MQTT – HOW IT WORKS

Rebecca S. Goodrich[*]
Graduate Student, M.S./Computer Information Systems program, Rivier University

Keywords: *MQTT Protocol, IoT, IoE, Smart Appliances, Smart Homes, Automation, IoT devices, TCP/IP Protocol Suite*

Abstract

The Internet of Things (IoT) covers a vast array of industries and commercial applications. These devices have various protocols for communicating with one another. The Message Queuing Telemetry Transport (MQTT) is one of the more popular protocols and has several features that make it an excellent choice for many applications. MQTT works off the TCP/IP protocol suite to use existing features and it is very lightweight, perfect for IoT devices with limited computing resources.

Introduction

The Internet of Things, or more often IoT, is the networking of devices that communicate with one another, and are individually addressable. People everywhere are dealing with IoT devices all the time and may not even be aware of it. Some common devices in everyday life are your smart watch, the GPS collar you use to track your dog, the coffee mug that keeps your coffee at just the right temperature for hours, your doorbell, or your home security system.

IoT expands to more than just consumer devices, industries like medical, manufacturing, agriculture, even retail environments take advantage of the network devices, such as sensors and robots. These devices provide vast amounts of data to help improve efficiency, medical outcomes, crop yields, and bottom-line dollars.

If all our devices are talking to each other and sharing data to ultimately make our lives better, the question is, HOW are they talking?

IoT Protocols

There are different protocols that can be used for IoT devices to communicate. The Constrained Application Protocol (CoAP), the Advanced Message Queuing Protocol (AMQP), and the Message Queuing Telemetry Transport (MQTT) are just a few of the different protocols used to transfer data between devices.

If many of these devices are communicating across the internet, then why not HTTP? The Hypertext Transfer Protocol (HTTP) is a protocol for transmitting hypermedia documents, such as HTML and the application layer. "It was designed for communication between web browsers and web servers, but it can also be used for other purposes" (HTTP | MDN, n.d.). HTTP can be used with IoT devices, and it likely is being used in some applications, however. HTTP has several drawbacks when it comes to using it with IoT devices.

First, HTTP is a 1-1 communication-based protocol, meaning only two devices can communicate at a time, like a client and server. When you take an example like manufacturing where there can be many sensors trying to transmit data to a server, HTTP breaks down. Secondly, HTTP is designed as a request-response protocol. For data to be transmitted a client needs to make a request, however, in IoT devices, event-based communication is usually what is needed. IoT devices also have small computing resources, so trying to program event-based messaging using HTTP can be challenging. A third reason HTTP is not a smart choice for IoT is that it is not scalable. If an IoT device is connected to multiple other devices there can be a lot of strain put on the system. Because of these, and many other reasons, other protocols have been developed to use with IoT devices. (Why HTTP Is Not Suitable for IoT Applications. - Concurrency, n.d.).

CoAP or the Constrained Application Protocol is a specialized protocol for use with constrained networks, meaning:

- Low bitrate
- High packet loss and high variability of packet loss
- Highly asymmetric link characteristics
- Lack of advanced network services.

"The protocol is designed for machine-to-machine (M2M) applications such as smart energy and building automation." ("CoAP — Constrained Application Protocol | Overview").

Another protocol used in IoT is the Advanced Message Queuing Protocol (AMQP) that, like the Message Queuing Telemetry Transport (MQTT) protocol, uses a middleman to route messages that are subscribed to. In AMQP, an exchange is used which routes messages to topic queues, and then onto the subscribing client. AMQP does offer more security and can be more extensible, subsequently, if bandwidth is a concern. It is not as light weight as MQTT, and performance can suffer.

MQTT

The History of MQTT

In the 1990s, the need for a lightweight messaging protocol to transmit data from remote locations with exceptionally low connection speeds was solved with the development of IBM's Message Queuing Telemetry Transport (MQTT). Ten years later, MQTT had some commercial use in automation and weather forecast, but it really shined when it was used by Facebook™ and Apple™ messaging applications. One of the reasons MQTT was interesting was it supported Quality of Service (QoS) to be a part of its packets, allowing a client to receive a message even when there was an unreliable network connection. (Schultz, n.d.)

