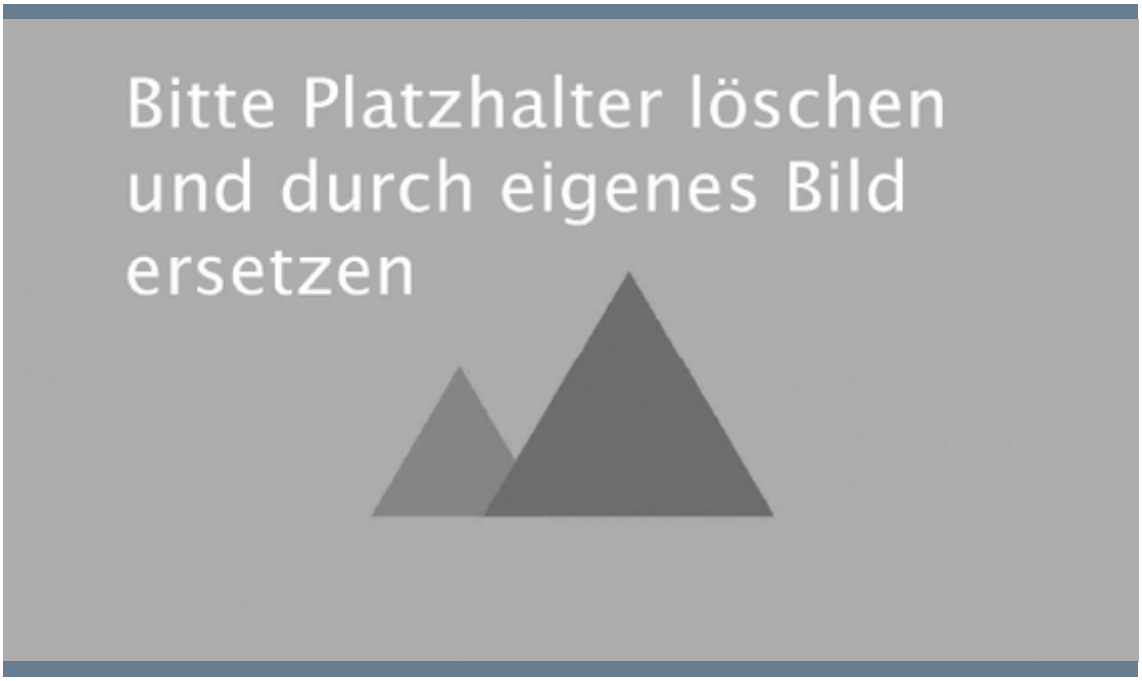

Titel des Berichts

Hier steht ein Untertitel

Bachelorthesis

Studiengang:	Informatik
Autor:	Adrian Bärtschi
Betreuer:	Prof. Dr. Ing. Reto E. Koenig
Experte:	Dr. Federico Flueckiger
Datum:	23.08.2015

Versionen

Version	Datum	Status	Bemerkungen
0.1	23.08.2015	Entwurf	Dokument erstellt

1. Management Summary

- Was ist das Ziel der Thesis
- Kurzbeschrieb des Themas
- Was wurde gemacht, erreicht

Inhaltsverzeichnis

Management Summary i

1. Management Summary i

2. Einleitung 1

3. Project Managament 3

3.1. Organisation . . . 3
3.2. Meilensteine . . . 3
3.3. Resourcen . . . 3
3.4. Ziele . . . 3
3.5. Aufwände . . . 4
3.6. Deliverables / Termine . . . 4

4. Technologie 5

4.1. MQTT . . . 5
4.2. Bestehende Konzepte . . . 8
4.3. Datenbeschreibung . . . 10
4.4. Konzept aus der nicht-IoT Welt . . . 11

5. Konzept 13

5.1. Allgemein . . . 13
5.2. Hierarchie Topics . . . 14
5.3. Device Description . . . 16
5.4. Format . . . 16
5.5. Datentypen . . . 16
5.6. Schema . . . 16

6. Umsetzung 17

6.1. Tinkerforge Teil . . . 17
6.2. MQTT Teil . . . 17
6.3. Thing Description . . . 17

7. Schlussfolgerungen/Fazit 19

Selbständigkeitserklärung 21

Acronyms 23

Literaturverzeichnis 25

Abbildungsverzeichnis **27**

Tabellenverzeichnis **29**

Stichwortverzeichnis **31**

A. Arbeitsjournal **31**

2. Einleitung

Bei Systemen im Internet of Things (IoT) Umfeld sind sehr viele und auch unterschiedliche Geräte in einen Netzwerk miteindander verbunden. Für diese IoT - Machine-To-Machine Kommunikation werden andere Netzwerkprotokolle eingesetzt als im 'klassischen' Internet. Dies ist nötig, weil die Geräte stark eingeschränkte Resourcen haben und die Netzwerke geringe Bandbreiten aufweisen.

