## Message Queuing Telemetry Transport (MQTT)

| CoAP | MQTT |
|---|---|
| UDP | TCP |
| IPv6 | |
| 6LoWPAN | |
| 802.15.4 MAC | |
| 802.15.4 PHY | |

**Example of a High-Level IoT Protocol Stack for CoAP and MQTT**

At the end of the 1990s, engineers from IBM and Arcom (acquired in 2006 by Eurotech) were looking for a reliable, lightweight, and cost-effective protocol to monitor and control a large number of sensors and their data from a central server location, as typically used by the oil and gas industries. Their research resulted in the development and implementation of the Message Queuing Telemetry Transport (MQTT) protocol that is now standardized by the Organization for the Advancement of Structured Information Standards (OASIS).

Considering the harsh environments in the oil and gas industries, an extremely simple protocol with only a few options was designed, with considerations for constrained nodes, unreliable WAN backhaul communications, and bandwidth constraints with variable latencies. These were some of the rationales for the selection of a client/server and publish/subscribe framework based on the TCP/IP architecture, as shown in Figure

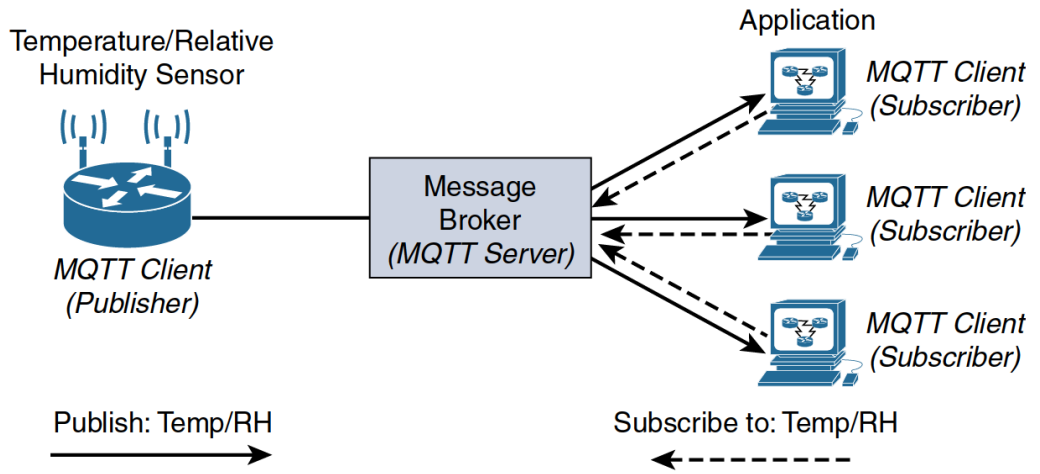


Fig: MQTT Publish/Subscribe Framework

An MQTT client can act as a publisher to send data (or resource information) to an MQTT server acting as an MQTT message broker. In the example illustrated in Figure the MQTT client on the left side is a temperature (Temp) and relative humidity (RH) sensor that publishes its Temp/RH data. The MQTT server (or message broker) accepts the network connection along with application messages, such as Temp/RH data, from the publishers. It also handles the subscription and unsubscription process and pushes the application data to MQTT clients acting as subscribers.
The application on the right side of Figure is an MQTT client that is a subscriber to the Temp/RH data being generated by the publisher or sensor on the left. This model, where subscribers express a desire to receive information from publishers, is well known.

MQTT control packets run over a TCP transport using port 1883. TCP ensures an ordered, lossless stream of bytes between the MQTT client and the MQTT server. MQTT is a lightweight protocol because each control packet consists of a 2-byte fixed header with optional variable header fields and optional payload.