Main Features

MQTT has several features that make it an attractive protocol for use in many IoT devices and networks. The first feature is that it is an IoT messaging protocol that sits on top of the TCP/IP protocol, utilizing and building off features of TCP. Secondly, MQTT has minimal overhead, it requires little bandwidth, and the packets are small. Third, MQTT is exceptionally reliable across unreliable networks. Because of features like QoS, Last Will and Testament, and Retained Messages, it has several built-in features that assure the client's messages are being delivered or inform a subscriber when something has gone offline.

A characteristic of MQTT that makes it an excellent choice for IoT is that since it was built for machines to transmit data, MQTT uses binary, which is very efficient, it is bi-directional, data agnostic (it does not care what kind of data you send, - strings, videos, images, etc.), it is scalable and built for push communication, which offers the lowest latency rates. Lastly, it is suitable for constrained devices, because it does not require a lot of computing power, it can be used on devices with limited memory. (Skerret, n.d.)

MQTT uses a publish/subscribe pattern, often referred to as pub/sub to send its messages. These are event-based messages: when an event occurs, the publisher sends the message to a broker. Other clients subscribe to topics, and when the broker receives a message that a client is subscribed to it will push the message out to them (shown in Fig. 1).

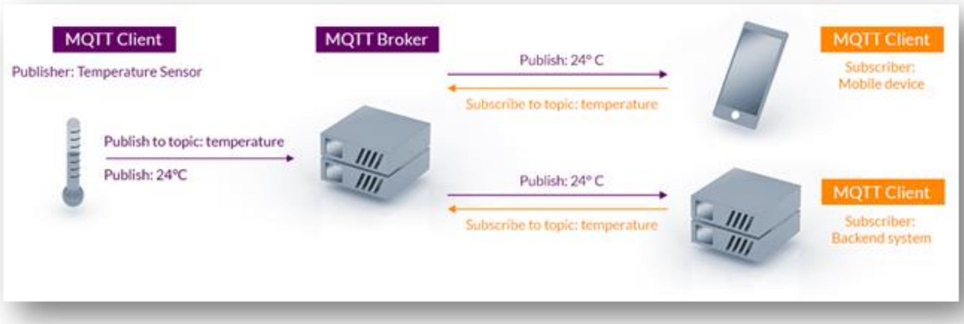
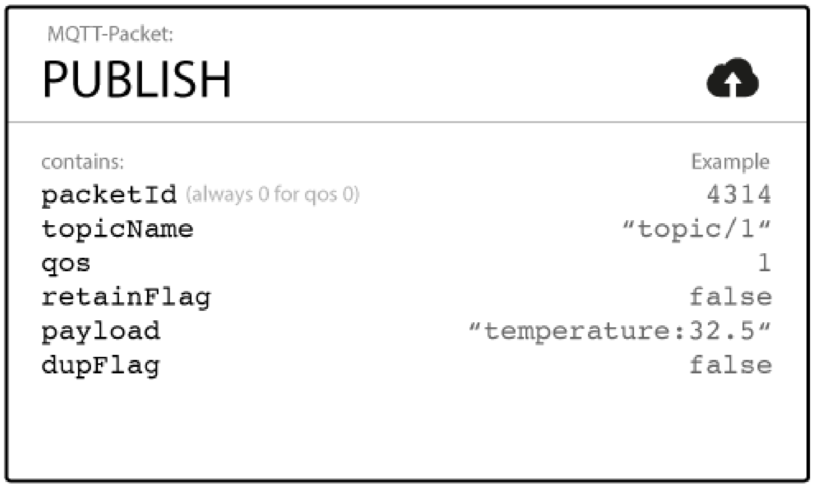

MQTT Client
Publisher: Temperature Sensor
Publish to topic: temperature
Publish: 24°C
MQTT Broker
Publish: 24° C
Subscribe to topic: temperature
MQTT Client
Subscriber: Mobile device
Publish: 24° C
Subscribe to topic: temperature
MQTT Client
Subscriber: Backend system

Figure 1. MQTT Publish/Subscribe Example (MQTT - The Standard for IoT Messaging, n.d.)

The MQTT architecture is also scalable: in that, clusters of brokers can be utilized as backups in the event a broker goes down. Clustering removes the single point of failure.

