What is MQTT?

https://www.paessler.com/it-explained/mqtt

MQTT stands for Message Queuing Telemetry Transport. It is a lightweight messaging protocol for use in cases where clients need a small code footprint and are connected to unreliable networks or networks with limited bandwidth resources. It is primarily used for machine-to-machine (M2M) communication or Internet of Things types of connections.

(Vedi anche: https://en.wikipedia.org/wiki/MQTT)

History

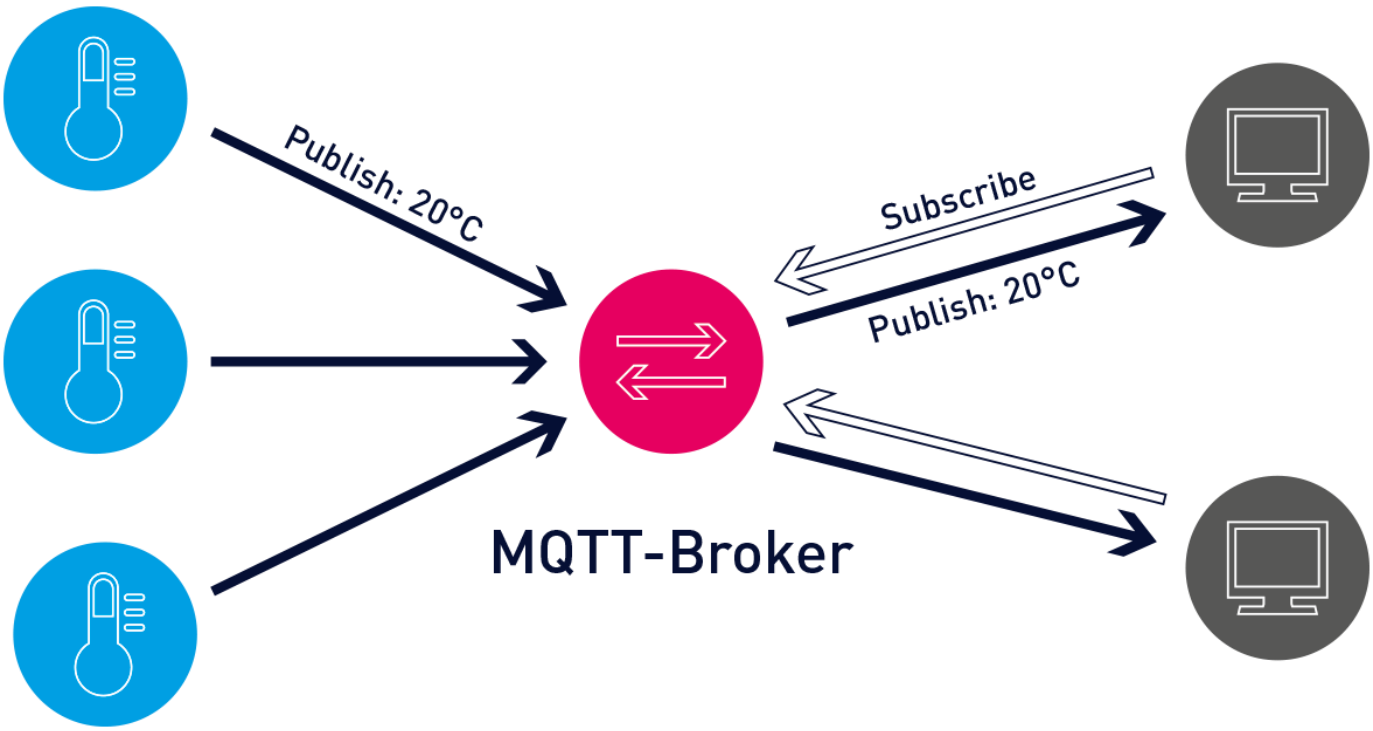
MQTT was originally created by Dr. Andy Stanford-Clark and Arlen Nipper in 1999. The original purpose of the communication method was to allow monitoring devices used in the oil and gas industry to send their data to remote servers. In many cases, such monitoring devices were used in remote locations where any sort of landline, wired connection, or radio transmission connection would be difficult, or impossible, to establish. At that time, the only option for such cases was satellite communications, which were very expensive and billed based on how much data was used. With thousands of sensors in the field, the industry needed a form of communication that could provide data reliably enough for use, while using minimal bandwidth.

MQTT was standardized as open source under the Organization for the Advancement of Structured Information Standards (OASIS) in 2013. OASIS still manages the MQTT standard.

