

Message Queuing Telemetry Transport

Implementierung einer IoT-Anwendung auf Basis von MQTT

Maximilian Gaul, Lukas Dorner

01.07.2019

- Idee, IoT-Projekte in 5 Layer aufzuteilen
 - **Objects Layer:** Physikalischer Sensoren und Aktuatoren, die verschiedene Funktionen übernehmen
 - **Object Abstraction Layer:** Transport der Daten zum nächsten Layer, z.B. über WiFi, Bluetooth
 - **Service Management Layer:** Verarbeitet und abstrahiert empfangene Daten bzw. die Hardwareplattform
 - **Application Layer:** Stellt den Anwendern die Daten zur Verfügung, die sie benötigen (z.B. Temperaturdaten)
 - **Business Layer:** Verwaltung der gesamten IoT-Applikation, Verwendung der Daten für z.B. *Big Data*

Paper

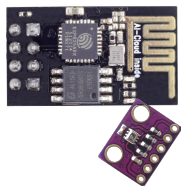
WiFi?, Bluetooth?, MQTT?, Temperatursensor?

- Viele verschiedene Hardware- und Protokollkombinationen denkbar

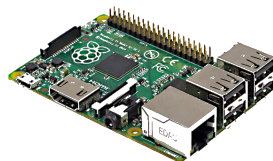
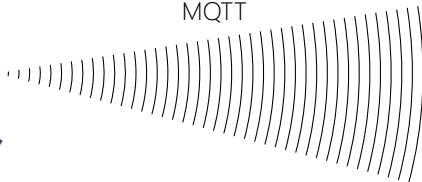
IoT Elements		Samples
Identification	Naming	EPC, uCode
	Addressing	IPv4, IPv6
Sensing		Smart Sensors, Wearable sensing devices, Embedded sensors, Actuators, RFID tag
Communication		RFID, NFC, UWB, Bluetooth, BLE, IEEE 802.15.4, Z-Wave, WiFi, WiFiDirect, , LTE-A
Computation	Hardware	SmartThings, Arduino, Phidgets, Intel Galileo, Raspberry Pi, Gadgeteer, BeagleBone, Cubieboard, Smart Phones
	Software	OS (Contiki, TinyOS, LiteOS, Riot OS, Android); Cloud (Nimbits, Hadoop, etc.)
Service		Identity-related (shipping), Information Aggregation (smart grid), Collaborative-Aware (smart home), Ubiquitous (smart city)
Semantic		RDF, OWL, EXI

Projektidee im IoT-Bereich

WiFi, MQTT, Temperatursensor



MQTT



ESP8266 + BMP280

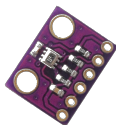
Raspberry Pi + Mosquitto

Projektidee im IoT-Bereich

WiFi, MQTT, Temperatursensor



- 32-Bit *RISC*-Controller, unterstützt 802.11 b/g/n mit bis zu 72.2Mbps
- 96 KByte RAM, 4 MB Flash
- Soft-I2C Anbindung an BMP280



- Temperatur- und Drucksensor
- 20-Bit Auflösung
- I2C-Interface



- Raspbian OS
- Mosquitto MQTT Broker & Subscriber

Projektumsetzung

Raspi & Mosquitto

Projektumsetzung

Raspi & Mosquitto

Besteht aus bis zu drei Teilen:

- **Fixed Header** - in allen MQTT Paketen vorhanden
 - Art des Paketes, ggf. QoS-Level
 - Verbleibende Länge im Paket (restlicher Variable-Header + Payload)
- **Variable Header**
 - Je nach Art des Paketes unterschiedlicher Inhalt
 - z.B. Topic, Verbindungs-Flags, Packet-Identifizier
- **Payload**
 - Ebenfalls verschiedener Inhalt je nach Paket möglich
 - z.B. Daten oder Topics, auf die man *subscriben* möchte

Projektumsetzung

Implementierung von MQTT-CONNECT & MQTT-PUBLISH auf ESP8266

ESP8266 verbindet sich mit MQTT-CONNECT zum Mosquitto-Broker auf Raspi

- Protokol-Name und Version
- Art der Verbindung
- Keep-Alive
- Client ID

```
MQ Telemetry Transport Protocol, Connect Command
  Header Flags: 0x10, Message Type: Connect Command
    0001 .... = Message Type: Connect Command (1)
    .... 0000 = Reserved: 0
  Msg Len: 25
  Protocol Name Length: 4
  Protocol Name: MQTT
  Version: MQTT v3.1.1 (4)
  Connect Flags: 0x02, QoS Level: At most once delivery (Fire and Forget), Clean Session Flag
    0... .... = User Name Flag: Not set
    .0.. .... = Password Flag: Not set
    ..0. .... = Will Retain: Not set
    ...0 0... = QoS Level: At most once delivery (Fire and Forget) (0)
    .... .0.. = Will Flag: Not set
    .... ..1. = Clean Session Flag: Set
    .... ...0 = (Reserved): Not set
  Keep Alive: 240
  Client ID Length: 13
  Client ID: ESP8266NETZE2
```

Projektumsetzung

Implementierung von MQTT-CONNECT & MQTT-PUBLISH auf ESP8266

ESP8266 sendet Temperaturdaten mit MQTT-PUBLISH an Mosquitto-Broker auf Raspi

- QoS-Flags
- Topic
- Payload (Temperaturdaten)

```
▼ MQ Telemetry Transport Protocol, Publish Message
  ▼ Header Flags: 0x30, Message Type: Publish Message, QoS Level: At most once delivery (Fire and Forget)
    0011 .... = Message Type: Publish Message (3)
    .... 0... = DUP Flag: Not set
    .... .00. = QoS Level: At most once delivery (Fire and Forget) (0)
    .... ...0 = Retain: Not set
  Msg Len: 9
  Topic Length: 3
  Topic: a/b
  Message: 0000bda
```

Demo