Introduction to MQTT and WebSockets

What is MQTT and how does it work?

MQTT stands for Message Queuing Telemetry Transport. It is a lightweight communication protocol.
According to the official MQTT v3.1 documentation:
MQTT is a Client-Server publish/subscribe messaging transport protocol. It is lightweight, open, simple, and designed to be easy to implement. These characteristics make it ideal for use in many situations, including constrained environments for communication in Machine to Machine (M2M) and Internet of Things (IoT) contexts where a small code footprint is required, and network bandwidth is at a premium.
This is a clear and clean definition of the MQTT protocol in just a few lines. It is a messaging protocol designed for easy implementation, primarily client side. It is an open and lightweight communication protocol with minimal packet overhead. It is generally used for communication between two or more devices.

Message Queuing Telemetry Transport (MQTT) is a pure IoT protocol. It was designed in 1999 by Andy Stanford-Clark (IBM) and Arlen Nipper (Arcom, now Cirrus Link) specifically to address communication between machines - Machine to Machine (M2M).

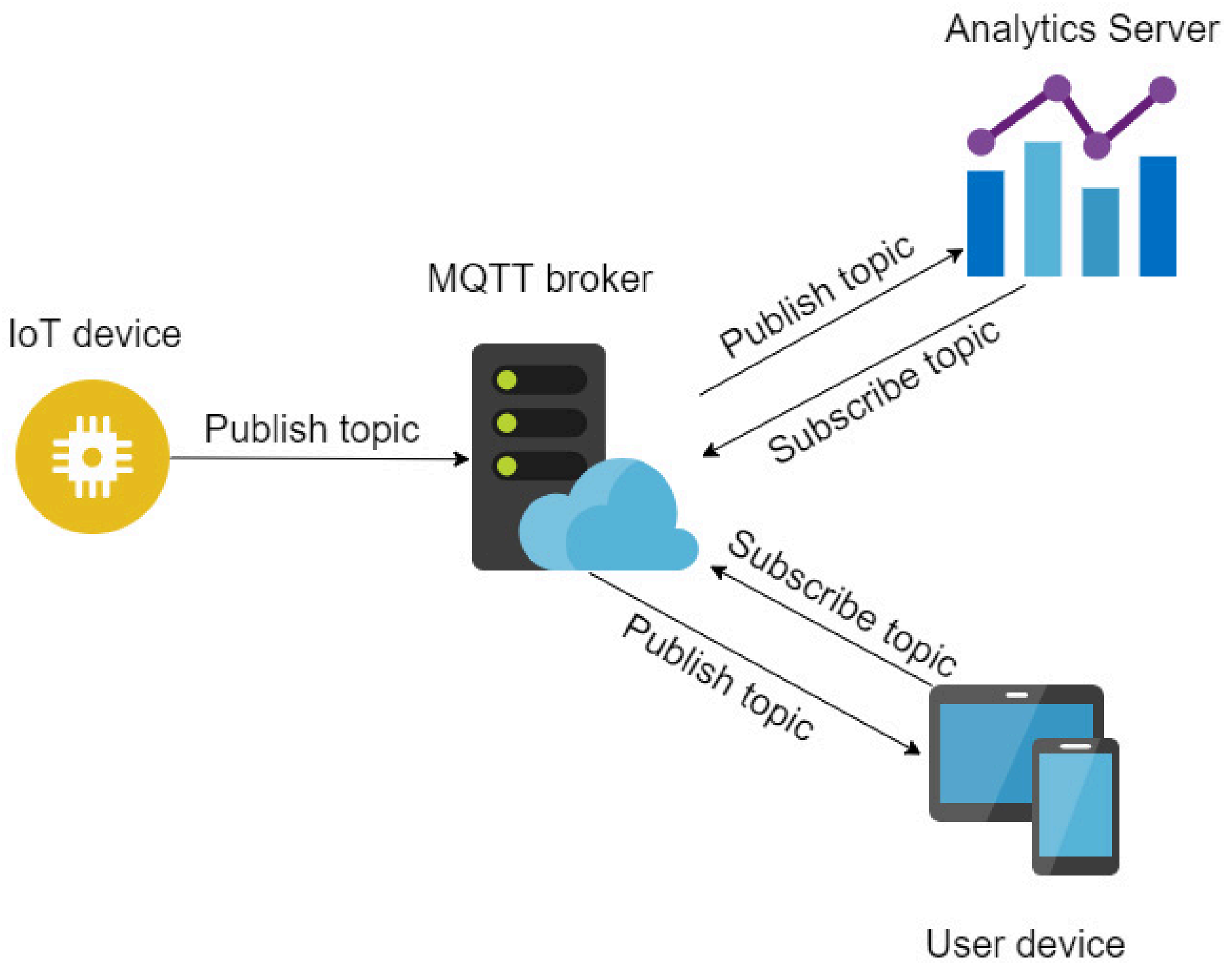
You can find MQTT in a wide range of applications, such as industrial, health, logistics, mobile, connected cars, smart buildings, and smart cities.

