CS544 - Computer Networks
Internet-Draft
Intended status: Experimental
Michael Oravec
Drexel University
15 May 2025

MQTT over QUIC (MoQ)

Abstract

This document specifies MoQ, an implementation of the MQTT application protocol over the QUIC transport protocol. By leveraging QUIC's built in TLS security, reliable delivery, and multiplexing streams, MoQ aims to provide a more robust alternative to MQTT over TCP.

Table of Contents

1. Service Description

1.1. Introduction

Message Queuing Telemetry Transport (MQTT) is a popular publish/subscribe messaging protocol used for Internet of Things (IoT) devices. The full definition for MQTT can be found in RFC 9431. MQTT uses TCP, which adds extra overhead to what should be very lightweight data. The theoretical maximum payload for MQTT is 256 MB though services like AWS IoT Core will impose smaller limits like 128 KB.

QUIC (sometimes treated as the acronym Quick UPD Internet Connection) is specified in RFC 9000, and offers a number of advantages over TCP such as built in TLS encryption, multiplexed streams to avoid head of line blocking, and improved congestion control.

1.2. Mapping MQTT to QUIC

MoQ leverages QUIC’s multiplexed steams to divide up aspects of the MQTT protocol within a single QUIC connection, such as assigning each subscribed topic to it’s own stream rather than trying to implement a strategy for packaging multiple publications into a single QUIC frame. The following model shown in Figure 1 shows the basic mapping of an MQTT packet fitting inside a QUIC Frame’s Stream Data.

```
+------------------------------------------------------+
|               QUIC Packet                            |
+-----------------------+------------------------------+
| QUIC Header           | AEAD Protected Payload       |
+-----------------------+------------------------------+
                                      |
                        +------------------------------+
                        |     QUIC Stream Frame        |
                        +--------------+---------------+
                        | Stream ID    | Offset        |
                        +--------------+---------------+
                        | Length       |  Stream Data  |
                        +--------------+---------------+
                                               |
                                       +-----------------------+
                                       |       MQTT Packet     |
                                       +---------+-------------+
                                       |   Fixed |    Variable |
                                       |   Header|      Header |
                                       +---------+-------------+
                                       |       Payload         |
                                       +-----------------------+
```

Figure 1 Mapping of MQTT to QUIC

This lays the groundwork to applications which can extend MQTT to much higher bandwidth use cases, assuming the proper serialization is implemented. QUIC’s unique Steam ID and packet number opens the door to encapsulate, for example, video data within the MQTT payload.

A further intricacy of leveraging QUIC’s Streams is to dedicate one stream specifically to Control packets. All the connection, publish, subscribe, ping, etc. commands would be handled through the first Frame. The subsequent Frames would be the respective data streams, each with their on Quality of Service (QoS) level.

1.2.1. Establishing a Connection

Unique to QUIC is the RTT-0 handshake where TLS authentication is performed in the first exchange of packets. MoQ would leverage this feature by sending the CONNECT control packet. This expects a CONNACK to be returned by the Broker in addition to QUIC’s handshake, otherwise the connection will be forced to terminate. Both QUIC and MQTT must achieve authentication in the first round.

```
  MoQ Client                                    MoQ Broker
---------------                              ---------------
      | Initial QUIC Handshake                        |
      | (0-RTT Data: MQTT CONNECT)                    |
      |--------------------------------------------> |
      |    Broker determines MQTT CONNECT is invalid |
      |                                (QUIC REJECT) |
      | <------------------------------------------- |
      | QUIC Connection Aborted                       |
      | X---------------------------------------->X |
      | **Fix MQTT credentials**                      |
      | (0-RTT Data: MQTT CONNECT)                    |
      |--------------------------------------------> |
      |                   Broker accepts MQTT CONNECT |
      |                     (0-RTT ACK, MQTT CONNACK) |
      | <------------------------------------------- |
      |         PUBLISH, SUBSCRIBE, etc.              |
      | <------------------------------------------> |
```

Figure 2 Connection Handshake Examples

The need for implementing MQTT’s AUTH Property is alleviated by using QUIC. By default, CONNECT would use TLS: Anon, MQTT: None (RFC 9431,2.2.1) in order to not double dip on encryption.

2. Message Definition

A basic implementation of MoQ would leverage existing versions of MQTT, with 5 being the current highest version. Since MQTT has some features that overlap with the capabilities of QUIC, an alternative version of MQTT would be proposed to take greater advantage of QUIC’s infrastructure.