MQTT ist ein Protokoll, welches die Anforderungen für IoT Systeme erfüllen soll. Beim Entwurf des Protokolls wurde auf Einfachheit und Leichtgewichtigkeit grossen Wert gelegt. Mit MQTT ist es möglich, beliebige Daten in beliebiger Codierung zu versenden. Dies bietet den Entwicklern der Systeme grosse Freiheiten.
Es ist aber ersichtlich, dass die fehlende Struktur und Beschreibung der Daten gewisse Schwierigkeiten mit sich bringen kann. Eine Anwendung, welche Daten per MQTT erhält, muss wissen wie diese vom Absender codiert wurden und was sie bedeuten.

Beispielsweise wird eine Messung eines Temperatursensors via MQTT versendet werden. Der Sensor liefert den Wert 22° Celsius. Der ganzzahlige Wert 22 wird als binärer Wert 10110 versendet. Der Empfänger erhält nun die MQTT Nachricht 10110. Er hat aber keine Ahnung, was mit diesem Wert anzufangen ist. Wird dieser Wert als Fliesskommazahl interpretiert (Float), so wird der Wert in 3.0828 konvertiert. Der Empfänger müsste wissen, dass es sich um einen Integer Wert (hier 32 bit signed) handelt, damit die Daten in das richtige Format gebracht werden können. Ausserdem muss der Empfänger jetzt noch wissen, dass in welcher Einheit (Celsius, Fahrenheit, etc.) die Temperatur übermittelt wird.

Um diese Information vom Anbieter der Daten resp. des Diestes oder Geräts den Entwicklern einer Anwedung zur Verfügung zu stellen, werden zurzeit Beschreibungen in Dokumentenform verwendet. Diese sogenannte out-out-band Dokumentation ist aber aufwändig in der Nachführung und bekanntegabe von Änderungen. Auch ist es schwierig die Struktur der Daten einheitlich und klar zu erklären.

Ziel dieser Thesis ist es, dass Geräte welche ihre Daten per MQTT versenden, die Möglichkeit erhalten, sich selbst inkusive ihrer Daten und Möglichkeiten zur Interkation zu beschreiben. Dies soll so getan werden, dass die Beschreibung für Mensch und Maschine les- und verstehbar ist, die Eigenschaften der eingeschränkten Geräte und Netze berücksichtig wird und der MQTT Standard weiterhin eigehalten wird.

3. Project Managament

3.1. Organisation

TODO: formatieren

Name	Rolle	Aufgaben
Adrian Bärtschi	Studierender	Selbständiges Projektmanagement während der Thesis. Setzt die Aufgaben gemäss Aufgabenstellung und Vorgaben Betreuer um. Organisiert Kommunikation mit Betreuer und Experte.
Prof. Dr. Ing. Reto E. König Berner Fachhochschule	Betreuer	Hauptansprechsperson für Studierenden, verantwortlich für den Ablauf der Thesis. Beurteilung aufgrund von Aufgabenstellung und abgegebenen Artefakten.
Dr. Federico Flueckiger Eidg. Finanzdepartement	Experte	Beurteilung aufgrund der Aufgabenstellung und abgelieferten Artefakten sowie mindestends ein bis zwei Sitzungen mit dem Studierenden.

Tabelle 3.1.: Involvierte Personen und deren Aufgaben

3.2. Meilensteine

Datum	Meilenstein	Bemerkungen
25.09.2015	Initialisierung, Vorgehen geklärt	-
01.10.2015	Ziele definiert	-
15.10.2015	Prototyp erstellt, Konzept der Lösung skizziert	-
30.11.2015	Implementation System abgeschlossen	-
15.12.2015	Implementation Demo Applikation abgeschlossen	-
18.01.2016	Dokumentation inhaltlich abgeschlossen	-
21.01.2016	Dokumentation fertiggestellt	-

Tabelle 3.2.: Meilensteine der Thesis

3.3. Resourcen

Kosten, etc.

