SET	IWM	LWM	TOP	0	0	0
Byte 10	Client ID Length (12)							
[...]	"Device-12392"							
[...]	Optional Properties							

Fig. 13. CONREQ Control Packet

1) Resume Flags: The resume session control flags RES informs the broker about a previously existing session that may have been interrupted due to a connection loss or store intentionally using the persistent flag. When this flag is set in the CONREQ packet, the broker will resume the session using the stored properties and send any retained messages if the session was still active.

2) Session Flags: The CONREQ packet uses the session flags (Byte 9 of Figure 13) to optionally provide additional session properties such as session settings, authentication data, etc. Following are the descriptions of each session flag:

a) Authentication Flag: The authentication property flag AUTH allows the client to provide an authentication method along with its associated data to the broker, thereby enabling authentication between both parties (see Figure 14).

CONREQ: Authentication Properties								
Bits	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Byte 1	USER	PWD	MTHD	DATA	0	0	0	0
Byte 11	Username Length (8)							
[...]	"username"							
Byte 20	Password Length (8)							
[...]	"password"							
[...]	Method & Data...							

Fig. 14. Authentication properties

b) Session Settings Flag: The session settings flag SET tells the broker about the presence of session settings (see

Figure 15) in the payload. Among these settings are the Keep-Alive Interval, the Session Expiry Interval as well as the persistent flag. The persistent flag PER instructs the broker to store all session properties, such as Topics, IWM, and Last Will Message (LWM) persistently. This allows the client to quickly resume its session by simply sending its client identifier in subsequent connections. The expiry interval is used to specify how long the session should remain active when the client unexpectedly lost its connection.

CONREQ: Session Properties								
Bits	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Byte 1	PER	KPAL	EXP	0	0	0	0	0
Byte 2	Keep Alive MSB (0)							
Byte 3	Keep Alive LSB (60)							
Byte 4	Expiry MSB (0)							
Byte 5	Expiry LSB (60)							

Fig. 15. Session properties

c) Initial/Last Will Flags: Clients can use these flags to send their IWM, IWM to the broker so that this last can properly publish them when required (see Figure 16). The retain flag RET tells the broker to retain the will message for later coming subscribers. The will message can also have a publish delay. This delay instructs the broker on how long to wait before publishing the will message. The client also has the option of providing an expiry of the will message using the expiry flag EXP.

CONREQ: Will Properties								
Bits	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Byte 1	EXP	DLY	QoS		RET	0	0	0
Byte 2	Topic Length (14)							
[...]	"client1/status"							
Byte 17	Message Length (6)							
[...]	"online"							
Byte 24	Expiry MSB (0)							
Byte 25	Expiry LSB (60)							
Byte 26	Delay MSB (0)							
Byte 27	Delay LSB (10)							

Fig. 16. Will Message Properties

Upon receipt of the CONREQ packet, the broker responds with a CONACK packet to acknowledge the request success or failure by providing a reason code. Figure 17 displays the format of the CONACK packet. In scenarios where authentication is required, both parties may authenticate themselves using the authentication packet AUTHEN.

d) Topic Request Flag: This flag can be utilized to provide a list of topic requests to the broker within the CONREQ packet. Using this, the client can directly subscribe or register topics during the connection handshake without having to send a separate TOPREQ packet. Topic requests are discusses in Section IV-F.

Bits	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Byte 1	CONACK (2)				RLF		0	0
Byte 2	Remaining Length (2)							
Byte 4	Reason							
Byte 3	0	0	0	0	0	0	0	0

Fig. 17. CONACK Control Packet

D. Session Keep Alive

The session is maintained alive using the CONSYN control packet (see Figure 18). If the client remains inactive within the keep-alive interval, the broker initiates the handshake by sending the CONSYN packet, which the client is also required to respond to using the same packet.

Bits	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Byte 1	CONSYN (3)				RLF		0	0
Byte 2	Remaining Length (0)							

Fig. 18. CONSYN Control Packet

E. Disconnection

To terminate the connection, the client or the broker can utilize the CONEND packet by specifying a reason code. The broker can close a client connection without sending a CONEND if disclosing the reason could compromise security.

