جامعة عجمان
AJMAN UNIVERSITY

IoT: Pub/Sub – MQTT

IoT Essential Technologies

Mohamed Al Solh

alsolh@alsolh.com

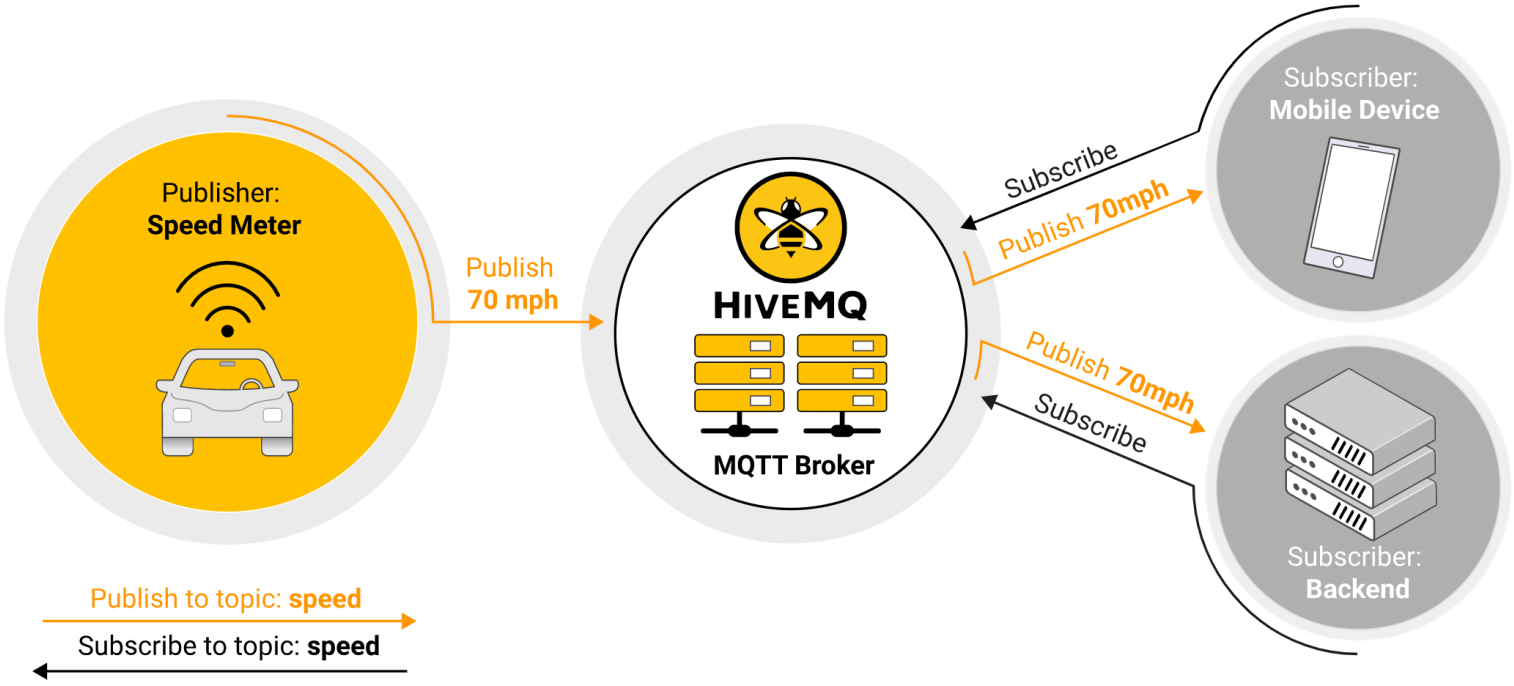
Previous Topics

- Swagger
- Actuator
- Certificates

MQTT History

- In the late 1990's, Andy Stanford-Clark (IBM) and Arlen Nipper (Cirrus Link) invented MQTT to monitor oil and gas pipelines over satellite networks. They designed the MQTT protocol to be open, simple, and easy to implement. The result is an extremely lightweight protocol that minimizes network bandwidth and device resource requirements with some assurance of reliable delivery
- These characteristics make MQTT ideal for use in constrained environments and low-bandwidth networks that have limited processing capability, low memory capacity, and high latency such as the Internet of Things (IoT)