3.4. Ziele

Die zu entwickelnde Lösung soll folgendes beinhalten:

- asdf

3.5. Aufwände

Task	Soll	Ist	Bemerkungen
asdf	2	2	df

Tabelle 3.3.: Planung der Tasks und Auswetung der Aufwände

3.6. Deliverables / Termine

Datum	Thema
	Abgabe Texte/Grafiken/etc. für das Book
	Abgabe Elektronische Version des Posters
21.01.2016	Abgabe Projektdokumentation an Betreuer, Experte, Sekretariat
22.01.2016	Final Day Bern, Präsentation und Ausstellung
	Verteidigung

Tabelle 3.4.: Termine und Fristen

4. Technologie

4.1. MQTT

Die folgende Einführung des MQTT Protokolls wurde aus der dem Ergebnis des BFH Modul BTI7302 (Projekt 2) von Adrian Bärtschi übernommen.

MQTT (Message Queue Telemetry Transport) ist ein Netzwerkprotokoll, das sich dank einfachem und leichtgewichtigen Design sehr gut für Geräte mit stark eingeschränkten Ressourcen und Netzwerke mit geringer Bandbreite eignet.
Die erste Version von MQTT wurde 1999 von Dr. Andy Stanford-Clark (IBM) und Arlen Nipper (Arcom) beschrieben und entwickelt. Inzwischen ist MQTT in der Version 3.1.1 verfügbar [6] und wird von OASIS Konsortium (`https://www.oasis-open.org`) standardisiert.

Im OSI Modell ist MQTT auf dem Application Layer eingeordnet, basierend auf dem TCP Stack.

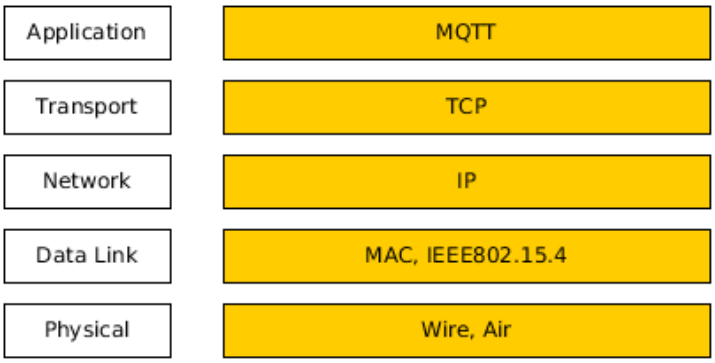

Abbildung 4.1.: MQTT im (vereinfachten) OSI Stack

4.1.1. Publish/Subscribe

MQTT funktioniert nach dem Publish/Subscribe Pattern. Im Gegensatz zum klassichen Client/Server Prinzip registrieren sich die Clients (Subscriber) bei einem Broker für bestimmte Bereiche (Topics), zu denen sie Nachrichten erhalten möchten.

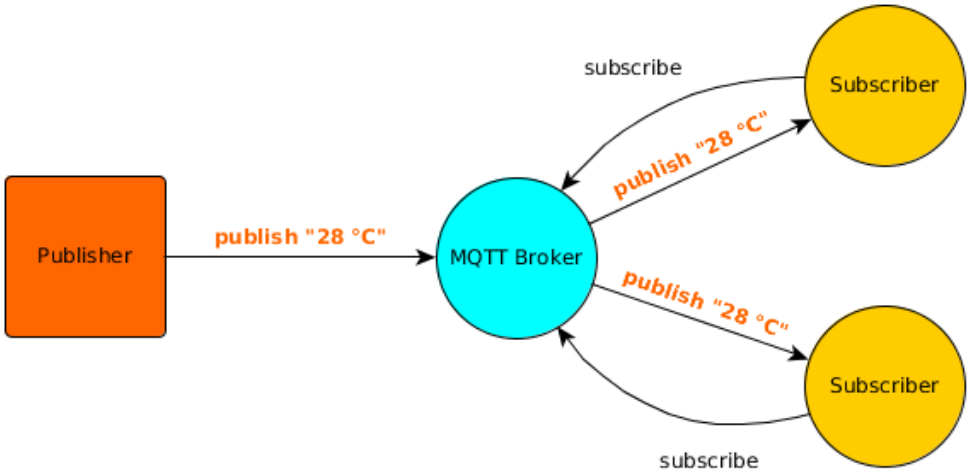

Abbildung 4.2.: Publish/Subscribe Prinzip

Ein Publisher, (z. Bsp. ein Sensor) sendet seine Nachrichten an den Broker. Alle Subscriber, die sich für das entsprechende Topic eingeschrieben haben, erhalten die Nachricht vom Broker.