The Publish/Subscribe pattern also allows for the decoupling of space, time, and synchronization. Clients do not need to be geographically close to each other, messages can be queued to send when the client is online and sending and receiving can be asynchronous (unlike HTTP).



MQTT-Packet:
PUBLISH

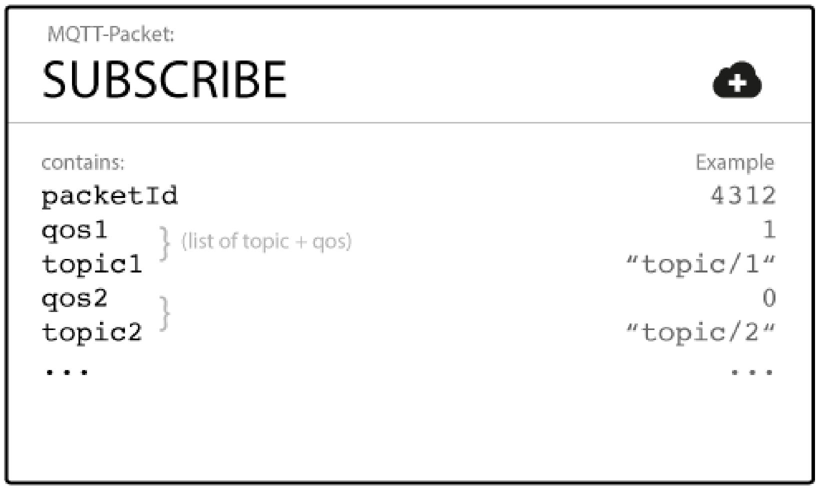
contains:	Example
packetId (always 0 for qos 0)	4314
topicName	"topic/1"
qos	1
retainFlag	false
payload	"temperature:32.5"
dupFlag	false

Figure 2. Publish Packet HiveMQ (Team, n.d.)

Topics

MQTT uses topics to organize the messages a client can subscribe to (see Fig. 2). The topics are UTF-8 strings, organized like hierarchal file folders, from general to specific, and use the forward slash between each level (e.g., firstfloor/hallway/lights). The topics are also case sensitive and must be at least one character in length. Wild card patterns can also be used to subscribe to an entire level of topics. Wildcards are either a '+' or a '#'. The '+' denotes that the client is subscribing to everything on that level (e.g., firstfloor/+/lights - the client is subscribing to everything related to lights on the first floor). The '#' denotes that the client is subscribing to everything after the level (e.g., /FirstFloor/# - the client is subscribing to everything on the first floor). (Skerret, n.d.)

Topics should be short and concise, only using ASCII characters, and unique identifiers should be used for client IDs, this allows for a client to subscribe to a single device or sensor. It is also important to note that a client should never subscribe to just '#' as a root topic, - it can cause many problems and break down the entire network.



MQTT-Packet:
SUBSCRIBE

contains:		Example
packetId		4312
qos1	} (list of topic + qos)	1
topic1		"topic/1"
qos2	}	0
topic2		"topic/2"
...		...

Figure 3. Subscribe Packet HiveMQ (Team, n.d.)

Quality of Service (QoS)

Quality of Services, or QoS, is a feature of MQTT that can guarantee delivery of messages sent between the client and the broker, depending on which type of QoS is chosen for a given message (see Fig. 3). This is a huge benefit for clients/devices that may drop connections frequently. For example, a car that gets data about the weather from a broker. As it travels it may go through an area without connection to a network and reconnect when it gets back to an area with a stable network, it can then receive messages from the broker about the current weather conditions.

There are three levels of QoS in MQTT:

- QoS 0 – At most once delivery
- QoS 1 – At least once delivery
- QoS 2 – Exactly once delivery

QoS 0, at most once delivery, specifies that the message will be published once without any response required, also known as 'publish and forget' (see Fig. 4). QoS 0 is recommended when message queuing is not required, and message loss is acceptable. For example, if a sensor is publishing the temperature every few seconds, the loss of a single message would not result in a major loss of data; therefore, QoS 0 could be reasonably implemented here. Also, if bandwidth is excellent and devices losing connection is rare, then QoS 0 can also be used. (Team, n.d.)