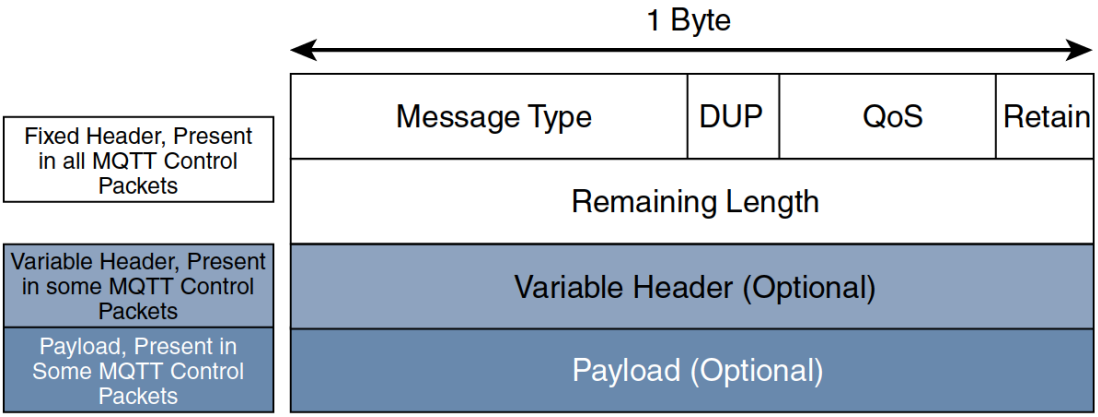


Fig: MQTT Message Format

The next field in the MQTT header is DUP (Duplication Flag). This flag, when set, allows the client to notate that the packet has been sent previously, but an acknowledgement was not received.
The QoS header field allows for the selection of three different QoS levels.
The next field is the Retain flag. Only found in a PUBLISH message the Retain flag notifies the server to hold onto the message data. This allows new sub- scribers to instantly receive the last known value without having to wait for the next update from the publisher.
The last mandatory field in the MQTT message header is Remaining Length. This field specifies the number of bytes in the MQTT packet following this field.
MQTT sessions between each client and server consist of four phases: session establishment, authentication, data exchange, and session termination. Each client connecting to a server has a unique client ID, which allows the identification of the MQTT session between both parties. When the server is delivering an application message to more than one client, each client is treated independently.

These are the three levels of MQTT QoS:

- QoS 0: This is a best-effort and unacknowledged data service referred to as “at most once” delivery. The publisher sends its message one time to a server, which transmits it once to the subscribers. No response is sent by the receiver, and no retry is per- formed by the sender. The message arrives at the receiver either once or not at all.
- QoS 1: This QoS level ensures that the message delivery between the publisher and server and then between the server and subscribers occurs at least once. In PUBLISH and PUBACK packets, a packet identifier is included in the variable header. If the message is not acknowledged by a PUBACK packet, it is sent again. This level guarantees “at least once” delivery.
- QoS 2: This is the highest QoS level, used when neither loss nor duplication of messages is acceptable. There is an increased overhead associated with this QoS level because each packet contains an optional variable header with a packet identifier. Confirming the receipt of a PUBLISH message requires a two-step acknowledgement process. The first step is done through the PUBLISH/PUBREC packet pair, and the second is achieved with the PUBREL/PUBCOMP packet pair. This level provides a “guaranteed service” known as “exactly once” delivery, with no consideration for the number of retries as long as the message is delivered once.

