TECHNISCHE HOCHSCHULE DEGGENDORF

IAS Spezielle Protokolle des IoT

Message Queueing Telemetry Protocol

Motivation

Die Kommunikation im Application-Layer von IoT-Geräten unterscheidet sich von genereller Übertragung in folgenden Punkten:

- Die zu übertragenden Informationen sind vergleichsweise einfach strukturiert (Attribut-Wert-Paare)
- Die Geräte sind zeitweise offline und können dann keine Daten empfangen
- Verbindungsauf- und abbau erfordern (zu) viel Energie
- Die Protokolle müssen mit wenig Programm- und Datenspeicher auskommen
- Je nach Anwendung sind unterschiedliche Übertragungsqualitäten erforderlich (vom Temperatursensor zum Herzschrittmacher)

Protokollübersicht

Typische IoT Protokolle sind:

- REST (Representational State Transfer Application Programming Interface)
- CoAP (Constrained Application Protocol)
- MQTT (Message Queueing Telemetry Protocol)
- SMTP (Simple Mail Transfer Protocol)
- ...

MQTT

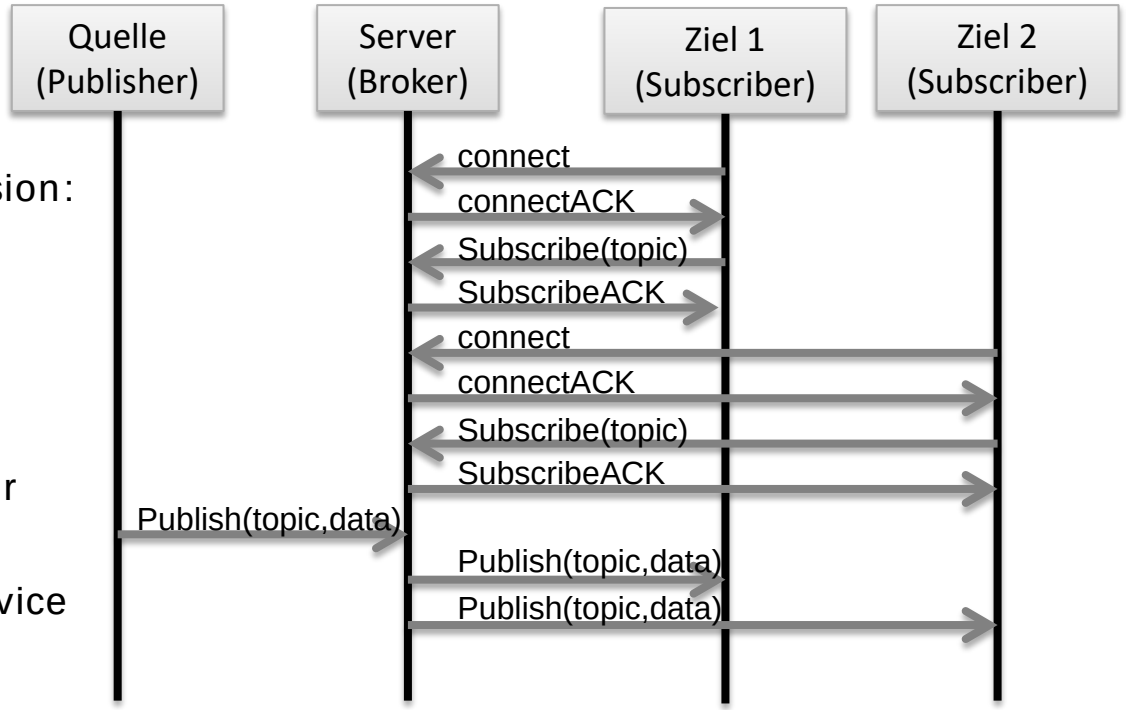
Geschichte

- 1999: MQTT von Dr. Andy Stanford-Clark (IBM) und Arlen Nipper (Arcom) entwickelt
- 2012: MQTT-SN Version 1.2
- 2014: MQTT wird von der OASIS (Organization for the Advancement of Structured Information Standards) standardisiert
- 2019: MQTT Version 5.0