Publish Subscribe



MQTT Publish / Subscribe Architecture

Share paper and diagram for mqtt performance

Criteria	HTTP	CoAP	MQTT
Architecture	Client/Server	Client/Server or Client/Broker	Client/Broker
Abstraction	Request/Response	Request/Response or Publish/ Subscribe	Publish/Subscribe
Header Size	Undefined	4 Byte	2 Byte
Message size	Large and Undefined (depends on the web server or the programming technology)	Small and Undefined (normally small to fit in single IP datagram)	Small and Undefined (up to 256 MB maximum size)
Semantics/Methods	Get, Post, Head, Put, Patch, Options, Connect, Delete	Get, Post, Put, Delete	Connect, Disconnect, Publish, Subscribe, Unsubscribe, Close
Quality of Service (QoS) /Reliability	Limited (via Transport Protocol - TCP)	Confirmable Message or Non-confirmable Message	QoS 0 - At most once QoS 1 - At least once QoS 2 - Exactly once
Transport Protocol	TCP	UDP, TCP	TCP (MQTT-SN can use UDP)
Security	TLS/SSL	DTLS/IPSEC	TLS/SSL
Default Port	80/443 (TLS/SSL)	5683 (UDP)/5684 (DTLS)	1883/8883 (TLS/SSL)

* N. Naik, "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP," 2017 IEEE International Systems Engineering Symposium (ISSE), Vienna, 2017, pp. 1-7.

https://datatracker.ietf.org/meeting/103/materials/slides-103-maprg-evaluating-the-performance-of-coap-mqtt-and-http-in-vehicular-scenarios-jaime-jimenez