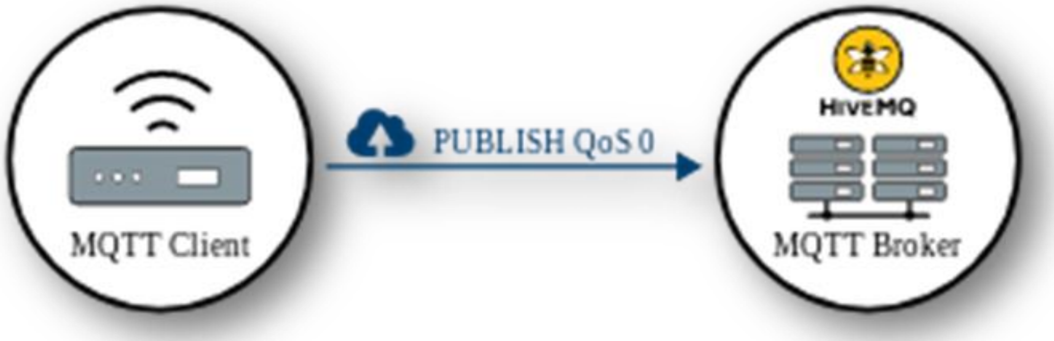


Figure 4. MQTT QoS 0, HiveMQ (Team, n.d.)

QoS 1, at least once delivery, specifies that a PUBLISH message requires a response and that the message will continue to be sent repeatedly until a response is sent back. This is the default setting for messages. The way QoS 1 works is that, when a message is sent, a PUBACK packet is required in response (see Fig. 5). If the PUBACK packet is not received in a certain time frame, then the message is sent again. The message will continue to resend until the PUBACK packet is sent in response. This level of QoS means that there is a tradeoff between bandwidth and the delivery guarantee. (Team, n.d.)

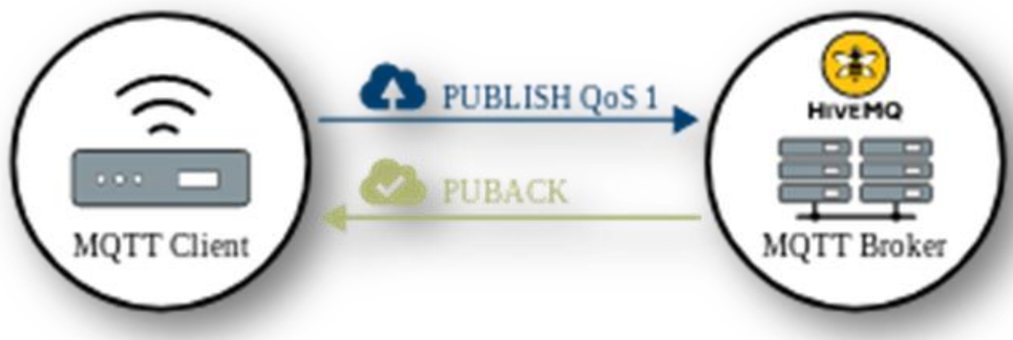


Figure 5. MQTT QoS 1 HiveMQ (Team, n.d.)

QoS 2, exactly once delivery, specifies that a message is sent exactly once, and this is accomplished by a four-part handshake (see Fig. 6). In this four-part handshake there are two separate request/response flows. First, the client sends the PUBLISH message and awaits a PUBREC packet from the broker to confirm receipt. If the PUBREC is not received, the client sends the PUBLISH message again with a duplication (dup) flag. Once the client receives the PUBREC from the broker, it destroys the PUBLISH message that it had saved and stores the PUBREC; it then sends back a PUBREL message. When the broker receives the PUBREL, it sends back a PUBCOMP message letting the client know the message receipt is complete. QoS 2 should only be used when bandwidth is insufficient and lower performance is acceptable, this is QoS, while the safest option, is also the slowest option and takes up more bandwidth than the others. The best practice is to use QoS 2 when duplicate delivery of messages is harmful to the application users or subscribing clients, otherwise the default can be QoS 1.