MQTT is based on short messages with a publish-subscribe mechanism.
Let’s see some characteristics of MQTT:

- Lightweight and bandwidth-efficient
- Design adapted to constrained devices
- Low power consumption
- Data agnostic
- Management of sessions
- Quality of Service (QoS) implementation
- SSL/TLS encryption
- User and client ID management

The basic idea of MQTT is to exchange data between clients. Each client connects to a server running an MQTT service (broker).
A client can send data to the broker (publish) or receive from it (subscribe).
In this architecture, clients are independent entities. All the information is exchanged between clients and the broker.
From a functionality point of view, it’s the same with email clients and servers.

Each client has access to certain information through the use of topics. These topics are specific fields in the data structure of the MQTT broker.
You can see an example of the MQTT architecture in the following figure:

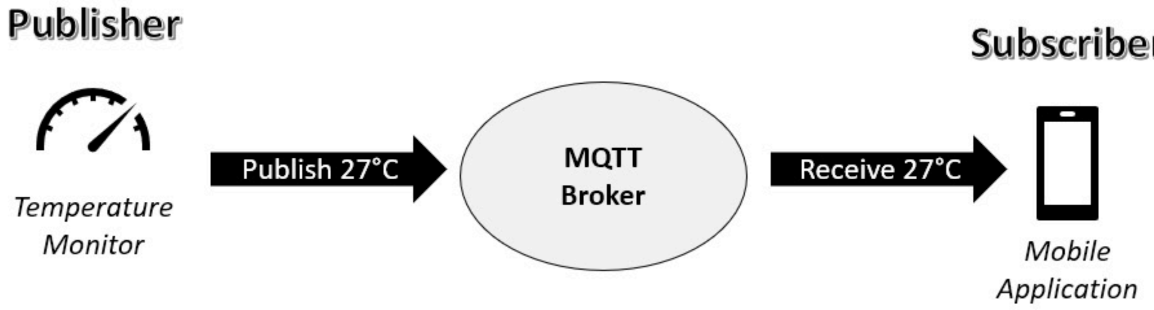


Basic concepts of MQTT

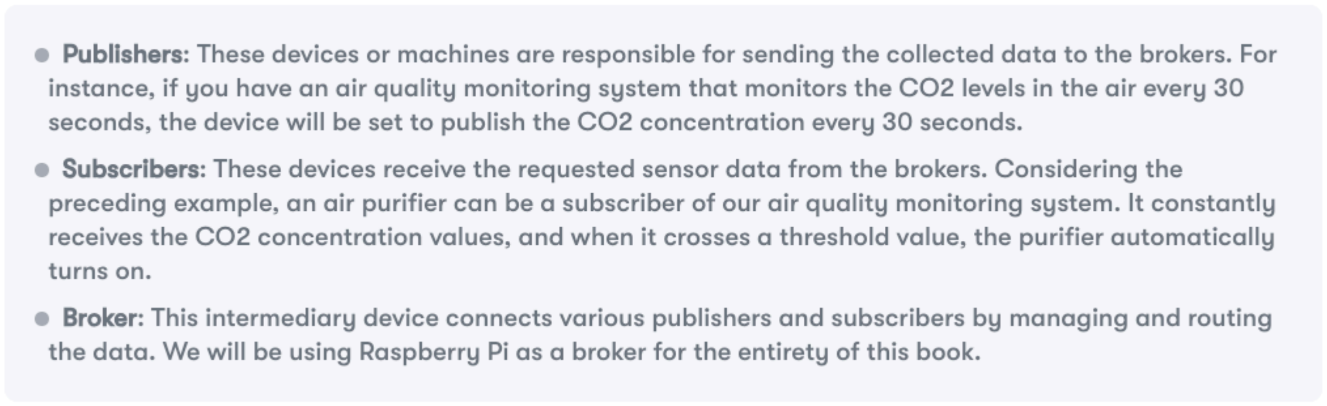
What exactly is a publish/subscribe protocol?

