MQTT

An introduction

Alexandre Moreno

MQ Telemetry Transport

MQTT is a machine-to-machine (M2M)/Internet of Things connectivity protocol. It was designed as an extremely lightweight publish/subscribe messaging transport. It is useful for connections with remote locations where a small code footprint is required and/or network bandwidth is at a premium.

http://mqtt.org

buzzwords

Internet of Things
Machine to Machine
Smart Grid

Purpose of MQTT?

- Fill the gap between the numerous devices and applications that can produce data and the wider world of data consumers
- Good fit for simple push messaging scenarios

At a glance...

- Application-layer protocol
- Runs atop TCP
- Binary wire-protocol
- No message queue albeit the name
- Publish/subscribe pattern
- Designed for resource-constrained devices
- UTF-8 encoded strings (since V3.1)
- Payload-agnostic
- Keep-Alive timer
- QoS

Scalability

“a single 1u-form-factor appliance can handle up to 1 million sensors and 13 million concurrent messages”

link to article

Interoperability

MQTT Interop Testing Day

The goal is to have as many different MQTT client and server implementations participate in interoperability testing to validate the implementation of the upcoming OASIS MQTT standard.

https://wiki.eclipse.org/Paho/MQTT_Interop_Testing_Day

Background

Publish/Subscribe pattern

Publish/Subscribe Messaging Model

- Clients subscribe to *topics* (SUBSCRIBE)
- Messages are published to a specific topic name (PUBLISH)
- A broker server handles the routing of messages

Publish/Subscribe Messaging Model

- Pros
  - Greater network scalability and a more dynamic network topology
  - Decoupling of applications
- Cons
  - Same security vulnerabilities as Client/Server model

Message broker

- *Authenticates* clients

- *Validates*, *transforms* and *routes* messages
- Performs *topic*-based message routing
- *Caches* messages for delivery at a later time
  e.g. Will messages, RETAIN flag
- *Bridges*: brokers can be connected together

The protocol

Message types

```
var MESSAGE_TYPE = {
    CONNECT: 1,
    CONNACK: 2,
    PUBLISH: 3,
    PUBACK: 4,
    PUBREC: 5,
    PUBREL: 6,
    PUBCOMP: 7,
    SUBSCRIBE: 8,
    SUBACK: 9,
    UNSUBSCRIBE: 10,
    UNSUBACK: 11,
    PINGREQ: 12,
    PINGRESP: 13,
    DISCONNECT: 14
};
```

port numbers

TCP/IP port **1883** for MQTT.
Port **8883** for MQTT over SSL.

Message format

Fixed header (2 bytes) + Variable header + Payload

Fixed header format

bit	7	6	5	4	3	2	1	0
byte 1	Message Type				DUP flag	QoS level		RETAIN
byte 2	Remaining Length							

Topics/Subscriptions

- Hierarchical structure of topics
  - e.g. *sensors/temperature*
- wildcard pattern matching for subscriptions
  - multi-level with '#'
    - e.g. *sensors/#* matches both *sensors/foo* and *sensors/foo/bar*
    - # matches all topics
  - single-level with '+'
    - e.g *sensors/+* matches *sensors/foo* and *sensors/bar*, but not *sensors/foo/bar*

Flags

QoS

The Quality of Service used to deliver a message

- 0: Best effort
  - PUBLISH
- 1: At least once
  - PUBLISH + PUBACK
- 2: Exactly once
  - PUBLISH + PUBREC + PUBREL + PUBCOMP

Quality of Service levels and flows

Implementations

Mosquitto

An umbrella project, providing an open source MQTT v3.1/v3.1.1 broker,
client libraries,
language bindings, and client utilities.

Seems the Mosquitto project is moving to Eclipse, discussed next

mosquitto.org

Examples

Mosquitto

C client

```c
#include <stdio.h>
#include <err.h>
#include <mosquitto.h>

void on_message(struct mosquitto *m, void *user_data, const struct mosquitto_message *msg)
{
    fprintf(stderr, "lights at %s are %s\n", msg->topic, (char*)msg->payload);
}

int main(int argc, char **argv)
{
    struct mosquitto *client;
    int ret;

    mosquitto_lib_init();

    client = mosquitto_new("client-id", true, NULL);
    if (!client)
        err(1, "mosquitto failed");

    ret = mosquitto_connect(client, "127.0.0.1", 1883, 60);
```

Mosquitto

Python client

```
#!/usr/bin/env python
import mosquitto

def on_message(mosq, obj, msg):
    print("lights at "+msg.topic+" are "+msg.payload)

client = mosquitto.Mosquitto()
client.connect("localhost")
client.subscribe("switches/+/status")
client.on_message = on_message

while client.loop() == mosquitto.MOSQ_ERR_SUCCESS:
    pass
```

A simple example showing how to subscribe to a topic and define a function to receive the messages.

Eclipse Paho

Project providing open source implementations of C, C++, Java, JavaScript, Lua and Python client libraries, plus a client view plugin for Eclipse IDE.

http://www.eclipse.org/paho

Eclipse Paho

JavaScript MQTT client

```
client = new Messaging.Client( "127.0.0.1", 80, 'clientId' );

client.onMessageArrived = function( msg ) {
  console.log( "lights at " + msg.destinationName +
               " are " + msg.payloadString );
  client.disconnect();

};

client.connect({
  onSuccess:function() {
    client.subscribe( "switches/+/status" );
  }
});
```

Browser-based library that uses WebSockets to connect to an MQTT server.

To use MQTT over WebSocket, you'll need that server supports the WebSocket protocol, e.g. lighttpd with mod_websocket

Any projects making use of MQTT?

The house that Twitters

“Monitoring things such as how much power our house is using can give us valuable insights into the cost of various appliances in the home,” he said.

“Recently I was out and got a tweet saying water usage was higher than normal. I phoned home and my wife looked out of the window to see the garden hose had been left on.

“This can help us take steps to reduce our carbon footprint and reduce energy bills. Mine has dropped by a third in the last year.

Telegraph article

Facebook messenger

chat sessions, where users can join and leave a conversation, fit well with the publisher-subscriber model

Under the hood: Rebuilding Facebook for iOS

Smart Lab

Monitoring experiments at the University of Southampton’s chemistry lab.

Heart pacemakers

Send cardio data to doctors remotely monitoring at home patients

Other messaging protocols

AMQP

XMPP

請問有其他問題嗎?

謝謝