lamaPLC Communication: MQTT

MQTT (originally an initialism of **MQ Telemetry Transport**) is a lightweight, publish-subscribe, machine to machine network protocol for message queue/message queuing service. It is designed for connections with remote locations that have devices with resource constraints or limited network bandwidth, such as in the Internet of Things (IoT).

It must run over a transport protocol that provides ordered, lossless, bi-directional connections—typically, TCP/IP, but also possibly over QUIC. It is an open OASIS standard and an ISO recommendation (ISO/IEC 20922).

Andy Stanford-Clark (IBM) and *Arlen Nipper* (then working for Eurotech, Inc.) authored the first version of the protocol in 1999. It was used to monitor oil pipelines within the SCADA industrial control system. The goal was to have a protocol that is bandwidth-efficient, lightweight and uses little battery power, because the devices were connected via satellite link which, at that time, was extremely expensive.

Historically, the "**MQ**" in "**MQTT**" came from the IBM MQ (then '*MQSeries*') product line, where it stands for *"Message Queue"*. However, the protocol provides publish-and-subscribe messaging (no queues, in spite of the name). In the specification opened by IBM as version 3.1 the protocol was referred to as *"MQ Telemetry Transport"*. Subsequent versions released by OASIS strictly refer to the protocol as just "MQTT", although the technical committee itself is named "OASIS Message Queuing Telemetry Transport Technical Committee". Since 2013, "MQTT" does not stand for anything.

In 2013, IBM submitted MQTT v3.1 to the OASIS specification body with a charter that ensured only minor changes to the specification could be accepted. After taking over maintenance of the standard from IBM, OASIS released version 3.1.1 on October 29, 2014. A more substantial upgrade to MQTT version 5, adding several new features,[14] was released on March 7, 2019.

MQTT-SN (MQTT for Sensor Networks) is a variation of the main protocol aimed at battery-powered embedded devices on non-TCP/IP networks, such as Zigbee.

The MQTT protocol defines two types of network entities: a **message broker** and a number of **clients**. An MQTT broker is a server that receives all messages from the clients and then routes the messages to the appropriate destination clients. An MQTT client is any device (from a micro controller up to a fully-fledged server) that runs an MQTT library and connects to an MQTT broker over a network.

Information is organized in a hierarchy of topics. When a publisher has a new item of data to distribute, it sends a control message with the data to the connected broker. The broker then distributes the information to any clients that have subscribed to that topic. The publisher does not need to have any data on the number or locations of subscribers, and subscribers, in turn, do not have to be configured with any data about the publishers.

If a broker receives a message on a topic for which there are no current subscribers, the broker discards the message unless the publisher of the message designated the message as a retained message. A retained message is a normal MQTT message with the retained flag set to true. The broker stores the last retained message and the corresponding *quality of service* (**QoS**) for the selected topic. Each client that subscribes to a topic pattern that matches the topic of the retained

message receives the retained message immediately after they subscribe. The broker stores only one retained message per topic. This allows new subscribers to a topic to receive the most current value rather than waiting for the next update from a publisher.

When a publishing client first connects to the broker, it can set up a default message to be sent to subscribers if the broker detects that the publishing client has unexpectedly disconnected from the broker.

MQTT relies on the TCP protocol for data transmission. A variant, MQTT-SN, is used over other transports such as UDP or Bluetooth.

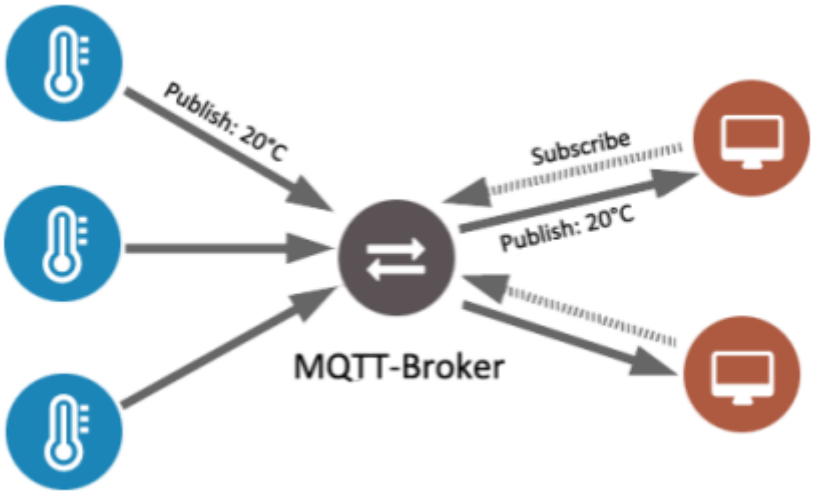
MQTT sends connection credentials in plain text format and does not include any measures for security or authentication. This can be provided by using TLS to encrypt and protect the transferred information against interception, modification or forgery.