Bits	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Byte 1	CONEND (4)				RLF		0	0
Byte 2	Remaining Length (1)							
Byte 3	Reason							

Fig. 19. CONEND Control Packet

F. Topic Management

MMQP consolidates all topic requests inside of the TOPREQ packet (Figure 10). It supports three types of requests: SUBSCRIBE, REGISTER and UNSUBSCRIBE.

The new REGISTER request allows publishers to register topics they wish to publish messages on by instructing the broker on how (OVERRIDE, FIFO, LIFO) and for how long messages should be retained. It also includes a packet identifier for proper acknowledgment using the TOPACK control packet (see Figure 21).

Clients are required to subscribe/register all topics they wish to respectively receive and send messages on. These can also be done using wildcards (*#, +*) as described in Section III-E.

Bits	7	6	5	4	3	2	1	0
Byte 1	TOPREQ (5)				RLF		0	0
Byte 2	Remaining Length							
Byte 3	Packet ID MSB							
Byte 4	Packet ID LSB							
[...]	Payload							

Fig. 20. TOPREQ Control Packet

Bits	7	6	5	4	3	2	1	0
Byte 1	TOPACK (6)				RLF		0	0
Byte 2	Remaining Length							
Byte 3	Packet ID MSB							
Byte 4	Packet ID LSB							
Byte 5	Reason							

Fig. 21. TOPACK Control Packet

In that case the topic alias is set to zero as no specific topic string is provided in the request. This gives the broker prior knowledge about which client will be publishing on specific topics which can enhance access management and security.

1) Topic Requests: Each topic request contains a request flags which is use to identify its properties such as the type, the QoS level, etc. (see Figures: 22,23,24) Each request also includes a topic alias and may optionally include the topic string. The topic string is necessary only during the first request, subsequent requests require only the topic alias. This simplification is crucial, especially for scenarios requiring re-subscription to adjust the QoS level.

a) Subscribe Request: Figure 22 depicts the layout of a subscribe request. The flag GRP indicates to the broker that the client wishes to share this subscription with other clients found in the group called *groupName*. This can be useful in scenarios where multiple clients subscribe to the same topic, but only one should receive and process the message.

TOPREQ: Subscribe						
Byte 1	SUB (0)	QoS	TOP	REQ	GRP	0
Byte 2	Topic Alias MSB					
Byte 3	Topic Alias LSB					
Byte 4	Topic Length (11)					
[...]	"room1/light"					
Byte 16	Group Length (9)					
[...]	"groupName"					

Fig. 22. Subscribe Request

The request flag REQ is used to informs the broker that this topic support request/response, meaning the subscriber is allowing the broker to forward requests from publishers. The QoS levels (0, 1, 2) instruct the broker on how to forward messages to the subscriber. In case of a subscription using wildcards, the broker will provide both the topic string and the topic alias in the first PUBREQ packet publish on one of the subtopics forwarded to the subscriber (see Section IV-G). The topic flag TOP helps communicate this. The first byte of a topic request describes the request flags. The first two bits of this byte represent the request type.

b) Register Request: Similar to the subscribe request, the register request (see Figure 23) also includes a QoS, indicating to the broker how to retain messages published on the topic. QoS 0 is the default retain mode, where only the lastest published message is retained. QoS 1 instructs the broker to retain messages using the FIFO pattern, ensuring messages are delivered in the order of arrival. QoS 2 instructs the broker to retain messages using the LIFO pattern, where the most recent message sent is the first to be delivered. Note that only packets published with QoS levels 1 and 2 can be retained.