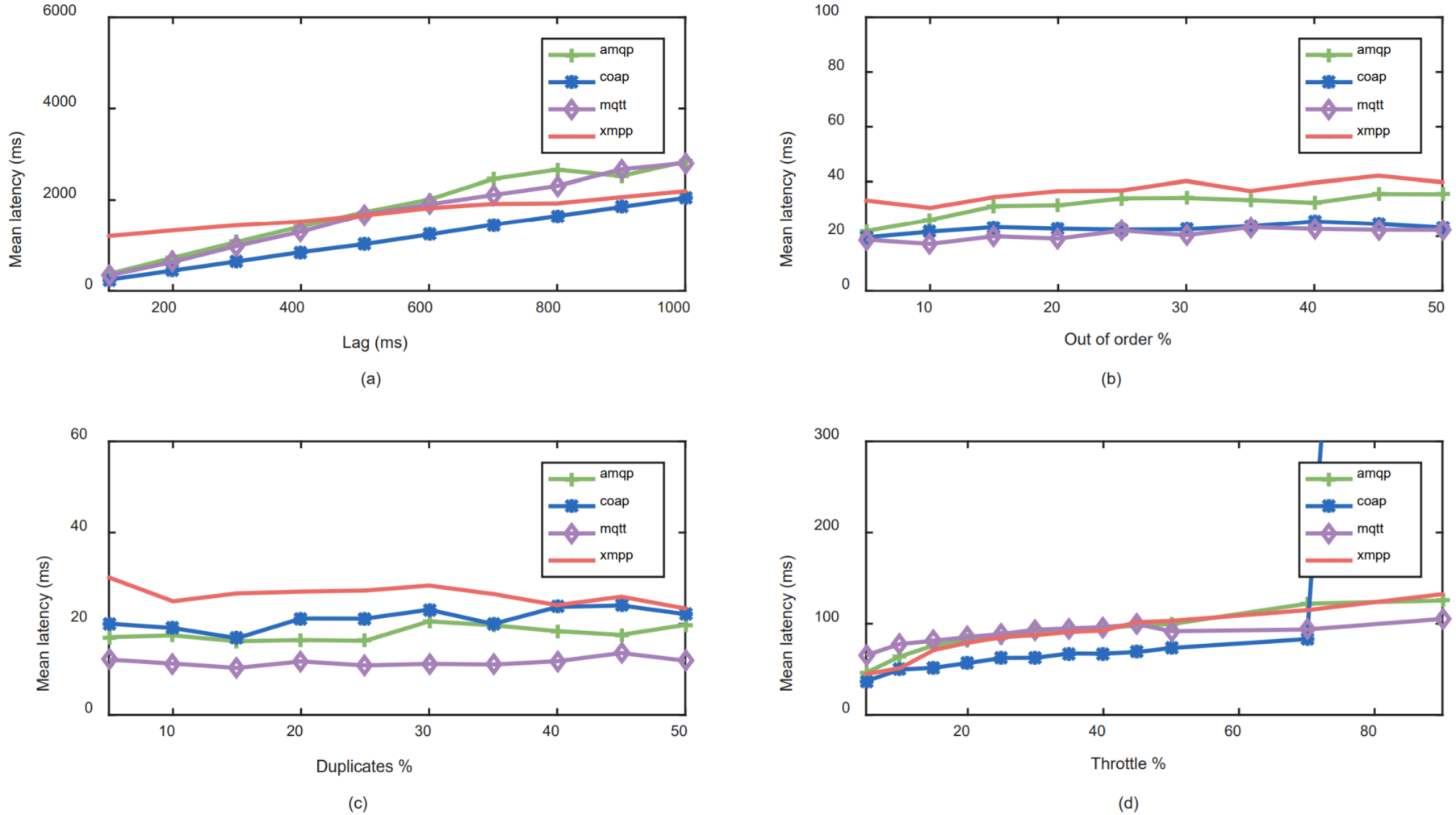


Figure 4 Average latency under various network channel disturbances

https://dl.acm.org/doi/pdf/10.1145/3290511.3290539

- **Pub/Sub scales better than the traditional client-server approach**. This is because operations on the broker can be highly parallelized and messages can be processed in an event-driven way

- OPTION 1: SUBJECT-BASED FILTERING
  - This filtering is based on the subject or **topic** that is part of each message. The receiving client subscribes to the broker for topics of interest. From that point on, the broker ensures that the receiving client gets all message published to the subscribed topics. In general, topics are strings with a hierarchical structure that allow filtering based on a limited number of expressions.
- OPTION 2: CONTENT-BASED FILTERING
  - In content-based filtering, the broker filters the message based on a specific content filter-language. The receiving clients subscribe to filter queries of messages for which they are interested. A significant downside to this method is that the content of the message must be known beforehand and cannot be encrypted or easily changed.
- OPTION 3: TYPE-BASED FILTERING
  - When object-oriented languages are used, filtering based on the type/class of a message (event) is a common practice. For example,, a subscriber can listen to all messages, which are of type Exception or any sub-type.
- **MQTT uses subject-based filtering of messages. Every message contains a topic (subject)** that the broker can use to determine whether a subscribing client gets the message or not

- MQTT decouples the publisher and subscriber spatially. To publish or receive messages, publishers and subscribers only need to know the hostname/IP and port of the broker
- MQTT decouples by time. Although most MQTT use cases deliver messages in near-real time, if desired, the broker can store messages for clients that are not online. (Two conditions must be met to store messages: the client had connected with a persistent session and subscribed to a topic with a **Quality of Service** greater than 0)
- MQTT works asynchronously. Because most client libraries work asynchronously and are based on callbacks or a similar model, tasks are not blocked while waiting for a message or publishing a message

Topic vs Queue

- **A message queue stores message until they are consumed** When you use a message queue, each incoming message is stored in the queue until it is picked up by a client (often called a consumer). If no client picks up the message, the message remains stuck in the queue and waits to be consumed. In a message queue, it is not possible for a message not to be processed by any client, as it is in MQTT if nobody subscribes to a topic.
- **A message is only consumed by one client** Another big difference is that in a traditional message queue a message can be processed by one consumer only. The load is distributed between all consumers for a queue. In MQTT the behavior is quite the opposite: every subscriber that subscribes to the topic gets the message.
- **Queues are named and must be created explicitly** A queue is far more rigid than a topic. Before a queue can be used, the queue must be created explicitly with a separate command. Only after the queue is named and created is it possible to publish or consume messages. In contrast, MQTT topics are extremely flexible and can be created on the fly.

- Home/#
- Home/bedroom/camera
- Home/livingroom/light
- Home/livingroom/#
- #
- Home/#/light
- Home2/#

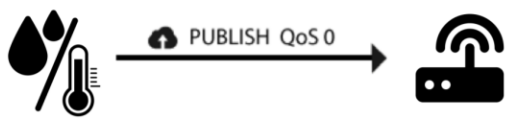
MQTT has three quality of service (QoS) levels

- *At most once* (0)
- *At least once* (1)
- *Exactly once* (2).