Diese Entkopplung der Teilnemhner bringt diverse Vorteile mit sich:

- Publisher und Subscriber müssen sich gegenseitig nicht kennen
- Clients können sich beliebig an- und abmelden
- Beim Ausfall eines Teilnehmers sind die anderen nicht blockiert

4.1.2. Topics

Jede Nachricht wird an ein bestimmtes Topic gesendet. Grundsätzlich kann jeder Client zu jedem Topic des Brokers Nachristen veröffentlichen. Die Topics sind hierarchisch aufgebaut, die Ebenen werden durch einen Slash (/) getrennt.
Zum Beispiel könnte ein Thermometer seine Temperatur im Topic `house/livingroom/temperature` veröffentlichen.

Um die Nachrichten zu erhalten, muss sich nun ein anderer Client für dieses Topic einschreiben. Dabei können die Wildcards + und # verwendet werden.
Das + Symbol steht für eine Ebene in der Topic Hierarchie.
Beispiel: `house/+/temperature` steht für die Temperaturmeldungen aller Räume des Hauses .

- `house/livingroom/temperature`
- `house/kitchen/temperature`
- `house/bedroom/temperature`
- `...`

Mit # werden alle Topics des Unterbaums abonniert. Beispiel: Wenn ein Client das Topic `house/#` abonniert, erhält dieser alle Meldungen des Hauses.

- `house/bedroom/temperature`
- `house/door/status`
- ...

4.1.3. Messages

Messages in MQTT sind sehr einfach aufgebaut. Eine Message hat folgende Attribute:

Attribut	Beschreibung
Payload	Beliebige Daten im Binärformat. Maximal 256 MB
QoS	Quality of Service 0, 1 oder 2. Details in Kapitel 4.1.4
Retained	Flag, true oder false, Details in Kapitel 4.1.5

Tabelle 4.1.: Aufbau einer MQTT Message

4.1.4. Quality of Service

MQTT bietet drei verschiedene QoS Einstellungen für das versenden von Nachrichten.

- 0: Die Nachricht wird einmal versendet, es gibt keine Bestätigung des Empfängers.
- 1: (Standard) Die Nachricht wird mindestens einmal beim Empfänger ankommen. Es wird so lange versucht zu senden, bis eine Bestätigung erhalten wurde.
- 2: Mit ein Handhshake Mechanismus wird sichergestellt, dass die Nachricht genau einmal beim Empfänger angekommen ist.

Je höher die QoS Einstellung, desto mehr Resourcen werden beim versenden benötigt und es desto mehr Daten werden über das Netzwerk gesendet.

4.1.5. Retaines Messages

Bei einer MQTT Message kann das Retained Flag gesetzt werden. Damit wird pro Topic die letzte Nachricht auf dem Broker gespeichert. Verbindet sich ein neuer Client und abonniert das Topic, erhält er die retained Message sofort. Dies kann nützlich sien bei Anwendungen mit Topics, die sehr lange Pausen zwischen den Meldungen haben.

4.1.6. Last Will

Jeder Client, der sich zum Broker vebindet, kann eine "Last Will" Message angeben. Diese Meldung wird an das gewünschte Topic gesendet, sobald die Verbindung beendet wird.

4.1.7. Implementationen

Inzwischen sind zahlreiche Implementationen von sowohl MQTT Brokern, als auch Client Libraries verfügbar.

Broker

Name	Beschreibung	URL
Mosquitto	Leichtgewichtiger Open Source Broker geschrieben in C. Zurzeit de-facto standard MQTT Broker.	`http://mosquitto.org/`
ActiveMQ Apollo	OSS Message Broker, untersützt neben MQTT noch andere Protokolle. Konfiguration und Administration über Web GUI.	`http://activemq.apache.org/apollo/`
Moquette	OSS Broker Implementation in Java. Kann auch als Library in eigene Projekte eingebunden werden.	`https://github.com/andsel/moquette`
HiveMQ	Proprietärer MQTT Broker, ausgerichtet für Enterprise Anwendungen.	`http://www.hivemq.com/`
Mosca	MQTT Broker für die NodeJS Platform. Kann als Standalone oder Embedded Broker verwendet werden.	`http://www.mosca.io/`

Client Libraries Das Eclipse Projekt Paho [2] stellt Client Libraries für C, Java, Android, Python, Javascript, C/C++ embedded und .Net / WinRT zur verfügung. Die Java Library sich als stabil und gut dokumentiert erwiesen.