2.1. Payload Considerations

QUIC implements congestion control which adjusts frame sizes on the fly to optimize stream speed. MQTT does not have a defined payload size, rather it is all the remaining bytes after the variable header. Because each topic is assigned its own QUIC stream, MQTT payloads which do not fit a single QUIC frame can be split to fit multiple, and reassembled by the subscriber. In this case the Packet Identifier would indicate to the application than a publication spanned multiple packets. To do this a QoS level of 1 or 2 is required.

2.2. Datatype Matching Consideration

MQTT does not enforce any firm rules on data structures, rather it is up to the subscribing applications to know how to deserialize the data appropriately. A common approach is to encode data in JSON structures which offers a level of human readability.

2.2.1. Subscription Based IDL Request

An Interface Definition Language (IDL) file is used by remote procedure calls (RPC) to specify the interface between client and server. This approach can be leveraged here to provide details on the JSON schema or deserialization control over raw byte streams contained within the payload. This approach is also inspired by Real Time Innovation’s Data Distribution Service, with QoS enforcement over datatype matching often requiring IDLs to be available to subscribers.

The fixed header of the Control Stream is where commands for PUBLISH and SUBSCRIBE are handled, so this is a reasonable place for an TYPEREQ (Type Request) to exist. A TYPEREQ Control Packet would look very similar to a SUBSCRIBE, mainly including the topic name. The Publisher would immediately, upon receiving this request send a PUBTYPE packet containing the IDL for the topic. This adds a discovery-like capability for clients to get detailed information on topics that are being published.

2.3. Payload Data

A common approach to MQTT payloads is using JSON Key-Value pairs. An example of a thermostat publishing temperature data may look like:

```
{
“timestamp”: “2025-05-18T22:30:00Z”,
“temperature”: 72.2,
“tempUnit”: “Fahrenheit”,
“humidity”: “63.2”
}
```

When designing a smart system, logic can be embedded in different places to implement different control schemes. For example, in a home HVAC model, typically the thermostat signals the air handler to run the fan along with a heater or evaporator. An even more advanced system could include variable vane registers to control the airflow entering a room. All these devices could report status back to a central hub, but they could also be designed to report to their companion device.

Consider a distributed home HVAC control scheme where any thermostat can enable the air handler, but depending on the position of vents, the air handler adjusts the fan speed as to maintain correct pressure through the ducting. The example is shown below in figure 3. The Broker is removed for simplicity

```
   +============+   PUBLISH: currentTemp: 78.2 °F   +==============+
   |            | ---------------------------------> |              |
   | Thermostat |     PUBLISH: “state”:”AC RUNNING” | Air Handler  |
   |            | <--------------------------------- |              |
   +------------+                                   |              |
          | PUBLISH: currentTemp,                   |              |
          |          setPoint  +========+  OPEN 80% |              |
          +------------------> |  VENT  | --------> |              |
                               +--------+           +--------------+
```

Figure 3 Multiple Publishers and Subscribers with integrated logic

3. Deterministic Finite Automaton

In MQTT, there are Clients and a Broker which is responsible for tracking subscriptions and delivering publications to the respective subscribers.

After connecting to a broker, a client can either subscribe to topics and wait for data to arrive or it can publish data of its own. Depending on the QoS level, the state machine of the client changes. Shown below in figures 4, 5 and 6 are the primary mechanisms for publication delivery at the three QoS levels.

3.1. QoS 0: At Most Once

```
Publisher                      Broker                     Subscriber
    | PUBLISH (QoS = 0)          |                             |
    | -------------------------> |           PUBLISH           |
    | -----+                     | --------------------------> |
    |      | Delete Message      |                             |
    | <----+
```

Figure 4 Publish States at QoS 0

A client that is publishing data at a quick and steady interval might not care if all updates make it to every subscriber, they can simply catch the next publication if they miss one or a few. QoS 0 allows very little processing overhead, simply fire out the message to the broker and forget about it.