TOPREQ: Register								
Bits	7	6	5	4	3	2	1	0
Byte 1	REG (1)		QoS		TOP	REQ	EXP	0
Byte 2	Topic Alias MSB							
Byte 3	Topic Alias LSB							
Byte 4	Topic Length (11)							
[...]	"room1/light"							
Byte 16	Message Expiry MSB (0)							
Byte 17	Message Expiry LSB (60)							

Fig. 23. Retain request

Additionally, the expiry flag EXP is used to manage the life cycle of the retained packets. This enhanced control is one of the key strengths of MMQP and represents a significant improvement over the basic control offered by MQTT. Moreover, the request also includes a topic alias which allows the broker to refer back to the client using the alias in case of failure. The client can subsequently publish messages using this topic alias without including the topic string in the packet, enhancing efficiency.

c) Unsubscribe Request: The unsubscribe request (see Figure 24) includes the topic alias, as a client must be subscribed to the topic in order to unsubscribe. A topic string can only be present if using wildcard (”#”, ”+”) unsubscription, in this case, the topic alias is set to zero.

G. Message Publishing

Clients can publish messages in MMQP using the PUBREQ control packet (see Figure 25). The packet's control byte provides the type of publish request using the Packet Control Flags (PCF). Three publish request types exist: PUBLISH (00) found in MQTT, REQUEST (01), and RESPONSE (10).

Each PUBREQ packet includes a packet identifier and a topic alias. The only time a PUBREQ packet is allowed to

TOPREQ: Unsubscribe								
Bits	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Byte 1	UNSUB (2)		0	0	TOP	0	0	0
Byte 2	Topic Alias MSB							
Byte 3	Topic Alias LSB							
[...]	"room1/#"							

Fig. 24. Unsubscribe request

Bits	7	6	5	4	3	2	1	0
Byte 1	PUBREQ (7)				RLF		TYPE	
Byte 2	Remaining Length (1 - 4 Bytes)							
Byte 5	RET	DUP	QoS		TOP	0	0	0
Byte 6	Packet ID MSB							
Byte 7	Packet ID LSB							
Byte 8	Topic Alias MSB							
Byte 9	Topic Alias LSB							
Byte 10	Topic Length (11)							
[...]	"room1/light"							
[...]	Payload							

Fig. 25. PUBREQ Control Packet

provide a topic string is when the topic was registered using wildcards. Upon receipt, the broker will forward the message to the subscriber(s) accordingly.

H. Publish QoS Levels

Similar to MQTT, MMQP also supports three levels of QoS. QoS 0 is for best effort delivery. QoS 1 requires an acknowledgement using the PUBACK (Figure 26) packet.

Bits	7	6	5	4	3	2	1	0
Byte 1	PUBACK (9)				RLF		TYPE	
Byte 2	Remaining Length							
Byte 3	Packet ID MSB							
Byte 4	Packet ID LSB							
Byte 5	Reason							

Fig. 26. PUBACK Control Packet

QoS level 2 involves a two way hand shake using the PUBSYN control packet (see Figures: 27, 28).

I. Request/Response

Employing the PUBREQ packet for request/response exchanges (see Figure 29) offers the advantage of maintaining support for all QoS levels. This also takes away the need to negotiate these features during the connection establishment because both parties have a concrete way of discerning a request/response packets from regular publish packets.

Bits	7	6	5	4	3	2	1	0
Byte 1	PUBSYN (8)				RLF		TYPE	
Byte 2	Remaining Length							
Byte 3	Packet ID MSB							
Byte 4	Packet ID LSB							
Byte 5	Reason							