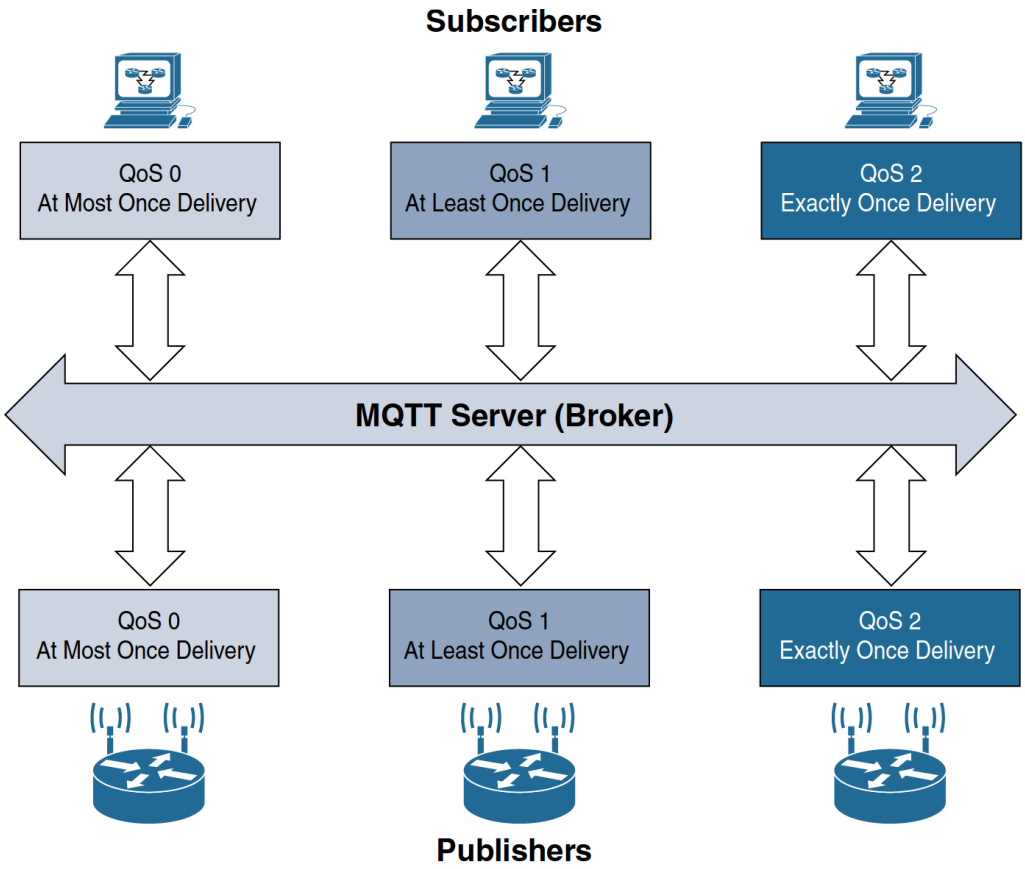


Fig: MQTT QoS Flows

## Constrained Application Protocol (CoAP)

The CoAP framework defines simple and flexible ways to manipulate sensors and actuators for data or device management. The IETF CoRE working group has published multiple standards-track specifications for CoAP, including the following:

- **RFC 6690:** Constrained RESTful Environments (CoRE) Link Format
- **RFC 7252:** The Constrained Application Protocol (CoAP)
- **RFC 7641:** Observing Resources in the Constrained Application Protocol (CoAP)
- **RFC 7959:** Block-Wise Transfers in the Constrained Application Protocol (CoAP)
- **RFC 8075:** Guidelines for Mapping Implementations: HTTP to the Constrained Application Protocol (CoAP)

The CoAP messaging model is primarily designed to facilitate the exchange of messages over UDP between endpoints, including the secure transport protocol Datagram Transport Layer Security (DTLS). (UDP is discussed earlier in this chapter.) The IETF CoRE working group is studying alternate transport mechanisms, including TCP, secure TLS, and WebSocket. CoAP over Short Message Service (SMS) as defined in Open Mobile Alliance for Lightweight Machine-to-Machine (LWM2M) for IoT device management is also being considered.