MQTT Überblick

MQTT im Überblick

- Webseite: mqtt.org
- TCP-basierte Version: MQTT
- Sensornetzwerk-angepasste Version: MQTT-SN (auch: UDP)
- Publish/Subscribe Architektur
- Die Attribute sind in einer Baumstruktur (Pfad) organisiert
- Freie Wählbarkeit des Formats für Werte
- 3 unterschiedliche Quality of Service Levels



Hierarchische Gliederung

Topics werden hierarchisch in einer Baumstruktur gegliedert:

- Beispiel:
  /DIT/Donau/J/J009/Sector1/WindowSensor1/State
  /DIT/Donau/J/J009/Sector1/WindowSensor2/State
  /DIT/Donau/J/J009/Sector1/Heater1/TemperatureSensor1
  /DIT/Donau/J/J009/Sector1/Heater1/Thermostat
  /DIT/Donau/J/J008/Sector1/WindowSensor1/State
  /DIT/Donau/E/E005/Sector1/WindowSensor1/State
- Folgende Wildcards sind definiert:
  - „#“: Gesamter Teilbaum, z.B. /DIT/Donau/J/#
  - „+“: Beliebiger Wert in dieser Hierarchieebene, z.B. /DIT/Donau/+/Sector1/WindowSensor1/State
- Spezielle Topics:
  - $SYS: Enhält Informationen über den Netzwerkstatus, z.B. $SYS/broker/clients/total

Messages

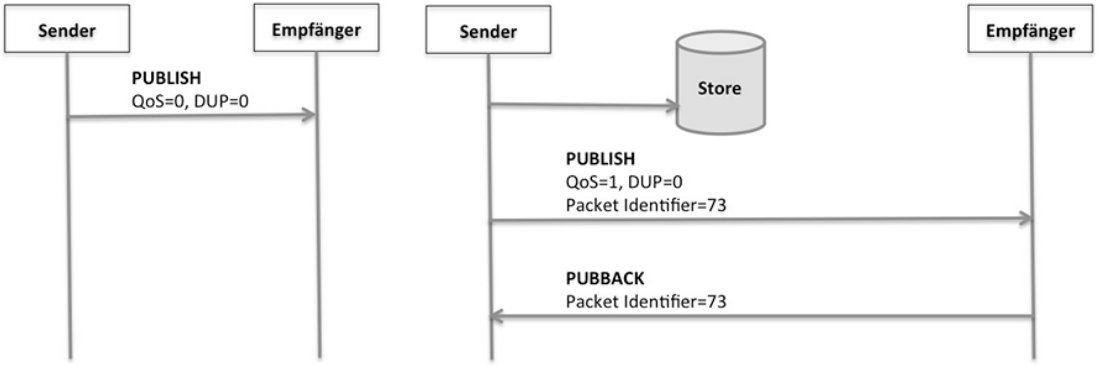
Das Nachrichtenformat (data) ist frei wählbar, typisch sind:

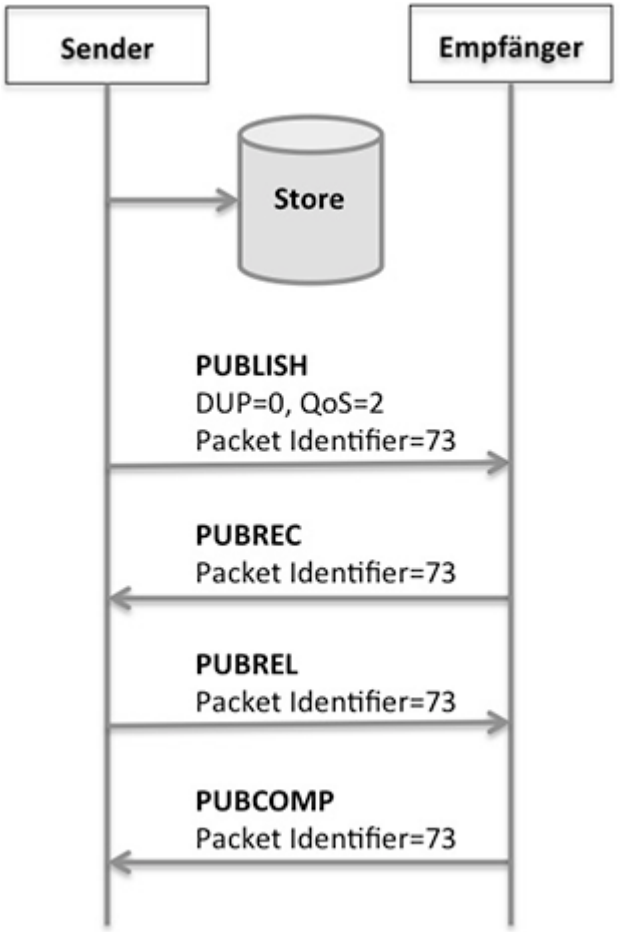
- Standardwerte (int, double)
- Komma-Separierte Listen
- JSON
- XML