The publish/subscribe protocol is an alternative to the traditional client-server architecture. It means that instead of categorizing both sending and receiving machines as clients, the clients who send a message are publishers and the clients who receive the messages are the subscribers.
Another essential feature of such protocols is the decoupling between the clients. In simple words, the clients never directly communicate with each other. They are mediated by the third component of this system, known as the broker. In this book, we will be using our Raspberry Pi as an MQTT broker, which connects different client devices within a local network. The primary

function of the broker is to mediate and manage all communications between the various clients (i.e., publishers and subscribers).
To better understand how the whole system works, please see the figure below, which shows how the communication protocol operates with a very simplified diagram. In this example, the publishing client is a temperature monitor and the subscribing device is a mobile phone:



Please note that this is just a simplified representation. There can be multiple publishers and subscribers connected to a single broker. As you can see, the temperature monitor sends the current temperature value of 27°C through the MQTT communication protocol, which is then received by the MQTT broker, which routes it to the subscriber, a mobile application in our case.



- **Publishers:** These devices or machines are responsible for sending the collected data to the brokers. For instance, if you have an air quality monitoring system that monitors the CO2 levels in the air every 30 seconds, the device will be set to publish the CO2 concentration every 30 seconds.
- **Subscribers:** These devices receive the requested sensor data from the brokers. Considering the preceding example, an air purifier can be a subscriber of our air quality monitoring system. It constantly receives the CO2 concentration values, and when it crosses a threshold value, the purifier automatically turns on.
- **Broker:** This intermediary device connects various publishers and subscribers by managing and routing the data. We will be using Raspberry Pi as a broker for the entirety of this book.

Please note that both the publishers and subscribers are referred to as clients. A client can be a publisher, subscriber, or both as both these processes are entirely independent of each other.
But another question arises now: **how does the broker manage or route which information is sent where?**

Functionality and components of MQTT

MQTT has no client device addresses or identifiers, making it easy to build an expansible, ad hoc network. The only thing all clients must know is the address of the

broker. So, how do messages get routed between the clients? The solution for this is topics and messages.
This is how the whole system works:

First, the publisher sends the data collected to the broker on a particular topic, which is similar to a channel for data transmission and reception. Please note that a topic can have several subtopics too. For example, in an application where you send the temperature data from a sensor connected to your fridge, the topic will look something like this:

Kitchen/Fridge/

The main topic is the kitchen, and the appliance is the subtopic. The message will be **Temperature:14** on the given topic.

1. The subscribers listen to the topic. So, if the subscriber is listening to the Kitchen topic, it will have access to all the subtopics that are a part of this topic.
2. The primary function of the broker is to manage all the available topics and route the information according to the type of client, namely publishers and subscribers.

MQTT clients are tiny, and they require minimal resources to operate. So, even microcontrollers such as ESP8266 can be used as a client as long as they have an active connection to a network.
This protocol is highly efficient thanks to the small message headers that provide maximum network bandwidth efficiency.

Bidirectional communication protocol: MQTT allows to-and-fro messaging capability. This means a device can be a publisher and a subscriber simultaneously. This also allows easy broadcasting of messages to several devices at once.

Highly scalable: There is no worry about maintaining clients' addresses or IDs; it is effortless to expand the MQTT network. Moreover, the decoupling between the publishers and subscribers makes things even more accessible. The only things required on the client side are the broker's IP address and the topic name.

Reliability: MQTT is highly reliable when it comes to message delivery. As this is an essential aspect of any communication protocol, MQTT comes with three predefined quality of service (QoS) levels:

