# Background

MQTT(MQ Telemetry Transport) is a light weight event and message oriented protocol which allows devices to asynchronously communicate efficiently across constrained remote networks. It is designed for connections with remote locations where only a ‘small code footprint’ is required or where the network bandwidth is limited.

MQTT was built in 1999, with the intention to simplify the connection between the M2M would to the ‘internet of things’. Its use wasn’t controversial until IBM created an open source MQTT. IBM’s version of MQTT had added extra aspect into the design, including a open standard and a list of favourable, neat features.

MQTT’s favourable features provide great advantage in growing its popularity in such short order. Its low code footprint makes it possible to be put onto very small devices. It is efficient which favours in battery saving. MQTT’s low cost consumption makes is possible to conduct rapid, real-time communication with low network usage, with extremely low latency, the smallest fixed header it requires could be as little as two bytes, which is very small compared to other messaging protocols, like http. This light weight protocol is also easy to understand and simple to implement. Change this sentence so that it fits into the paragraph…

MQTT has gotten many acknowledgment as a valid protocol recently. A major one comes from Facebook. Facebook messenger is built upon MQTT, MQTT acts as a low intensive protocol, it provides low latency for about a billion users and makes sure that the messages show up instantly without destroying device battery and network.

The Amazon Web Services has announced Amazon loT based on MQTT.

Home users also receive great benefit from MQTT as many Smart Home systems and alarm systems are also built upon MQTT.

# What does it do

MQTT is designed for obtaining controlled data events from constrained networks and constrained systems into a desired location.

It is a messaging protocol which implement the publish/subscriber messaging (one to many) mechanism.

Say something about p/s mechanism here??!

# Usage

|  |  |
| --- | --- |
| **Term** | **Definition** |
| Topic | A structured string that defines a location in a namespace with ‘/’ used to delimit levels of tat namespace’s hierarchy. A topic can also be viewed as a key or subject of interest. |
| Message | Strings or files in bytes that need to be sent from the publish to the consumers. |
| Publisher | A producer. The uploader of the message. |
| Server | Broker of the message sending media, listens for incoming publish requests and pushes them to corresponding consumers. |
| Consumer | A client, the receiver of the message. |
| Subscription | A consumer can subscribe to one or more topic that they are interested in, in order to receive future updates published to that topic. |

The process: fix this line, should it be a mini heading or a paragraph??

The publish creates a message on a Topic.

Once the message is published, it goes to the server.

The server constantly listens out for requests, when it receive the publish request, it accepts it and pushes it to all of the consumers subscribed to this particular topic. MQTT pushes messages both directions, when there is a client waiting for that message, the server pushes the message across the network immediately, so there is no polling required, which is good mechanism for timely delivery of event messages and data between the producer and the consumer, as polling is knowingly expensive. If there is no consumers, the message gets discarded, if there is more than one consumer, all of them will receive the same message.

There are two types of subscriptions, durable and non-durable.

When a durable subscription is made to the server, the consumer receives the message immediately when connected to the server. If the consumer is offline, the server saves the message and push it again when the consumer reconnects again. This is adopted widely by messaging services.

In a non-durable subscription, the message only last the lifetime when the subscribers are connected to the broker. A good example of this would be the stock market system.

The publisher has a choice of sending a retaining message when publishing. If a message is marked as retained, the server remembers the last topic on the topic, with the purpose that when a new consumer subscribes, they would receive the last message immediately instead of having to wait for the next message to be published. Live weather and road watch broadcasts make great use out of this feature, as it returns the latest update of the condition to even new subscribers.

The quality of service level is an agreement between the producer and the receiver regarding the guarantees of delivering a message.

There are three levels in MQTT: 0, 1 and 2.

Level one sends the message at most once, regardless if it gets delivered or not.

Level two sends the message at least once, because of this, the consumer might receive duplicate of the message. The sender stores the message until it receives an acknowledgement from the receiver. This level guarantees delivery, even if the connection is lost.

Level three guarantees that each message is received only once by the consumer. It is the safest and also the slowest quality of service level. The producer and the client communicate with each other with PUB- packages to ensure that both parties understand that the message has been delivered and received, in order to avoid duplicate.

There are two models for working with MQTT, the always-online model and the sometimes-online model.

The always-online model, the device is always connected, provides timely delivery, which supports real-time messaging and instant updates.

The sometimes-online model is when the device occasionally connects to the server, receives messages and sends messages before becoming offline again.

Both of the models have their uses, but for this assignment, the always-online model is used.

# Introduction

In this assignment, we were required to design and implement a distributed system to read data from an end device to the cloud, then display the results on a mobile phone.

The end device we used consisted of a Linus VM running a system health monitor script. The script monitors load average, disk space and free memory.

The cloud agent we used was CloudMQTT, it was also used to retrieve data onto the mobile phone.

# Design

In design, our distributed system would run the bash file to read data from the VM, a java file would be used to load the result onto the cloud, every six hours. The users would then be able to receive data regarding the VM on their mobile every six hours.

API:

Create a mqtt client oobject

Java client in paho, instantiate client obj tell what sever to connect to

Client id must be unique, if two tries to connect, the first gets kicked out, bad for atuo signins

Specify connection options and connect to it

// Keepalive() use to detect if connect breaks out , smaller number = quicker to find out but expensive

client.connect(opts)

connect with option

sending message:

creating a message:

create mqt messae, “dfgsd”.getBytes()

everything is in bytes in MQTT

can setRetained(true)

publish

sending a message:

need to know what topic to send to

so get topic and publish

the returned delivery token given by the publish is used to determine when delivery is complete,

asynchronous callback = delivery token

callback()

when msg get to server, a callback listener is notified, and returns token, to tell delivery is completed

connectionLost()

if connection breaks, client will be notified and ask if wanted to reconnect and resubcript

messageArrived()

be told topic, message content, process message in this method

hwo to subcribe:

client.subcribe(“sdf”);

client.unsubcribe(“dfsg”);

Describe clodu mqtt dashboard