Eine Lister mit weiteren Client Libraries ist im MQTT Wiki [5] zu finden.

4.2. Bestehende Konzepte

Die verschiedenen Hersteller von MQTT Anwendungen entwickeln jeweils ihre eigenen Ansätze, um die Daten zu strukturieren.

4.2.1. IBM Internet of Things Foundation

IBM hat unter dem Brand 'IBM IoT Foundation' [3] einen Dienst entwickelt, mit dem vernetzte Geräte verwaltet werden können. Als Kommunikationsprotokoll wird MQTT eingesetzt. Die Plattform verwendet folgende konzeptionelle Ideen:

- Organizations: Eindeutige Identifikation der Kunden der Plattform
- Devices: Beliebiges vernetztes Gerät. Versendet Events und reagiert auf Commands.
- Applications: Anwendung, welche mit den Daten der Devices interagiert.
- Events: Daten, welche von den Devices an die Plattform gesendet werden
- Commands: Applications können mittels Commands mit den Devices kommunizieren.

Events

Events müssen an ein definiertes Topic nach folgendem Schema gesendet werden:
`iot-2/evt/<event_id>/fmt/<format_string>`

Beispiel: `iot-2/evt/temperature_outdoor/fmt/json`

Eine Anwendung, welche Events empfangen möchte, muss sich auf ein Topic in der Form `iot-2/type/<device_type>/id/<device_id>/evt/<event_id>/fmt/<format_string>` registrieren. Die Teile `device_type`, `device_id`, `event_id` und `format_string` des Topics können auch mit dem Wildcard Charakter '+' ersetzt werden, um jeweils alle Events der Komponenten zu erhalten.

Beispiel: `iot-2/type/temp/id/+/evt/temperature_outdoor/fmt/+`

Commands

Um einen Command zu erzeugen, sendet eine Anwendung eine MQTT Message mit Topic gemäss folgenden Schema: `iot-2/type/<device_type>/id/<device_id>/cmd/<command_id>/fmt/<format_string>`

Beispiel: `iot-2/type/temp/id/sensor1/cmd/setInterval/fmt/json`

Das Device `sensor1` würde damit eine Message auf Topic `iot-2/cmd/setInterval/fmt/json` erhalten.

Payload Format

Grundsätzlich unterstützt IBM IoT Foundation ein beliebiges Payload Format. Es wird jedoch empfohlem, JSON zu verwenden. Um alle Funktionen der Platform nutzen zu können, müssen die JSON Dokumente zusätzlich nach den Vorgaben [4] von IBM strukturiert sein.

```
{
  "d": {
    "host": "IBM700-R9E683D",
    "mem": 54.9,
    "network": {
      "up": 1.22,
      "down": 0.55
    },
    "cpu": 1.3,
  }
}
```

Listing 1: JSON Beispiel im IBM IoTF Payload Format

4.2.2. Tinkerforge MQTT Proxy

Tinkerforge hat ein modulares System von Sensoren und Aktoren (so gennate Bricklets) entwickelt, die u. A. für Prototyping und in der Ausbildung (auch an der BFH) eingesetzt werden. Um die Mudule zu steuern, wird klassischerweise das bereitstellete SDK in der gewünschten Programmiersprache verwendet. Tinkerforge ausserdem eine Anwendung entwickelt und die Bausteine per MQTT ansprechen zu können [1].

Topics

Die Tinkerforge Devices senden ihre Daten an ein MQTT Topic nach Schema `tinkerforge/<prefix>/<uid>/<suffix>`.

Ein Temperatur Bricklet mit Unique Identifier (UID) `xf2` würde also den gemessenen Wert an das Topic `tinkerforge/bricklet/temperature/xf2/temperature` senden.

Die Bricklets reagieren auf Messages die an ein passendes Topic mit Siffix `/set` gesendet werden. Sollen beispielsweise die LEDs des Dualbutton Bricklets mit UID `mxg` eingeschaltet werden, muss eine Message an das Topic `tinkerforge/bricklet/dual_button/mxg/led_state/set` gesendet werden mit folgendem Payload:

```
{
    "led_l": 2,
    "led_r": 2
}
```

Listing 2: JSON Beispiel Tinkerforge Format

Payload Format

Die Tinkerforge MQTT Komponente verwendet JSON als Datenformat für die Messages. Jede Message, die von einem Bricklet gesendet wird, enthält unter dem Key `_timestamp` den Zeitpunkt der Erzeugung als UNIX Timestamp.

```
{
  "_timestamp": 1440083842.785104,
  "temperature": 2343
}
```

Listing 3: JSON Beispiel Tinkerforge Format

Die Beschreibung, unter welchen Topics Daten publiziert werden und wie die Bricklets angesprochen werden können, ist in der Dokumentation von Tinkerforge [1] beschrieben.