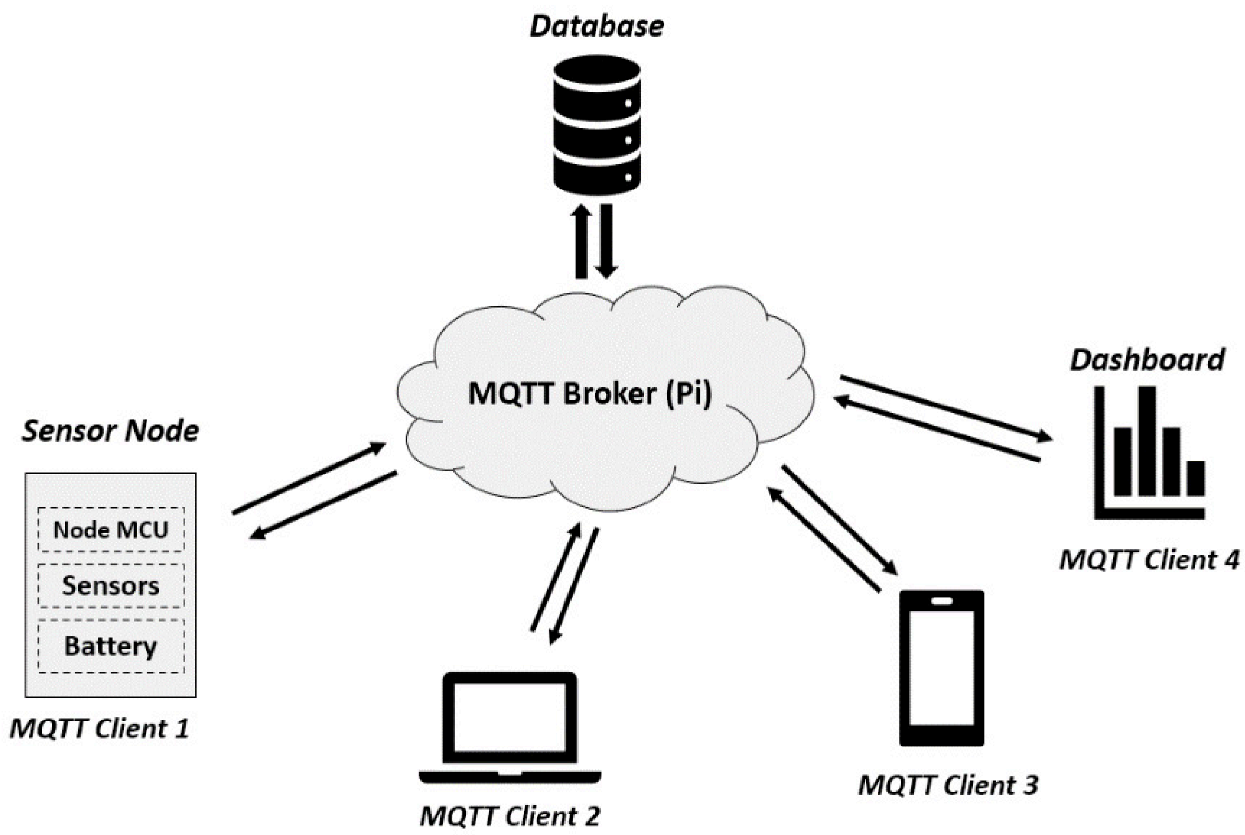
- QoS 0: At most once
- QoS 1: At least once
- QoS 2: Exactly once

Many IoT devices are connected over unreliable networks, and MQTT's support for persistent sessions reduces the client's time with the broker. For example, several monitoring devices are deployed on moving vehicles or in remote areas such as forests.

Highly secure: MQTT makes it easy to encrypt messages using TLS and authenticate clients using modern authentication protocols, such as OAuth.

MQTT messages

MQTT communication flow consists of a client (which can be a publisher or subscriber and in certain instances, both) and the broker, which manages the flow of all information across different clients. The following diagram provides an overview of how the MQTT message flow works:



MQTT stands for Message Queuing Telemetry Transport. Simply put, it is a communication protocol designed for constrained devices with network limitations. It is designed as a lightweight publish/subscribe messaging protocol. But what does this mean? For this, we need to be familiar with the concepts of messages, topics, clients, and brokers. Let's cover each and how they work

A message is a term given to the data that's shared between different MQTT clients. It can be some text, sensor readings, and so on.

MQTT topics

Topics are one of the essential components of this protocol. They provide you with a unique address for where your message should go. An MQTT topic is a series of strings separated by forward slashes. Each string before a forward slash indicates a new topic level. This gives you a lot of options for unique topics. Here is an example of a topic:

```
Bedroom/Lighting/Lamp
```

In this example, the topic is interpreted as follows: under the main topic, Bedroom, there is a sub-topic called Lighting, and under that, there is a subtopic called Lamp. If we send any message to this topic via a client, any clients connected to this particular topic via our broker will receive this message.

There may be cases when you would want to subscribe to multiple topics from a single client. For instance, if you want to connect to 30 such topics, it would be very tedious to write each topic name. There is where Wildcards come into play. They let you subscribe to multiple topics with a single statement. There are two types of wildcards in MQTT: single-level and multi-level.