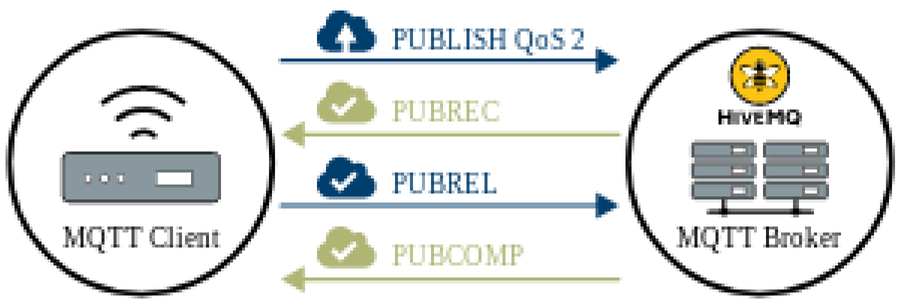


Figure 6. MQTT QoS 2 HiveMQ (Team, n.d.)

Persistent Sessions and Message Queuing

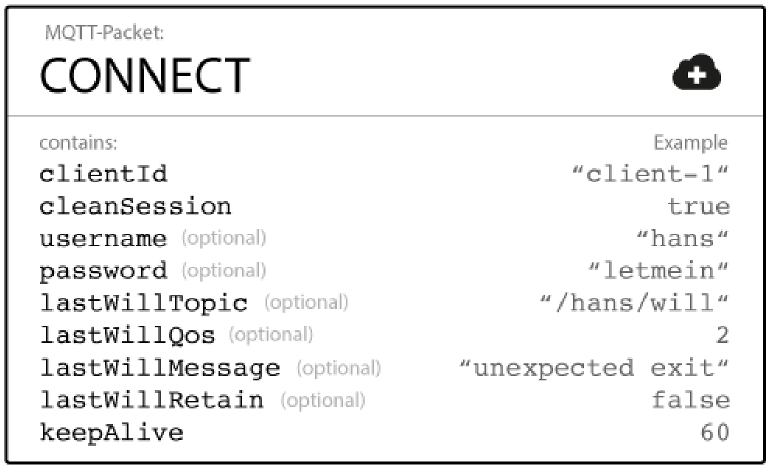
Persistent Sessions. Unlike a clean session where all stored client information is deleted when the session ends, a persistent session allows the broker to store information for future reuse. This feature of MQTT is especially useful for storing subscriptions, unacknowledged QoS messages and queued messages. The storing of subscriptions is especially important, so clients do not have to resubscribe to topics every time they connect to the broker, the broker will remember what topics clients are subscribed to, this improved performance and moves the complexity to the broker. (Team, n.d.)

Queued messages. Messages stored from the QoS 1 and QoS 2 messages waiting to be delivered to a persistent session client who is currently offline are called queued messages. Queued messages are used in a client context, and a broker queues the messages to send to a specific client, as opposed to all clients subscribed to a topic. (Team, n.d.)

Retained Messages. Another type of stored message is a retained message. A retained message works at the topic level and is a message that is stored for a newly subscribed client, - this can sometimes be referred to as a birth message. These messages are useful for when a start state is required and is sent to a new subscriber to avoid the empty start value problem. It is important to note that only one retained message per topic is stored at a time. When a new message that is flagged as a retained message is sent, it overwrites any existing message for the same topic. (Team, n.d.)

Last Will and Testament

The Last Will and Testament (LWT) in MQTT is an optional feature that assists with error detection. The LWT is a notification when a client goes offline (ungracefully disconnects). LWT messages are coded in the connect packet and can contain the following optional values lastWillTopics, lastWillQoS, lastWillMessage, lastWillRetain. The broker will send a LWT message in several circumstances such as when it detects an error or networking failure, the client fails to communicate within a defined keep alive period, the client does not send a disconnect packet before closing the network connections, or the broker closes the network connection due to protocol error. LWT can be used for any online/offline mechanism like letting a subscriber know a publisher/client is offline.


MQTT-Packet:

CONNECT

contains:	Example
clientId	"client-1"
cleanSession	true
username (optional)	"hans"
password (optional)	"letmein"
lastWillTopic (optional)	"/hans/will"
lastWillQos (optional)	2
lastWillMessage (optional)	"unexpected exit"
lastWillRetain (optional)	false
keepAlive	60

Figure 7. Connect Packet (Team, n.d.)