- https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/

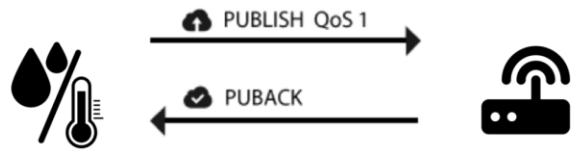
Empirical Results – Impact of **QoS** in **MQTT***

QoS 0 ✖



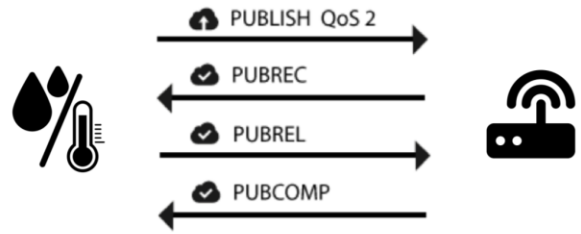
- Best-effort delivery.
- No guarantee of delivery.
- Recipient does not acknowledge receipt of the message and the message is not stored and re-transmitted by the sender

QoS 1 ✔



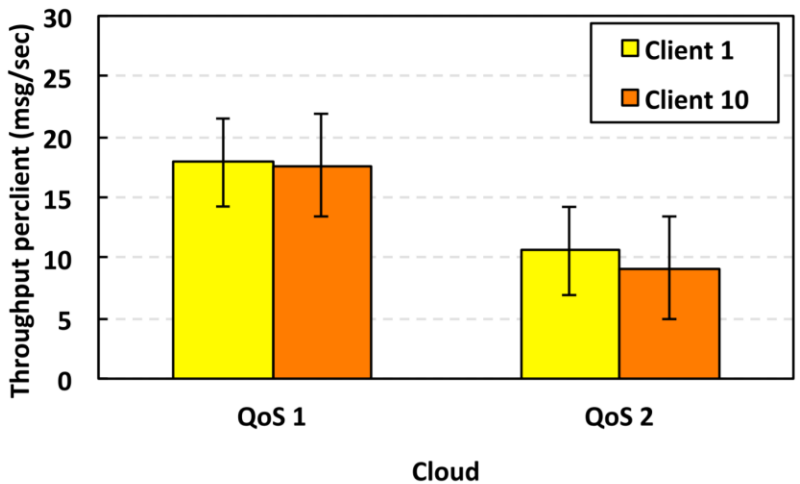
- It guarantees that a message is delivered at least one time to the receiver.
- The sender stores the message until it gets a packet from the receiver that acknowledges receipt of the message.
- Message can be sent or delivered multiple times.

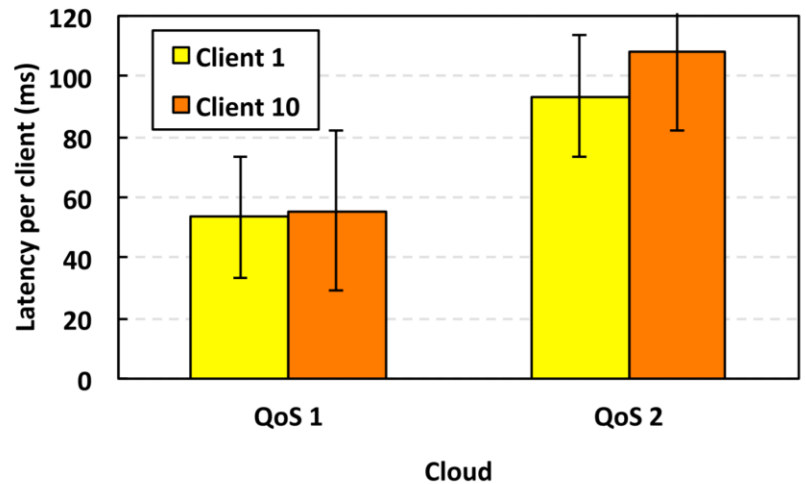
QoS 2 ✔



- It guarantees that each message is received only once by the intended recipients.
- Safest and slowest QoS.
- Guarantee is provided by at least two request/response flows (a four-part handshake) between the sender and the receiver

Empirical Results – Impact of **QoS** in **MQTT.**





- Higher QoS produces a throughput reduction in the order of 40%.
- Higher QoS slows down the message transmission by approximately 75%

- **ClientId**
  - The client identifier (ClientId) **identifies each MQTT client** that connects to an MQTT broker. The broker uses the ClientID to identify the client and the current state of the client.Therefore, this ID should be unique per client and broker
- **Username/Password**
  - MQTT can send a **user name and password for client authentication and authorization**. However, if this information isn't encrypted or hashed (either by implementation or TLS), the password is sent in plain text
- **Will Message**
  - The last will message is part of the Last Will and Testament (LWT) feature of MQTT. **This message notifies other clients when a client disconnects ungracefully**

Other IoT Message Broker Protocols

- AMQP
- COAP

Available Message Brokers

- Mosquito
- HiveMQ
- RabbitMQ
- PONTE
- REDIS
- https://github.com/mqtt/mqtt.github.io/wiki/server-support