4.3. Datenbeschreibung

4.3.1. JSON Schema

4.3.2. Protocol Buffer

4.3.3. DFDL

Aus Proj2 übernehmen

4.3.4. Vergleich

4.4. Konzept aus der nicht-IoT Welt

SOAP-WSDL, REST, etc. HATEOAS

5. Konzept

5.1. Allgemein

Gateway
Die einzelnen Sensoren eines Internet of Things (IoT) Systems sind an einen Gateway angeschlossen. Dies kann fix per Kabel oder über eine Drahtlosschnittstelle wie z. Bsp. Bluetooh umgesetzt sein. Typsicherweise werden Kleincomputer (Single Board Computer) wie ein Raspberry Pi oder Intel Galileo eingesetzt. Diese Geräte zeichen sich aus durch sehr kompakte Bauform (Kreditkatenformat) und bieten trotzdem genügend Ressourcen um darauf hardwareunahängige Anwendungen (z. Bsp. Java, Python, Javascript) laufen zu lassen. Ein Gateway führt die Daten der einzlenen Sensoren zusammen und vereinheiltliche die Formate der Daten. Ausserdem übernimmt er die Kommunikation mit der Aussenwelt, indem er eine Verbindung zum zentralen Broker herstellt.

MQTT-Broker
Der Broker ist bei MQTT Anwendungen von zentraler Bedeutung, da alle Daten via Broker zwischen den Teilnehmern ausgetauscht werden. Pro System gibt es typischeise einen Broker. Dieser muss für alle Teilnehmener (Gateways und Applikationen) einfach zu erreichen sein und muss eine hohe Verfügbarkeit aufweisen.

Application
Es können grundsätzlich beliebige Applikationen auf allen gängigen Plattformen und Technologien mit den System kommunizieren. Die einzige Grundbedingung ist, dass die Applikation MQTT fähig ist, was sich durch das Einbinden der entsprechenden Libraries einfach realisieren lässt.

Bei der Beschreibung eines einzelnen Devices werden folgende drei Bereiche unterschieden:

State
Die Menge aller Eigenschaften des Devices und deren Werte wird als State bezeichnet. Beispielsweise Name, Firmwareversion, etc. Der State eines Devices ist beständig, d.h. solange keine Interaktion geschieht, verändert sich der State nicht.

Events
Tritt auf dem Device ein Ereignis ein, wird dadurch ein Event erzeugt. Ein Event wird grundsätzlich vom Device selbst ausgelöst. Beispielsweise erzeugt ein Temperatursensor alle 5 Sekunden ein Event welches den Messwert enthält. Applikationen registrieren sich, um bestimmte Events von den Devices zu erhalten.

Commands
Eine Anwendung interagiert mit einem Device, indem Commands an eines oder mehrere Devices gesendet werden. Die Devices empfangen die Commands und reagieren entsprechend. Ein Command ist bestimmt durch einen Namen und Parameter mit dazugehörigen Werten. Beispielsweise kann bei einem Temperatursensor über einen Command

`SetInterval: 10s` der Abstand der Messungen eingestellt werden. Ein Device muss bekanntgeben, auf welche Commands mit welchen Paramtern es reagiert.