Keep Alive and Client Takeover

MQTT offers some useful features for when the issue of a half open TCP connection occurs. Sometimes, particularly when the network is unreliable, and TCP connection can be half open, meaning one side of the connection has closed and the other side is not aware. A half open TCP connection can cause problems if there is not a failsafe in place; for example, if the client that has gone offline comes back online with a new session and the old session is never terminated it can wreak havoc. So, MQTT has the Keep Alive feature which allows the broker to disconnect a client if it does not receive a packet in a designated time frame. The default for this period is 60 seconds, however, the time can be set to whatever is needed.

Another feature helps solve issues with half open TCP connection and the Client Takeover. When a client disconnects from the broker and reconnects with a new session, the old session is broken. It is important to use unique client IDs for this reason, otherwise, client takeovers could happen when they are not expected. This is also an important reason to use authentication as this could become a major vulnerability to the network.

MQTT in the Wild

Industrial Cases

In the industrial arena, MQTT can be used to publish all kinds of messages for error detection and for data analysis. For example, a welding controller publishes data about voltage, currents, temperature, etc., and publishes this information to a broker who then sends the data to a server that has subscribed to this data. The server stores the data in a database. The company uses this data for quality control and to improve their processes. The company also uses the Last Will and Testament feature of MQTT to know when a machine goes offline unexpectedly.

Agricultural Cases

In the world of agriculture, the weather is everything. Using sensors and devices connected to a network can be the advantage farmers need to stay on top of their crops. Farmers can use sensors to detect the conditions of the soil, - is it dry and, if so, does it need water or shade? How is the pH or nutrient content? Do the crops need to be fertilized? Sensors can detect all these data points and, using MQTT, relay them to the subscribed clients. One of the interesting things being done in agriculture using IoT and (in this case) MQTT protocol is that sensors can detect when crops need, for example, water. The sensor publishes this data to the broker. The broker then sends this data to a watering device that is subscribed to the topic. When the watering device gets the message that the crops need water, it can then do so automatically. This process automates the farming process and makes it foolproof; the result can be larger crop yields for farmers across the globe. By improving crop yields, the cost of food can be reduced. In countries that have large growing populations, this can be the key to keeping the masses fed.

Consumer

An extremely popular device for the smart home is the Ring doorbell. This little device can alert your smart phone when the doorbell rings, it can also take video of who is outside your home and allow you to communicate with them via microphone/speaker capability. When someone rings the Ring doorbell, it uses MQTT to publish a message to the broker that the doorbell has rung. The subscribing client is then notified of the event. Depending on the subscribing client, different actions can be taken. The security camera could be a subscriber that once alerted to the motion, and the ringing of the doorbell begins recording and transmitting the video feed. The homeowner's cell phone app, as a subscriber, would be notified of the doorbell event and notify the homeowner via push notification. Lastly, if the homeowner also owns the Ring Chime device, the chime device would also be a subscriber to this topic, and when notified of the event would then play the designated sound effect, alerting everyone inside the home that there is someone at the door.

Conclusion

The IoT is all around us, from the way our cars are made, to the insulin pumps keeping our friend alive, to the doorbell on your home, etc. These devices are helping to make life better, easier, and more automated.

They improve medical outcomes and crop yields, keeping our friends and family alive and well. The ways in which they communicate and share the data that is helping us to ensure better outcomes is fascinating, and MQTT is the only one of a handful of ways in which these devices communicate. As we see, the emergence of the 5G improvements to our bandwidth connections will only improve the way these devices can communicate and usher in a new era of protocols as well.

References

AMQP vs MQTT: Comparing Instant Messaging Protocols. (n.d.). Retrieved July 27, 2022, from https://www.cometchat.com/blog/amqp-vs-mqtt-comparing-instant-messaging-protocols

CoAP — Constrained Application Protocol | Overview. (n.d.). Retrieved July 27, 2022, from https://coap.technology/

El-Azab, R. (2021). Smart homes: potentials and challenges. *Clean Energy*, *5*(2), 302–315. https://doi.org/10.1093/ce/zkab010