The TCP default **unencrypted MQTT port is 1883**. The TCP **encrypted port is 8883**.

MQTT broker

The MQTT broker is a piece of software running on a computer (running on-premises or in the cloud), and could be self-built or hosted by a third party. It is available in both open source and proprietary implementations.

The broker acts as a post office. MQTT clients don't use a direct connection address of the intended recipient, but use the subject line called “Topic”. Anyone who subscribes receives a copy of all messages for that topic. Multiple clients can subscribe to a topic from a single broker (one to many capability), and a single client can register subscriptions to topics with multiple brokers (many to one).



Each client can both produce and receive data by both publishing and subscribing, i.e. the devices can publish sensor data and still be able to receive the configuration information or control commands (MQTT is a bi-directional communication protocol). This helps in both sharing data, managing and controlling devices. A client cannot broadcast the same data to a range of topics, and must publish multiple messages to the broker, each with a single topic given.

With the MQTT broker architecture, the client devices and server application become decoupled. In this way, the clients are kept unaware of each other's information. MQTT, if configured to, can use TLS encryption with certificate, username and password protected connections. Optionally, the connection may require certification, in the form of a certificate file that a client provides and must match with the server's copy.ű

Eclipse Mosquitto MQTT broker

Eclipse Mosquitto is an open source (EPL/EDL licensed) message broker that implements the MQTT protocol versions 5.0, 3.1.1 and 3.1. Mosquitto is lightweight and is suitable for use on all devices from low power single board computers to full servers.

The MQTT protocol provides a lightweight method of carrying out messaging using a publish/subscribe model. This makes it suitable for Internet of Things messaging such as with low power sensors or mobile devices such as phones, embedded computers or microcontrollers.

The Mosquitto project also provides a C library for implementing MQTT clients, and the very popular *mosquitto_pub* and *mosquitto_sub* command line MQTT clients.

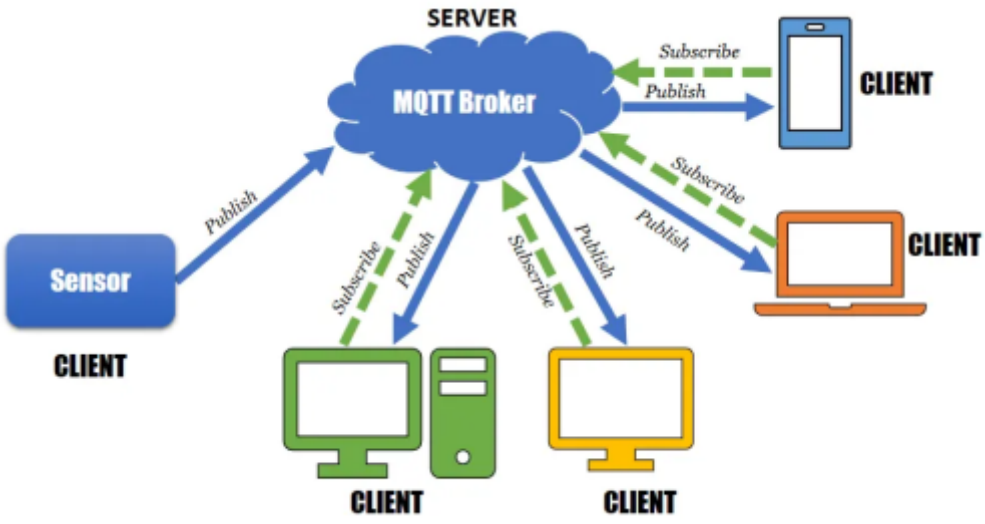
Other MQTT server / broker programs

List of other MQTT server / broker programs: https://mqtt.org/software/

MQTT client

In Client-Server architecture, a client connects to a server for the use of a service.

In MQTT context, an MQTT client is a device that connects to an MQTT broker over a network. The service provided by the MQTT broker (server) is the possibility to publish and/or subscribe on one or many topics. In MQTT, a client can be a publisher and subscriber or both.