Fig. 27. PUBSYN Control Packet

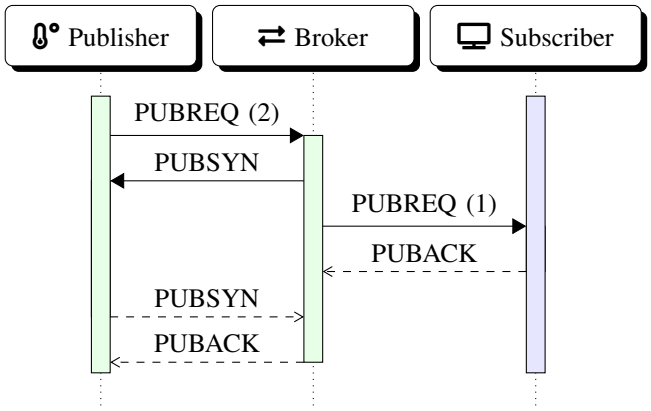

Fig. 28. Publishing with QoS level 2

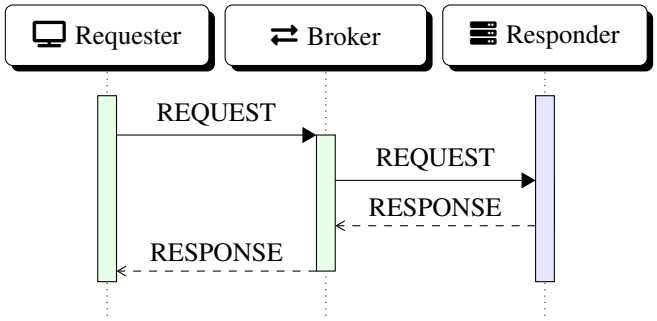

Fig. 29. Request/Response Exchange

J. Security/Privacy

MMQP introduces a lightweight and robust end-to-end encryption mechanism to safeguard data exchange between clients (see Figure 30). One of the primary objectives of the security architecture is to protect the privacy of communication and ensure that the Broker cannot access or intercept sensitive data, exchanged between clients. An Admin is introduced to play a crucial role in key generation and management.

*a) **Key Generation**:* The Admin securely generates symmetric keys for each topic. These keys are used for encryption and decryption of messages published and subscribed to that specific topic.

*b) **Key Delegation**:* When a client wishes to participate in a specific topic, they securely communicate with the Admin. The Admin verifies their credentials and then shares the corresponding symmetric key for that topic.

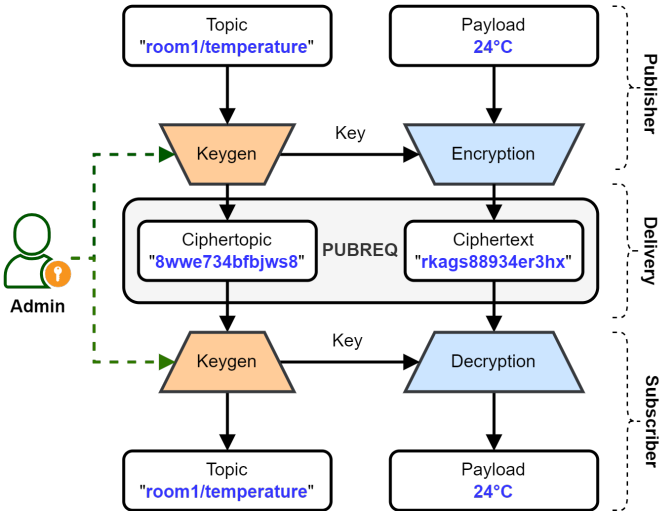

Fig. 30. End-to-End Encryption Model

1) ASCON Cryptosystem: ASCON [11] is a symmetric cryptosystem optimized for resource-constrained devices. It offers efficient encryption, decryption, and additional functionalities like, integrity checking, hashing, all while maintaining strong security against known attacks [12].

As of February 2023, it has been selected by the National Institute of Standards and Technology (NIST) for future standardization in lightweight cryptography. This recognition further validates ASCON's robustness and suitability for securing communication in resource-constrained environments.

MMQP utilizes the ASCON cryptosystem for encrypted communication. ASCON is chosen for its efficiency, security, and suitability for IoT environments, providing a reliable method for securing data exchanges.

*a) **Publisher Encrypts**:* Before publishing a message to a topic, the publisher encrypts the message content using the shared symmetric key received from the Admin. This ensures that only authorized clients with the same key can decrypt it.

*b) **Broker Acts as Relay**:* The encrypted message is then sent to the Broker. However, due to the encryption, the Broker cannot decipher the message content. It simply acts as a relay, forwarding the message to all subscribed devices. ASCON offers authenticated encryption with associated data (AEAD), which is a variant of encryption that allows for the inclusion of "associated data" (AD) alongside the encrypted message. AD refers to additional non-confidential information, also known as "additional authenticated data" (AAD). This means that while the content of the topics in MMQP can be encrypted to ensure confidentiality and integrity, the topic itself can be kept in plaintext, allowing for efficient routing and processing of messages without compromising security.