Global Smart Home Appliances Market Is Set to Reach $139 Billion by 2030, says P&S Intelligence. (n.d.). Retrieved July 11, 2022, from https://www.prnewswire.com/news-releases/global-smart-home-appliances-market-is-set-to-reach-139-billion-by-2030-says-ps-intelligence-301541609.html

How Smart, Connected Products Are Transforming Competition. (n.d.). Retrieved July 11, 2022, from https://hbr.org/2014/11/how-smart-connected-products-are-transforming-competition

How will IoT devices or systems cope with the crowded network? (2016, February 16). *Itrinegy*. https://itrinegy.com/how-will-iot-devices-cope-with-crowded-networks/

HTTP | MDN. (n.d.). Retrieved July 27, 2022, from https://developer.mozilla.org/en-US/docs/Web/HTTP

IP-Addressable Smart Appliances for Demand Response Applications. (n.d.). Retrieved July 11, 2022, from https://www.epri.com/research/products/1016080

MQTT - The Standard for IoT Messaging. (n.d.). Retrieved July 17, 2022, from https://mqtt.org/

Nistor, S., Wu, J., & Sooriyabandara, M. (2013). The impact and opportunities of smart appliances on distribution networks. *IEEE PES ISGT Europe 2013*, 1–4. https://doi.org/10.1109/ISGTEurope.2013.6695479

Paessller, AG. (n.d.). *What is MQTT? What you need to know* [Video]. Retrieved July 17, 2022, from https://youtu.be/QSwR-JMmNOo

Schultz, D. (n.d.). *What Is MQTT, and How Can Industrial Automation Companies Use It?* Retrieved July 27, 2022, from https://blog.isa.org/what-is-mqtt-and-how-can-industrial-automation-companies-use-it

Skerret, I. (n.d.). *What Are IoT Devices & How MQTT Powers Them*. Retrieved July 19, 2022, from https://www.hivemq.com/blog/what-are-iot-devices-and-how-mqtt-powers-them/

Team, T. H. (n.d.-a). *Introducing the MQTT Protocol - MQTT Essentials: Part 1*. Retrieved July 18, 2022, from https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt/

Team, T. H. (n.d.-b). *Persistent Session and Queuing Messages - MQTT Essentials: Part 7*. Retrieved August 3, 2022, from https://www.hivemq.com/blog/mqtt-essentials-part-7-persistent-session-queuing-messages/

Team, T. H. (n.d.-c). *Quality of Service (QoS) 0,1, & 2 MQTT Essentials: Part 6*. Retrieved August 3, 2022, from https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/

The Driving Demand for Bandwidth in the Home | Corning. (n.d.). Retrieved July 11, 2022, from https://www.corning.com/optical-communications/worldwide/en/home/the-signal-network-blog/the-driving-demand-for-bandwidth-in-the-home.html

The growth of Smart Appliances and what it means for Utilities. (2018, April 17). Open International. https://www.openintl.com/the-growth-of-smart-appliances/

Top 10 Barriers for Adoption of the Internet of Things. (n.d.). Retrieved July 14, 2022, from https://www.iotworldtoday.com/2016/04/20/top-10-reasons-people-aren-t-embracing-iot/

Trifork. (n.d.). *A Gentle introduction to IoT protocols: MQTT, CoAP, HTTP & WebSockets*. Retrieved July 18, 2022, from https://gotoams.nl/2017/sessions/126/a-gentle-introduction-to-iot-protocols-mqtt-coap-http-and-websockets

What Happens to Your Smart Home if the Internet Goes Down? (n.d.). Retrieved July 17, 2022, from https://www.makeuseof.com/smart-home-internet-goes-down/

What is a Smart Appliance - and How Do You Make One? | Ayla Networks. (n.d.). Retrieved July 24, 2022, from https://www.aylanetworks.com/blog/what-is-a-smart-appliance-and-how-do-you-make-one

Why HTTP is not suitable for IOT applications. - Concurrency. (n.d.). Retrieved July 27, 2022, from https://www.concurrency.com/blog/june-2019/why-http-is-not-suitable-for-iot-applications

---

[*] Rebecca Slosberg Goodrich is a student in Rivier's M.S./Computer Information Systems program. This paper is based on her research project in the CS553 Networking Technology core course taught by Prof. Peter Macdonald in summer 2022.

.