Quality of Service (QoS)

Es gibt 3 Service Levels:

- QoS0: Die Nachricht wird höchstens einmal empfangen
  Beispiel: Regelmäßiger Temperatur-Broadcast
- QoS1: Die Nachricht wird mindestens einmal empfangen
  Beispiel: Temperatur auf 20Grad setzen
- QoS2: Die Nachricht wird genau einmal empfangen
  Beispiel: Temperatur um 2 Grad erhöhen





[Thomas Bayer, „MQTT, Das M2M und IoT Protokoll“, online: https://www.predic8.de/mqtt.htm]

Gespeicherte Nachrichten (retained)

Eine „retained“ Message dient dazu, neue Clients mit einer Information zu versorgen, z.B. dem letzten verfügbaren Temperaturwert

- Das „Retained“ Flag kann bei jeder Nachricht oder nur bei bestimmten Nachrichten gesetzt werden
- Der Server speichert für jedes Topic (nur) die letzte „Retained“ Nachricht ab
- Wenn ein neuer Client als Subscriber auftritt, wird diese letzte Nachricht zugesendet
- Retained Messages werden auch bei Wildcard-Subscribern versendet (# oder +)

Letzter Wille und Testament

Ein Client kann eine Message beim Server hinterlegen, die dieser im Falle eines Verbindungsabbruchs sendet (publish), um z.B. auf einen Ausfall eines Alarmsystems hinzuweisen

- Die Message wird direkt beim Verbinden konfiguriert, sie besteht aus Topic, QoS, Message und Retain-Flag
- Falls keine andere Aktivität stattfindet, sendet der Server in konfigurierbaren Intervallen eine Watchdog Message (2 Byte Ping)
- Mit MQTT Keep-Alive verpflichtet sich der Client in einem bestimmten Zeitintervall PINGREQ Messages zu senden, bei Ausbleiben wird von einem Verbindungsabbruch ausgegangen
- Die maximale Keep-Alive Zeit liegt bei etwas über 18 Stunden

Persistente Sessions

In persistenten Sessions speichert der Broker (fast) alle Informationen, während der Client offline ist

- Identifikation der Session geschieht über die Client-ID (auf unikate Client-ID achten)
- Gespeichert werden:
  - Alle Subscriptions des Clients
  - Alle QOS1 (at least once) und QOS2 (exactly once) Messages in Richtung des Clients, die noch nicht bestätigt sind
  - Alle QOS2 Messages vom Client, deren Empfang noch nicht vollständig ausgehandelt wurde
  - Auch der Client muss QOS1+2 Nachrichten speichern, die noch nicht vom Server bestätigt wurden

Authentifizierung und Verschlüsselung

MQTT ermöglicht Verschlüsselung:

- Transport Layer Security (TLS)

MQTT ermöglicht die Authentifizierung durch:

- Benutzername und Password (Klartext)

Neu in MQTT 5.0

Wichtige Neuerungen in MQTT 5.0:

- Fehlercodes, in denen die Gründe für eine Annahmeverweigerung angegeben werden können (CONNACK/PUBACK/PUBREC/PUBREL/UNSUBACK/DISCONNECT/SUBACK/AUTH)
- Nachrichtenflags als Metadaten im Header, z.B. für die Schlüsselauswahl bei Verschlüsselung oder zur Signalisierung von Payload als UTF-8 Message oder Binärmessage
- Shared Subscriptions, mit denen Clients sich selbst zu Gruppen zusammenfassen können. Wenn eine Message auf einem Topic gesendet wird, auf den ein solcher Cluster subscribed ist, empfängt nur einer aus der Gruppe diese Message.
- Message Expiry und Session Expiry: Konfigurierbare Zeitperioden, nach denen eine Message nicht weiter zugestellt wird oder eine Session als beendet angesehen wird
- Topic Alias: Um den Daten-Overhead weiter zu reduzieren kann ein (längeres) Topic, z.B. /wohnzimmer/sofa/lampe/farbe/rgb als Zahl angesprochen werden
- Will Delay, sendet den Letzten Willen nicht sofort, sondern erst nach einer konfigurierbaren Zeitspanne, um Fehlalarme zu verhindern
- Der Server kann dem Client bei der Verbindung Limitierungen vorgeben (Anzahl der Nachrichten, Nachrichtengröße, ...)

Vor- und Nachteile von MQTT

Vorteile von MQTT

- Push-Architektur: Der Client wird sofort bei Änderung informiert und muss nicht nachfragen
- Entkopplung: Sender und Empfänger müssen nur den Broker kennen, Sender und Empfänger können zu unterschiedlichen Zeiten aktiv sein
- Eingebaute Mechanismen gegen unzuverlässliche Übertragung, z.B. Wireless: Last Will und Testament, Retained Messages
- One-to-many und many-to-many Übertragung ermöglicht Transparenz
- Einfaches (lightweight) Protokoll mit geringem Implementierungsaufwand und Speicherkosten
- Binärprotokoll, kein textueller Overhead in den Headerstrukturen

Nachteile von MQTT

- Weniger geeignet für One-to-One (Point-to-point) Kommunikation, insbesondere Lastverteilungsaufgaben, in denen 1 Message an einen aus einer Auswahl von Clients gesendet werden muss
- Keine synchrone Kommunikation
- Keine Vorgaben für typische Übertragungsdaten, z.B. Timestamp, Priorität, ...
- Single point of Failure durch Broker

REST

REST – Representational State Transfer

- Auf dem HTTP-Protokoll aufbauend
- Request-Response basiert
- URI Adressierung
- REST baut auf existierenden Standards auf, ist aber selbst kein Standard
- Vielzahl von hilfreichen Erweiterungen, siehe z.B. https://standards.rest/

REST

Aufbau von REST

- Basierend auf bekannter Web-Terminologie definiert REST als zentrales Element die Ressource
  - Eine Ressource bestimmt ein oder mehrere abfragbare Informationen
  - Eine Ressource ist durch den Pfad (URI) eindeutig spezifiziert
  - Teil-Ressourcen können durch tiefere Navigation im URI-Pfad aufgerufen werden, z.B. http://example.com/haus/wohnzimmer (gibt Informationen über das Wohnzimmer zurück) und http://example.com/haus/wohnzimmer/licht/an
- Zum Abfragen, Setzen und Ändern von Ressourcen nutzt REST bekannte HTTP Befehle:
  - GET: Zum Abfragen von Informationen aus statischen oder dynamischen Ressourcen
  - POST: Zur Erstellung von dynamischen Ressourcen (z.B. …/wohnzimmer/lichtstimmung mit neuer Ressource „romanLesen“)
  - PUT: Zum Verändern oder Erstellen von Ressourcen (z.B. …/wohnzimmer/lichtstimmung/fernsehen)
  - PATCH: Ändern eines Teils einer Ressource, andere Teile bleiben unangetastet (z.B. …/wohnzimmer/lichtstimmung/fernsehen mit Information „lampe3=aus“)
  - DELETE: Zum Löschen von Ressourcen