MQTT architecture

MQTT runs on top of TCP/IP using a PUSH/SUBSCRIBE topology. In MQTT architecture, there are two types of systems: clients and brokers. A broker is the server that the clients communicate with. The broker receives communications from clients and sends those communications on to other clients. Clients do not communicate directly with each other, but rather connect to the broker. Each client may be either a publisher, a subscriber, or both.

MQTT is an event-driven protocol. There is no periodic or ongoing data transmission. This keeps transmission to a minimum. A client only publishes when there is information to be sent, and a broker only sends out information to subscribers when new data arrives.



Message architecture

Another way MQTT minimizes its transmissions is with a tightly defined, small message construction. Each message has a fixed header of just 2 bytes. An optional header may be used but increases the size of the message. The message payload is limited to just 256 MB. Three different Quality of Service (QoS) levels allow network designers to choose between minimizing data transmission and maximizing reliability.

- **QoS 0** – Offers the minimum amount of data transmission. With this level, each message is delivered to a subscriber once with no confirmation. There is no way to know if subscribers received the message. This method is sometimes referred to as “fire and forget” or as “at most once delivery.” Because this level assumes that delivery is complete, messages are not stored for delivery to disconnected clients that later reconnect.
- **QoS 1** – The broker attempts to deliver the message and then waits for a confirmation response from the subscriber. If a confirmation is not received within a specified time frame, the message is sent again. Using this

method, the subscriber may receive the message more than once if the broker does not receive the subscriber's acknowledgment in time. This is sometimes referred to as "at least once delivery".

- **QoS 2** – The client and broker use a four-step handshake to ensure that the message is received, and that it is received only once. This is sometimes referred to as "exactly once delivery".

For situations where communications are reliable but limited, QoS 0 may be the best option. For situations where communications are unreliable, but where the connections are not as resource limited, then QoS 2 would be the best option. QoS 1 provides a sort of best-of-both-worlds solution but requires that the application receiving the data knows how to handle duplicates.

For both QoS 1 and QoS 2, messages are saved or queued for clients that are offline and that have an established persistent session. These messages are resent (according to the appropriate QoS level) once the client is back online.

Topics

Messages within MQTT are published as *topics*. Topics are structures in a hierarchy using the slash (/) character as delimiter. This structure resembles that of a directory tree on a computer file system. A structure such as *sensors/OilandGas/Pressure/* allows a subscriber to specify that it should only be sent data from clients that publish to the *Pressure* topic, or for a broader view, perhaps all data from clients that publish to any *sensors/OilandGas* topic. Topics are not explicitly created in MQTT. If a broker receives data published to a topic that does not currently exist, the topic is simply created, and clients may subscribe to the new topic.

Retained messages

To keep the footprint small, received messages are not stored on the broker unless they are marked with the retained flag. This is called a retained message. Users who wish to store received messages will need to store them elsewhere outside of the MQTT protocol. There is one exception.

As an event-driven protocol, it is possible, even likely, that a subscriber may receive very few messages for a given topic, even over a long period of time. In the previously mentioned topic structure, perhaps messages to the *Pressure* topic are only sent when a sensor detects that the pressure has exceeded a certain amount. Assuming that whatever that sensor is monitoring does not fail often, it could be months, or even years before a client publishes a message to that topic.