Single-level wildcards

In the case of single-level wildcards, you can use them to substitute a single sub-topic hierarchy. This can be done by simply using the + symbol instead of the subtopic's name:

`Bedroom/+/Lamp`

In this case, you can create any topic with the main topic as Bedroom and the third subtopic as Lamp. The value of the second topic can be anything:

```
Bedroom/Lighting/Lamp
Bedroom/State/Lamp
```

However, you can use the following code because the third subtopic in the hierarchy changes:

```
Bedroom/Lighting/LED
```

Multi-level wildcards

Now, consider a case when you have multiple possible values in the third subtopic hierarchy as well. In that case, using the single-level wildcard won't be enough. Here, we must use the multi-level wildcard, #, which allows you to subscribe to all the subtopic levels. Consider the following code:

```
Bedroom/Lighting/#
```

In this case, all the topics that start with Bedroom/Lightning/ will be subscribed (all the subtopic levels will be included).

Now, the question is, how does our broker distinguish between clients? The answer is using client IDs. Every MQTT client has a client ID that should be unique according to the protocol rules. In most libraries that you will be using, the system's client ID would be an auto-generated random string to keep the IDs unique.

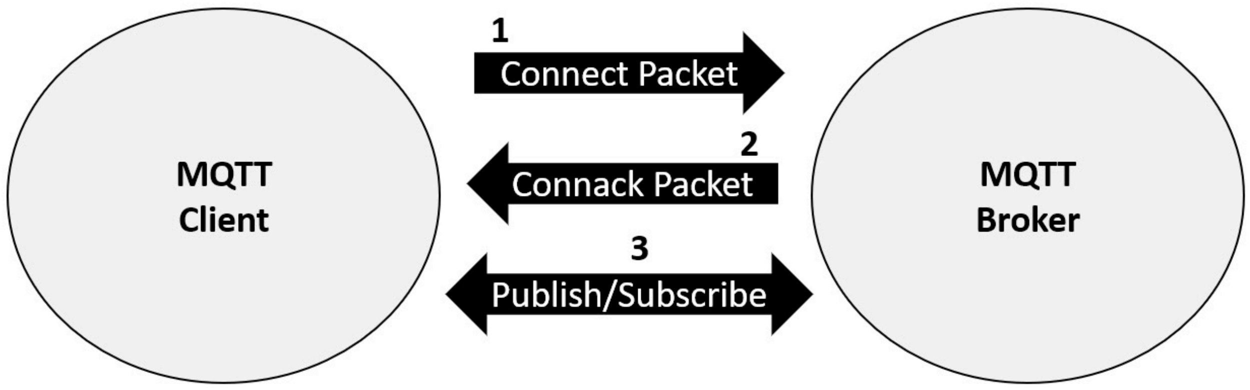
MQTT clients

I've simplified the definition of an MQTT client as any device that runs an MQTT connection package and connects to an MQTT broker over a local or internet network.
Note that no specific device type is mentioned in the definition. This indicates that a client can be a small microcontroller or microprocessor-based device, or that it can be a full-fledged server. For example, the MQTT client can be a tiny and portable device that connects wirelessly (Wi-Fi) and has a basic MQTT library (for instance, a NodeMCU board). The MQTT client can even be a computer running an MQTT client program for testing purposes. Any device that can use MQTT over TCP/IP can be called an MQTT client.
Next, an MQTT client can either be a publisher, subscriber, or both. This depends on the application. For example, a computer dashboard would most likely be a subscriber to several MQTT topics as its main task is to show a visual representation of collected data. On the other hand, a sensor node will most likely be a publisher who constantly sends the collected sensor data.

How does an MQTT client connect to a broker?

The MQTT broker is another component of an MQTT connection. Its main task is to manage all the incoming and outgoing messages. This includes handling all the topics of the network. Moreover, it stores some missed messages if a particular client has opted for the corresponding

Quality of Service (QoS). Hence, a single broker can simultaneously handle thousands of clients or more if it is a large-scale implementation.
Now, how exactly does a client connect to a broker either as a subscriber, publisher, or both? The answer is using something called MQTT control packets (in simple terms, this involves exchanging specific information via a network) – a connect packet in this case. Whenever a client wishes to connect to a particular MQTT broker, it sends a connect packet to the broker with the necessary attributes. In response, a broker sends a connect acknowledgment (CONNACK) packet that contains a status code indicating if the connection was successful and, if not, the reason why it failed. The following diagram visualizes how this process works:

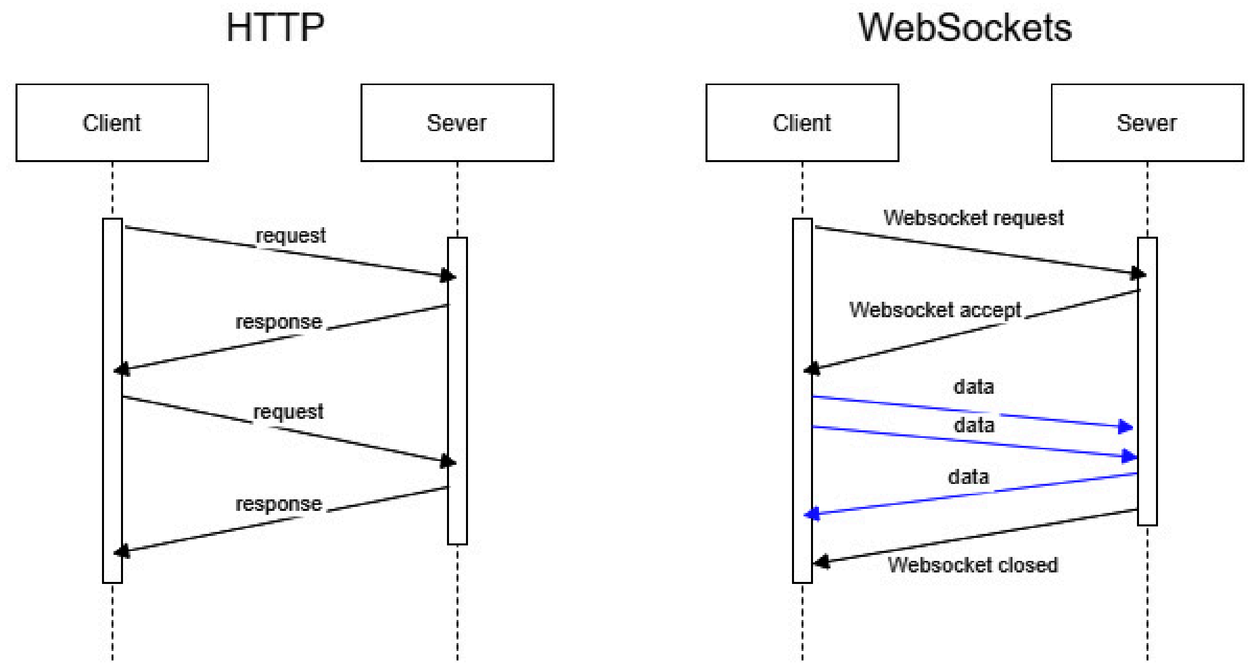


Please note that in actuality, what happens is that after a client establishes a connection to the broker in a network, the CONNECT message is the first message it sends.
Moreover, only one CONNECT message can be sent by a single client. If it attempts to send another one, it results in a protocol violation and disconnects from that client.

What about WebSocket?

WebSocket takes the best of both worlds: HTTP and persistent connections. It uses a handshake to establish communication.

The connection starts with an HTTP GET from the client. In the request, the client specifies an upgrade to WebSocket. WebSocket allows exchanging data without the need to open and close the connection for each transfer. This reduces the overhead produced by the protocol:



How to select the right IoT protocol

The selection of the right protocol for your IoT device depends on several factors, as follows:

- **Type of communication technology:** Ethernet, cellular, Wi-Fi, BLE, and so on
- **Type of electrical power supply:** Mains, battery, energy harvesting, or any combination of them
- **Electrical consumption**
- **Type of data transferred:** Sensor values, images, sound, video, and so on
- **Latency expected**
- **Type of application:** Remote sensing, real time, smart building, industrial, and so on

If you have a wired connection, power usage is generally not a problem. Maybe you are using Power over Ethernet (PoE) to feed your IoT device, or you feed it directly from the mains.
In this case, you have high power and high bandwidth available. This type of device can be, for example, a network camera, a smart Power Distribution Unit (PDU), or a wired sensor, to name a few.
In this case, you can use HTTP or WebSocket to transfer data in a reliable and secure way.
Now imagine a wireless sensor and feed it from a little battery. You will try to extend the battery life as long as you can. In this case, you should choose light protocols, such as MQTT.
So, to choose the right protocol for your IoT device, you will have to address the specific requirements of your application.