With this message, a client presents itself to a broker providing the following main information:

ClientID

ClientID is an unique ID used by brokers to identify the client and store information (called session) about it. An empty ClientID means an *“anonymous”* connection: therefore, the broker does not memorize any information about the client.

CleanSession

If the CleanSession is set to false and the broker has information stored for that client, broker uses the existing session and delivers previously queued messages to the client. Instead, if the flag is set to true, it means discarding all existing sessions and messages for that client (mandatory if the ClientId is empty).

KeepAlive

This interval, express in seconds, defines the maximum period of time that broker and client can remain in contact without sending a message. The client needs to send regular PING messages, within the KeepAlive period, to the broker to maintain the connection alive.

Username and Password (optional)

Client can send a username and password to improve communication security.

WillMessage (optional)

A client can specify its last will message in the form of a MQTT message and topic. Broker will send this message, on behalf of the client, when the client “badly” disconnects.

List of MQTT clients: https://mqtt.org/software/

Arduino Client for MQTT

Arduino Client for MQTT: https://github.com/knolleary/pubsubclient
Arduino Sending Data over MQTT:
https://docs.arduino.cc/tutorials/uno-wifi-rev2/uno-wifi-r2-mqtt-device-to-device/

Message types

- **Connect:** Waits for a connection to be established with the server and creates a link between the nodes.

- **Disconnect:** Waits for the MQTT client to finish any work it must do, and for the TCP/IP session to disconnect.
- **Publish:** Returns immediately to the application thread after passing the request to the MQTT client.

QoS: Quality Of Service

QoS level is an agreement between sender and receiver on the guarantee of delivering a message.

MQTT protocol manages the message retransmission and guarantees delivery, making communication in unreliable networks a bit less complex.

There are 3 levels:

1. **At most once:** No guarantee of delivery. This level is called “fire and forget”. Use it when you have a stable communication channel and when the loss of messages is acceptable.
2. **At least once:** Guarantees that a message is delivered at least one time to the receiver. Use it when clients can tolerate duplicate messages. It’s the most used.
3. **Exactly once:** Guarantees that message is received only once by the receiver. Use it when your application is critical and you cannot tolerate loss and duplicate messages.

Version 5.0

In 2019, OASIS released the official MQTT 5.0 standard. Version 5.0 includes the following major new features:

- **Reason codes:** Acknowledgements now support return codes, which provide a reason for a failure.
- **Shared subscriptions:** Allow the load to be balanced across clients and thus reduce the risk of load problems
- **Message expiry:** Messages can include an expiry date and are deleted if they are not delivered within this time period.
- **Topic alias:** The name of a topic can be replaced with a single number

MQTT 5.0 also supports MQTT connections over the QUIC transport protocol. MQTT over QUIC offers improved performance by reducing the number of exchanges during the connection process, reducing overall latency, and offering better handling of network congestion and switching.

Sources

Wikipedia (here)

MQTT topics on lamaPLC

Page	Date	Tags

• lamaPLC Communication: MQTT	2024/11/15 21:28	communication, mqtt, iot, tcp, udp, bluetooth, tcp ip, quic, mqtt-sn, oasis, message broker, mosquitto, qos, arduino, iso iec 20922
• lamaPLC Communication: OPC	2024/11/15 21:33	communication, opc, scada, ole, net, xml, tcp, hmi, server, opc ua, opc ua client, mqtt, json, dcom, simatic, erp
• lamaPLC Communication: TCP / UDP Basic	2025/11/20 22:49	communication, bus, tcp, udp, tcp ip, darpa, ietf, smtp, http, https, ftp, ftps ssh, pop3, imap, mysql, cpanel, whm, ssl, webmail, mqtt, ethernet, ip
• lamaPLC: ESP32 / ESP8266	2025/11/22 00:07	esp8266, esp32, esp32-c2, esp32-c3, esp32-c5, esp32-c6, esp32-c61, esp32-h2, esp32-s2, esp32-s3, esp32-p4, espressif systems, communication, ethernet, ip, wi-fi, thread, zigbee, matter, homekit, bluetooth, mqtt, adc, spi, uart, i2c, i2s, rmt, pwm, usb, usb otg, twai

communication, MQTT, IoT, TCP, UDP, Bluetooth, TCP/IP, QUIC, MQTT-SN, OASIS, message broker, Mosquitto, QoS, Arduino, ISO/IEC 20922