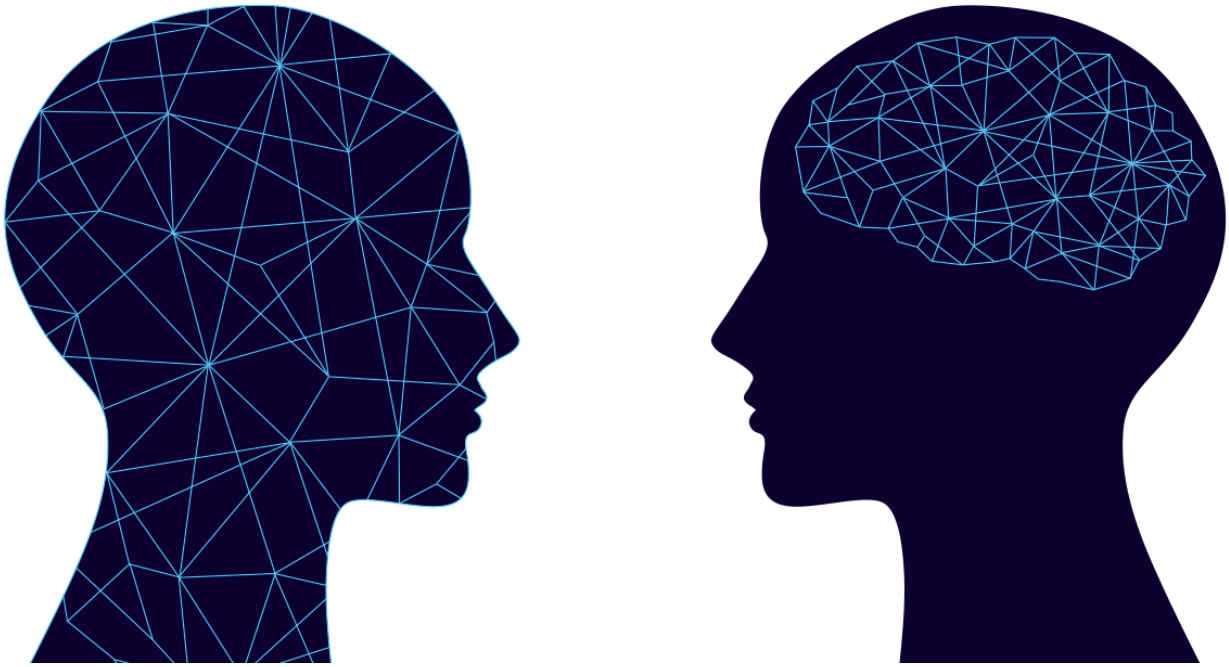
To ensure that a new subscriber receives the messages from a topic, brokers may keep the last message sent to each topic. This is called a *retained message*. Whenever a new client subscribes to a topic or when an existing client comes back online, the retained message is sent to the subscribers, thereby ensuring that the subscription is active, and that it has the latest information.

Last will and testament

Where communications are unreliable, it is possible that a publisher will disconnect from the network without warning. A publisher may register a message to be sent to subscribers in the event the publisher disconnects unexpectedly, this is a so called *last will and testament*. This message is cached on the broker and sent to subscribers should the publisher disconnect improperly. Typically, such a message includes information allowing the disconnected publisher to be identified so that the appropriate actions can be taken.

MQTT messages

- **Connect:** The client has to establish a connection to the broker by sending a connect packet (With username and password if needed)
- **Publish** – Sends a block of data containing the message to be sent. This data is specific to each implementation but could be something as simple as an on/off indication, or a value of a certain sensor, such as temperature, pressure, etc. In the case that the topic being published to does not exist, the topic is created on the broker.
- **Subscribe** – Turns a client into a subscriber of a topic. Topics may be subscribed to specifically or via wildcards that allow subscriptions to an entire topic branch or part of any topic branch. To subscribe, a client sends a SUBSCRIBE packet and receives a SUBACK packet in return. If there is a retained message for the topic, the new subscriber receives that message as well.
- **PING** – A client may ping the broker. A PINGREQ packet is sent by the subscriber and a reply PINGRESP packet is sent in reply. Pings may be used to ensure that the connection is still working and that the TCP session was not unexpectedly closed by other networking equipment like a router or gateway.
- **DISCONNECT** – A subscriber or publisher may send a DISCONNECT message to the broker. This message informs the broker that it will no longer need to send or queue messages for a subscriber and that it will no longer receive data from a publisher. This kind of shutdown allows the client to reconnect using the same client identity as before. When a client disconnects without sending a disconnect message, its last will and testament is sent to subscribers



Security

The original goal of the MQTT protocol was to make the smallest and most efficient data transmission possible over expensive, unreliable communication lines. As such, security was not a primary concern during the design and implementation of MQTT.

However, there are some security options available at the cost of more data transmission and a larger footprint.

- **Network security** – If the network itself can be secured, then the transmission of insecure MQTT data is irrelevant. In this case, security issues would have to occur from inside the network itself, perhaps via a bad actor or another form of network penetration.
- **Username and password** – MQTT does allow usernames and passwords for a client to establish a connection with a broker. Unfortunately, in order to keep the overhead light, the usernames and passwords are transmitted in clear text. In 1999, this was more than sufficient because intercepting a satellite communication for what was essentially an unimportant sensor reading would have been prohibitively difficult. However, today, intercepting many types of wireless network communications is trivial, making such authentication all but useless. Many use cases require a username and password not as protection against bad faith actors, but as a way to avoid unintentional connections.
- **SSL/TLS –** Running on top of TCP/IP, the obvious solution for securing transmissions between clients and brokers is the implementation of SSL/TLS. Unfortunately, this adds substantial overhead to the otherwise lightweight communications.