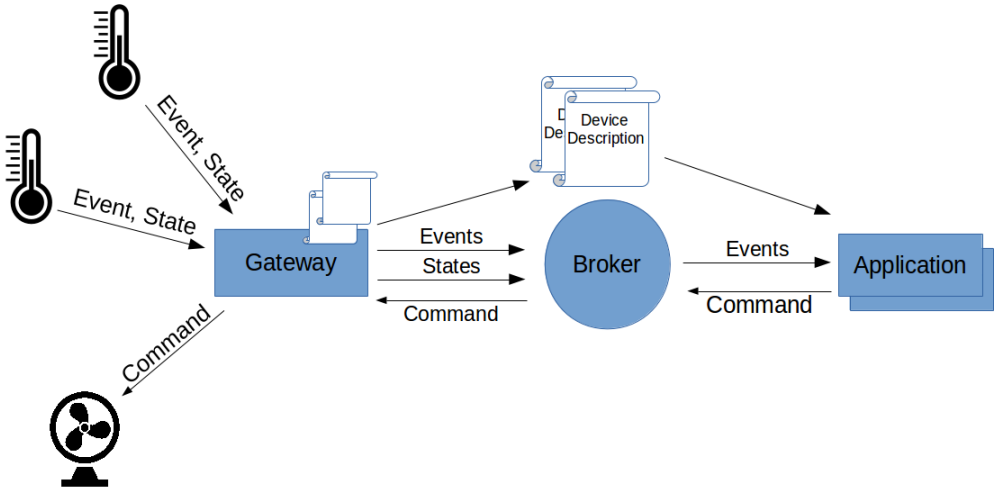

Abbildung 5.1.: Übersicht der

5.2. Hierarchie Topics

Die Topic Hiearachie wird nach folgenden Muster aufgebaut:

Level	Beschreibung	Beispiel
0	**Identifikation Anwendung** Eindeutige identifikation der Anwendung.	ch.bfh.barta3.myApp
1	**Master Host** Gruppierung der Devices auf Stufe 1	-
2	**Host** Gruppierung der Devices auf Stufe 2	-
3	**Device Typ** Bezeichnung, um was für einen Typ von Sensor oder Aktor es sich handelt.	Temperatursensor
4	**Device ID** Da mehrere Devices vom selben Typ im Einsatz sein können, wird auf dieser Stufe mit einer eindeutigen ID der konkrete Sensor resp. Aktor angegeben.	mxg
4	**Device Description**	-
5	**Kategorisierung Devicedaten** Die Eigenschaften und Daten werden in die nachfolgenden drei Teile aufgegliedert.	Events, State, Commands
6	**Events**	Temperatur
6	**State**	Interval
6	**Commands**	setInterval

Tabelle 5.1.: Topic Hierarchie

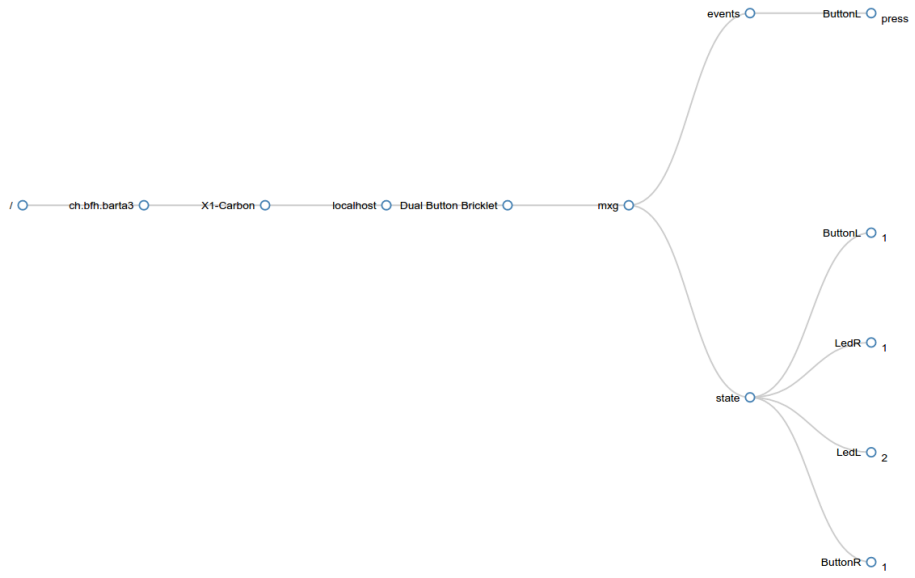

Abbildung 5.2.: Visualisierung MQTT Topics Tinkerforge IR Temperatur Sensor

TODO: better diagram

5.3. Device Description

5.3.1. Codierung

UTF-8

5.4. Format

Json, Yaml

5.5. Datentypen

atomare Typem (Interger, Long etc.)

5.6. Schema

Tabelle mit Felder und Beschreibung

6. Umsetzung

6.1. Tinkerforge Teil

Enumeration

6.2. MQTT Teil

Topic Tree

6.3. Thing Description

7. Schlussfolgerungen/Fazit

Selbständigkeitserklärung

Ich bestätige mit meiner Unterschrift, dass ich meine vorliegende Bachelor-Thesis selbständig durchgeführt habe. Alle Informationsquellen (Fachliteratur, Besprechungen mit Fachleuten, usw.) und anderen Hilfsmittel, die wesentlich zu meiner Arbeit beigetragen haben, sind in meinem Arbeitsbericht im Anhang vollständig aufgeführt. Sämtliche Inhalte, die nicht von mir stammen, sind mit dem genauen Hinweis auf ihre Quelle gekennzeichnet .