REST

Aufbau von REST Anfragen

- Es muss immer eine gültige URI angegeben werden
  z.B. http://example.com/get/timetable/today
- Das Format der URI ist frei wählbar
- Parameter können in der URI codiert werden (sog. Query-String Parameter)
  z.B. …timetable/today?format=JSON&currentHour=10

Inhalt von REST Anfragen

- Die Übertragung von Informationen mit dem REST Protokoll nutzt häufig
  - Einfache, unstrukturierte Daten (z.B. Temperaturabfrage)
  - XML-Format, insbesondere bei komplexen, unbekannten Datenformaten
  - JSON-Format, insbesondere bei komplexen aber vorab definierten Datenformaten

REST

REST Software Service Levels

- REST ist weit verbreitet, man unterscheidet zwischen 3 Qualitätsstufen:
  - 1. Stufe: Es wird nur ein Typ von HTTP Anfragen unterstützt, meist entweder POST oder GET, häufig bei Embedded-Geräten anzutreffen
  - 2. Stufe: Es werden alle Typen von HTTP Anfragen unterstützt, Statuscodes werden korrekt zurückgeliefert (z.B. 404 Not Found), Create-Read-Update-Delete (CRUD) Funktionalität ist häufig für Ressourcen implementiert
  - 3. Stufe: Zusätzlich verweisen Ressourcen auf andere Ressourcen (Hyperlinks, Hypermedia), z.B. um weitergehende Informationen zum Abruf zu referenzieren oder URI für bestimmte Aktionen vorzugeben (z.B. Löschen eines Datensatzes durch Aufruf der spezifizierten URI)

Vor- und Nachteile von REST

Vorteile von REST:

- Zustandsloses Design: Alle Informationen werden in der Nachricht selbst übermittelt – Service-Handover und Service-Verteilung werden erleichtert
- Bestimmte häufige Message-Typen können von Client und Server in einem Cache vorgehalten werden
- Die Business-Logic wird vom Design entkoppelt (Model / View)
- Resourcen, Services und Server sind getrennte Entitäten und können ausgetauscht werden
- Plattform- und Sprachunabhängigkeit der Schnittstellen-Implementierung

Nachteile von REST:

- Es wird kein festes, konfigurierbares Antwort-Schema vorgegeben
- Häufig textuelles Protokoll mit viel Overhead (z.B. XML)
- PULL-basiert, d.h. Änderungen an Zuständen müssen erst aktiv vom Client erfragt werden → der Client stellt auch viele nutzlose Anfragen, wenn sich nichts geändert hat

REST universell

Hypermedia As The Engine Of Application (HATEOAS):

- Durch die Hypertext und Hyperlink Fähigkeit von REST kann die Navigation vollständig in die Daten verlagert werden:
  - Der Client fragt die oberste REST-Ebene an, die auf tieferliegende Informationen verweist
  - Die Darstellung liegt beim Client
  - Die Navigation und die Daten liegen beim Server
- Inhalt und Aufbau von Datenstrukturen (z.B. « Adresse ») sind auf dem Server gespeichert in standardisierter Form

GraphQL

GraphQL als Alternative zu REST

- GraphQL ist formaler als REST, indem zunächst die Daten und ihre Repräsentation in festen Schemata definiert werden
- Bei Anfragen kann der Client angeben, welche Daten aus einem schematisierten Datensatz gesendet werden sollen und welches Format diese Daten haben sollen
- Veränderungen von Daten werden durch den Aufruf (sog. Mutationen) von Funktionen auf dem Server realisiert. Die verfügbaren Funktionen können als Funktionsköpfe abgefragt werden
  Beispiel Schema:
  ```
  type Mutation {
          getTemperature(location: String): Float
  }
  ```
  Ein Teil der Verarbeitungskomplexität kann dadurch transparent auf den Server verschoben werden
- Validierungstests lassen sich automatisch durchführen, z.B. mittels GraphiQL

TECHNISCHE HOCHSCHULE DEGGENDORF

Danke für Ihre Aufmerksamkeit!