Transport Protocols

- Web Socket
- Server Sent Events
- Long Polling

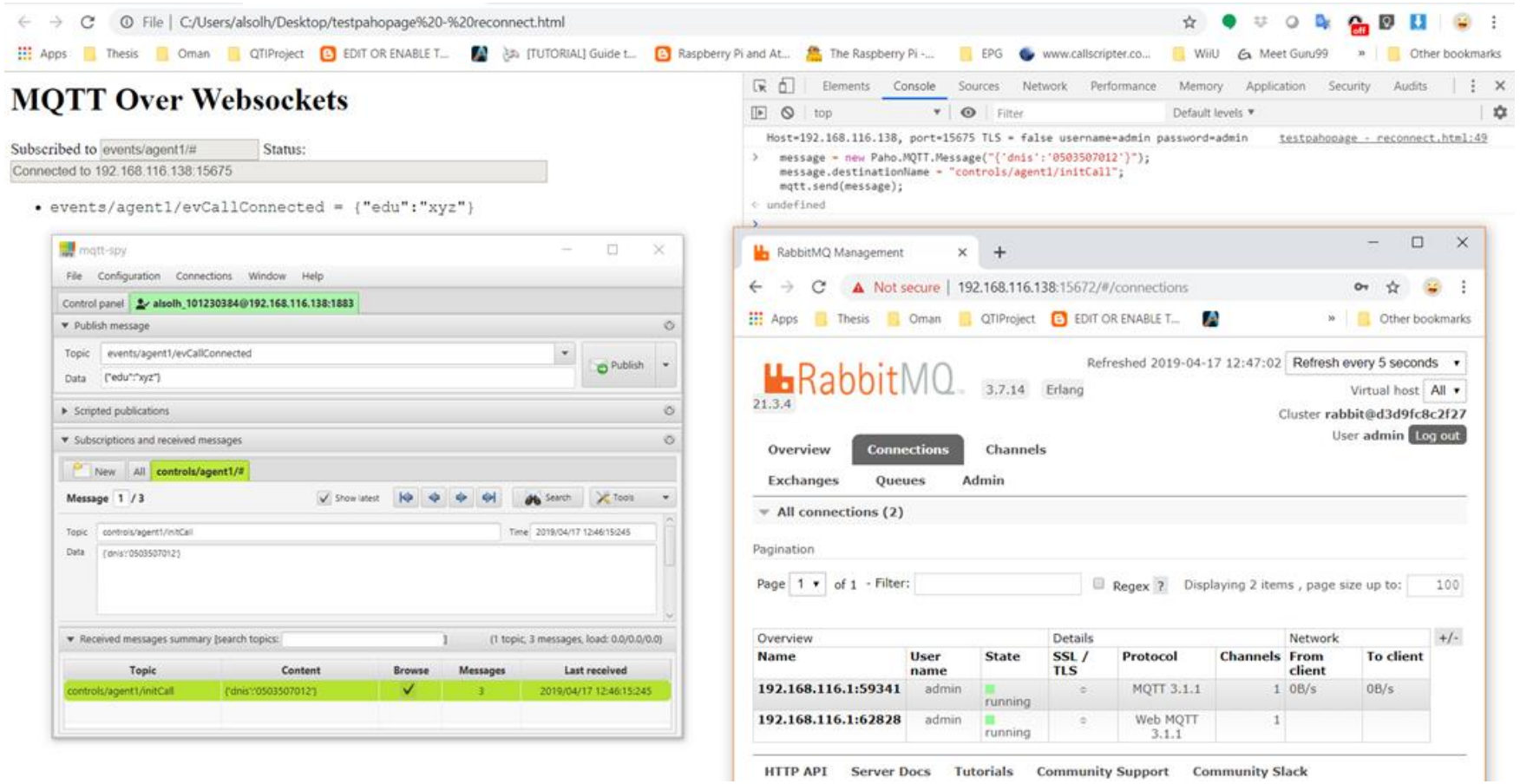
MQTT Clients

- MQTT Spy – (oracle java required) - https://github.com/eclipse/paho.mqtt-spy/releases/tag/1.0.0
- MQTT Explorer - http://mqtt-explorer.com/
- Android App – My MQTT - https://play.google.com/store/apps/details?id=at.tripwire.mqtt.client&hl=en_GB
- Paho js - https://www.eclipse.org/paho/clients/js/

MQTT Topics

- Wildcards
- Heirarchy

MQTT Over Websocket



Some Use Cases

- MQTT in smart watches