Ort, Datum: Bern, 23.08.2015

Namen Vornamen: Bärtschi Adrian

Unterschriften: .....

Acronyms

IoT Internet of Things. 1, 13

MQTT Masdf. 13

UID Unique Identifier. 10

Literaturverzeichnis

[1] Brick MQTT Proxy – Tinkerforge. Accessed: 2015-11-02. [Online]. Available: http://www.tinkerforge.com/en/doc/Software/Brick_MQTT_Proxy.html

[2] Eclipse Paho Projekt. Accessed: 2015-11-15. [Online]. Available: http://www.eclipse.org/paho/

[3] IBM Internet of Things Foundation. Accessed: 2015-11-02. [Online]. Available: https://internetofthings.ibmcloud.com

[4] Message Payload — IBM IOT Foundation 1.0 documentation. Accessed: 2015-11-02. [Online]. Available: https://docs.internetofthings.ibmcloud.com/messaging/payload.html

[5] MQTT Client Libraries. Accessed: 2015-11-01. [Online]. Available: https://github.com/mqtt/mqtt.github.io/wiki/libraries

[6] MQTT Specification Version 3.1.1. Accessed: 2015-11-02. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html

Abbildungsverzeichnis

4.1. MQTT im (vereinfachten) OSI Stack 5
4.2. Publish/Subscribe Prinzip 6

5.1. Übersicht der 14
5.2. Visualisierung MQTT Topics Tinkerforge IR Temperatur Sensor 15

Tabellenverzeichnis

3.1. Involvierte Personen und deren Aufgaben 3
3.2. Meilensteine der Thesis 3
3.3. Planung der Tasks und Auswetung der Aufwände 4
3.4. Termine und Fristen 4

4.1. Aufbau einer MQTT Message 7

5.1. Topic Hierarchie 15

A.1. Arbeitsjournal 31

A. Arbeitsjournal

Woche	Datum	Arbeiten
1	14.09.	Kickoff-Vorlesung BFH Bernhard Anrig Projektinitialisierung, Aufsetzen Dokumentation auf sharelatex.com Besprechung mit Reto König betreffend Ablauf, Organisatorisches, Ziele
2	21.09.	Projektplanung Dokumentation Projektmanagement, Dokumentation Einleitung
3	28.09.	Recherche bestehende Konzepte Einarbeitung Tinkerforge Bausteine Aufsetzen Prototyp Tinkerforge
4	05.10.	Einlesen bestehende Technologien (Coap, LWM2M, RESTful API Dokumentation) Besprechung mit Reto König, skizzieren des Prototyps
5	12.10.	Prototyp Tinkerforge Temperatursensor mit automatischer Enumeration/Erkennung, Versenden der Werte per MQTT Einbindung MQTT Topic Tree Webapp für bessere Übersicht
6	19.10.	Übernahme der Prototyp Ergenisse in neues Projekt, Refactoring Server Setup DigitalOcean. Installation Docker, Mosquitto Broker und Apache Webserver Analyse IBM IoT Foundation
7	26.10.	Einbinden von Tinkerforge DualButton und Joystick Probleme mit Online Dokumentation auf sharelatex.com, Aufsetzen und Einrichtung lokale Latex Umgebung Umstrukturierung der Applikation Erzeugung Device Description in JSON Einführung einheitliches Logging
8	02.11.	Besprechung mit Federico Flueckiger (Einführunhg in Thema, Termin Verteidigung) Ansatz für generische Einbindung der Tinkerforge Bricklets mittels Reflection des Java APIs. Besprechung mit Reto König betreffend Termin Verteitigung, Poster, Book
9	09.11.	Generischer Reflection Ansatz verworfen, nicht praxistauglich Organisation Termin für Verteidigung Verfeinerung Device Description
10	16.11.	Dokumentation Konzept, Architektur Besprechung mit Reto König betreffend TODO
11	23.11.	Start Webapplikation für Device Description Anzeige
12	30.11.	Besprechung mit Reto König betreffend TODO
13	07.12.	Commands
14	14.12.	Poster für Finalday BFH Book Page
15	11.01.	Trennung Library und Tinkerforge-Demo Git Repo Reorg
16	18.01.	Commands

Tabelle A.1.: Arbeitsjournal