3.2. QoS 1: At Least Once

```
Publisher                      Broker                    Subscriber

    | -----+                    |                             |
    |      | Store Message      |                             |
    | <----+                    |                             |
    | PUBLISH (QoS = 1)         |                             |
    | ------------------------> |                             |
    |                           | -----+                      |
    |                           |      | Store Message        |
    |                           | <----+    PUBLSIH           |
    |                           | --------------------------> |
    |                           | -----+                      |
    |                           |      | Delete Message       |
    |         PUBACK            | <----+                      |
    | <------------------------ |                             |
    | -----+                    |                             |
    |      | Delete Message     |                             |
    | <----+                    |                             |
```

Figure 5 Publish States at QoS 1

The next level of QoS increases the reliability of message by storing the message for retransmit. After sending the message to the broker the client waits to receive a PUBACK as a confirmation that the Broker has Published the message to the respective subscribers.

3.3. QoS 2: Exactly Once

```
Publisher                      Broker                    Subscriber
    | -----+                     |                             |
    |      | Store Message       |                             |
    | <----+                     |                             |
    | PUBLISH (QoS = 2)          |                             |
    | -------------------------> |                             |
    |                            | -----+                      |
    |                            |      | Store Message        |
    |                            | <----+    PUBLSIH           |
    |          PUBREC            | --------------------------> |
    | <------------------------- |                             |
    |         PUBREL             |                             |
    | -------------------------> |                             |
    |                            | -----+                      |
    |                            |      | Delete Message       |
    |         PUBCOMP            | <----+                      |
    | <------------------------- |                             |
    | -----+                     |                             |
    |      | Delete Message      |                             |
    | <----+                     |                             |
```

Figure 6 Publish States at QoS 2

The highest QoS level, level two, adds one more level of confirmation between the Publisher and the Broker. After the Broker publishes to the Subscribers, a PUBREC is sent from the Broker back to the Publisher to notify that the message has been processed. Upon receiving that confirmation the client offers a PUBREL to allow the Broker to delete the message. The last step is for the Broker to send a PUBCOMP to the Publisher to notify that publication is complete.

4. Extensibility

MQTT implements a version identifier as a single-byte unsigned integer. MoQ supports MQTT version 5.0 (current latest version). This combination allows for subtle modifications of the base MQTT protocol, so MoQ specific version identifiers would take advantage of the same field. For the sake of backwards compatibility, a CONNECT packet that specifies MQTT version 5 or lower would force the application to follow the established MQTT protocol rules. MoQ specific versions of MQTT would start at the maximum version and assume MQTT will not go through greater than 250 more iterations.

The primary extensibility benefit is in opening the payload size. By implementing separate control and data streams, along with serialization of payload data to a QUIC stream allows for higher bandwidth of data within publications. It may not be particularly relevant to a simple home thermostat, but smart home security products would leverage this stream as a secure channel for video data alongside discrete status data while also receiving configuration from a central hub.

The paring down of features from MQTT that overlap with QUIC into a MoQ specific version is targeted to embedded implementations. The main target for MoQ are edge devices and embedded products with hardware limitations. Devices can benefit from the security and reliability while cutting back on communication overhead which ultimately gives time back to whatever data acquisition or processing was intended as the core competency of the device.

5. Security Considerations

MQTT uses a username and password for authentication during the CONNECT sequence. The MQTT packet is inside the encrypted portion of the QUIC frame even for 0-RTT, so the login credentials aren’t sent in the clear. Both the QUIC TLS handshake and the MQTT username and password must succeed before a connection can open, and if either fails then the connection is aborted as described in 1.2.1.

QUIC is subject to replay attacks, where the attacker copies the 0-RTT TLS Hello packet, and sends it to a server. Obtaining the MQTT username and password ends up being irrelevant because a connection can be established without knowing them.

There is a level of obscurity to MoQ that provides a level of security. Subscribing to a topic requires having the name of the topic. Similarly, making use of the published data is only useful if the datatype mapping is known. Even if a nefarious system were to connect to a MoQ broker, there isn't necessarily a way to ask for a list of topic publications especially ones using QoS 0. A future version could include a feature where Brokers keep a log of all the topic names running QoS > 0 and provide them to clients in a discovery service, but that could defeat the obscurity.

References

[RFC9431]	Sengul, C., "Message Queuing Telemetry Transport (MQTT) and Transport Layer Security (TLS) Profile of Authentication and Authorization for Constrained Environments (ACE) Framework", RFC 9431, DOI 10.17487/RFC9431, July 2023, <https://www.rfc-editor.org/rfc/rfc9431>
[RFC9000]	Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/rfc/rfc9000>