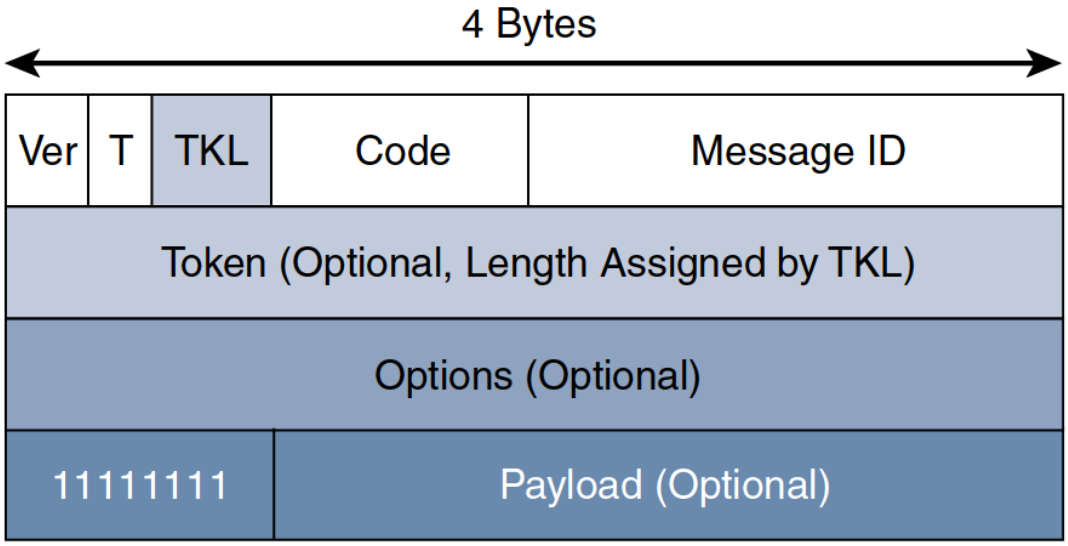


Fig: CoAP Message Format

**Table 6-1** *CoAP Message Fields*

| CoAP Message Field | Description |
|---|---|
| Ver (Version) | Identifies the CoAP version. |
| T (Type) | Defines one of the following four message types: Confirmable (CON), Non-confirmable (NON), Acknowledgement (ACK), or Reset (RST). CON and ACK are highlighted in more detail in Figure 6-9. |
| TKL (Token Length) | Specifies the size (0–8 Bytes) of the Token field. |
| Code | Indicates the request method for a request message and a response code for a response message. For example, in Figure 6-9, GET is the request method, and 2.05 is the response code. For a complete list of values for this field, refer to RFC 7252. |
| Message ID | Detects message duplication and used to match ACK and RST message types to Con and NON message types. |
| Token | With a length specified by TKL, correlates requests and responses. |
| Options | Specifies option number, length, and option value. Capabilities provided by the Options field include specifying the target resource of a request and proxy functions. |
| Payload | Carries the CoAP application data. This field is optional, but when it is present, a single byte of all 1s (0xFF) precedes the payload. The purpose of this byte is to delineate the end of the Options field and the beginning of Payload. |

CoAP can run over IPv4 or IPv6. However, it is recommended that the message fit within a single IP packet and UDP payload to avoid fragmentation. For IPv6, with the default MTU size being 1280 bytes and allowing for no fragmentation across nodes, the maximum CoAP message size could be up to 1152 bytes, including 1024 bytes for the payload.

CoAP defines four types of messages: confirmable, non-confirmable, acknowledge- ment, and reset. Method codes and response codes included in some of these messages make them carry requests or responses. CoAP code, method and response codes, option numbers, and content format have been assigned by IANA as Constrained RESTful Environments (CoRE) parameters.

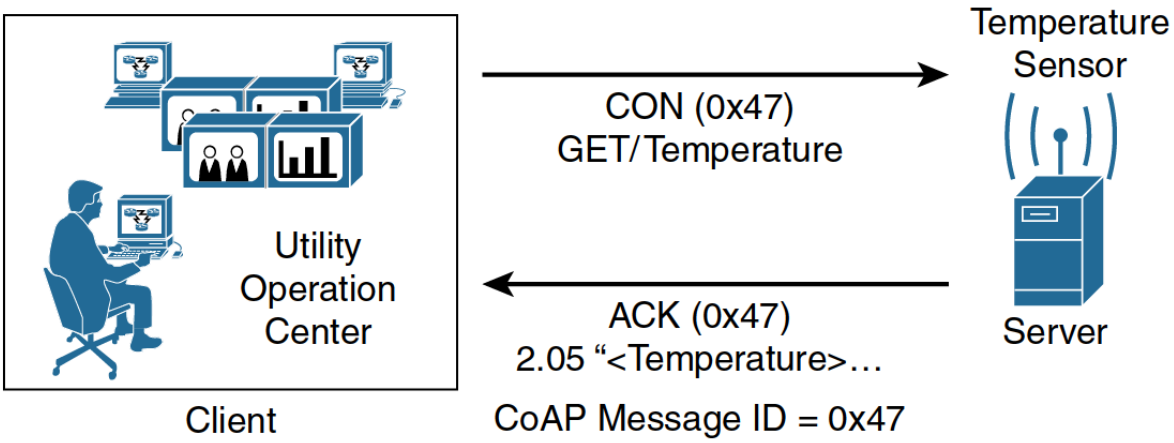


Fig:CoAP Reliable Transmission Example

Figure shows a utility operations center on the left, acting as the CoAP client, with the CoAP server being a temperature sensor on the right of the figure. The communication between the client and server uses a CoAP message ID of 0x47. The CoAP Message ID ensures reliability and is used to detect duplicate messages.

The client in Figure sends a GET message to get the temperature from the sensor.

Notice that the 0x47 message ID is present for this GET message and that the message is also marked with CON. A CON, or confirmable, marking in a CoAP message means the message will be retransmitted until the recipient sends an acknowledgement (or ACK) with the same message ID.

In Figure the temperature sensor does reply with an ACK message referencing the correct message ID of 0x47. In addition, this ACK message piggybacks a successful response to the GET request itself. This is indicated by the 2.05 response code followed by the requested data.