*c) **Subscriber Decrypts**:* Upon receiving the encrypted message, authorized subscribers who possess the same symmetric key can decrypt the message content.

With end-to-end encryption, only authorized devices can access the message content, protecting sensitive information from unauthorized parties, including the Broker. The approach is also scalable. The key delegation approach allows for efficient key management with a large number of devices. In addition, the lightweight nature of ASCON makes it suitable for resource-constrained IoT devices.

V. Conclusion and Future Work

In conclusion, MMQP provides significant improvements to the limitations present in MQTT. It offers increased flexibility in terms of number of packets, their structure, and format, along with extended features for topic management and fine-grained control down to the topic level. Additionally, MMQP introduces a more scalable and efficient request/response pattern, as well as a lightweight encryption solution that enables secure communication even on resource-constrained devices.

Potential future work includes evaluating Key Performance Indicatorss (KPIs) such as latency across diverse scenarios. This analysis could encompass examining latency under varying network conditions, different message payload sizes, and during peak usage periods, etc.

Acknowledgment

The authors acknowledge the financial support by the German *Federal Ministry for Education and Research (BMBF)* within the project Open6GHub {16KISK003K}.

References

1 Pouhela, F., Krummacker, D., and Schotten, H. D., "Towards 6G Networks," in *A Context Management Architecture for Decoupled Acquisition and Distribution of Information in Next-Generation Mobile Networks*, ser. ITG, vol. 157, VDE. IEEE, 5 2023.

2 Fischer, M., Fischer, M., Kumper, D., Kumper, D., Tönjes, R., and Tönjes, R., "Towards improving the privacy in the mqtt protocol," *Global Internet of Things Summit*, 2019.

3 Marra, A. and al., "Improving mqtt by inclusion of usage control," *International Conference on Security, Privacy, and Anonymity in Computation, Communication, and Storage*, 2017.

4 Luzuriaga, J. E. and al, "Improving mqtt data delivery in mobile scenarios: Results from a realistic testbed," *Mobile Information Systems*, 2016.

5 Jaloudi, S., "Communication protocols of an industrial internet of things environment: A comparative study," *Future Internet*, vol. 11, 03 2019.

6 Pouhela, F., Krummacker, D., and Schotten, H. D., "Entity component system architecture for scalable, modular, and power-efficient iot-brokers," in *2023 IEEE 21st International Conference on Industrial Informatics (INDIN)*, 2023, pp. 1–6.

7 Al-Ani, A., Shen, W. K., Al-Ani, A. K., Laghari, S. A., and Elejla, O. E., "Evaluating security of mqtt protocol in internet of things," in *2023 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2023, pp. 502–509.

8 Iqbal, M., Ari Laksmono, A. M., Prihatno, A. T., Pratama, D., Jeong, B., and Kim, H., "Enhancing iot security: Integrating mqtt with aria cipher 256 algorithm cryptography and mbedtls," in *2023 International Conference on Platform Technology and Service (PlatCon)*, 2023, pp. 91–96.

9 Yusoff, Z. Y. M., Ishak, M. K., Rahim, L. A. B., and Ali, O., "Elliptic curve cryptography based security on mqtt system for smart home application," in *2022 19th International Conference on Electrical Engineering/-Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, 2022, pp. 1–4.

10 OASIS. Mqtt version 5.0, oasis standard. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

11 Dobraunig, C., Eichlseder, M., Mendel, F., and Schläffer, M., "Ascon v1. 2: Lightweight authenticated encryption and hashing," *Journal of Cryptology*, vol. 34, pp. 1–42, 2021.

12 Turan, M. S., Turan, M. S., McKay, K., Chang, D., Bassham, L. E., Kang, J., Waller, N. D., Kelsey, J. M., and Hong, D., *Status report on the final round of the NIST lightweight cryptography standardization process*. US Department of Commerce, National Institute of Standards and Technology, 2023.