Fachmodulbericht

Fernwartung und Ferndiagnose von „intelligenten" Geräten

Herbstsemester 2015/2016

Autoren:

Manuel Werder
Felbenstrasse 4
9469 Haag
manuel.werder@ntb.ch

Sandro Hobi
Brunnenackerstrasse 18
9437 Marbach
sandro.hobi@ntb.ch

Referent:

Prof. René Pawlitzek
rene.pawlitzek@ntb.ch

Koreferent:

Prof. Dr. Andreas Zogg
andreas.zogg@ntb.ch

Industriepartner:

Triopan AG
Säntisstrasse 11
9400 Rorschach
071 844 1616

Alexander Kleger
ak@triopan.ch

Computechnic AG
Rietlistrasse 3
9403 Goldach
071 858 2666

Martin Zünd
martin.zuend@computechnic.ch

TRIOPAN computechnic NTB Interstaatliche Hochschule für Technik Buchs

Abstract

Das Internet der Dinge (IoT) erfährt gerade einen enormen Aufschwung. Es werden in naher Zukunft alle Geräte, oder eben „Dinge", einen Anschluss zum Internet haben. Somit wissen wir zu jeder Zeit ob der Postbote das Paket schon in den Briefkasten gelegt hat, oder wie viele Packungen Milch noch im Kühlschrank, bzw. im Vorratsraum sind. Die Waschmaschine per Smartphone einschalten? Kein Problem! Das riesige Einsatzgebiet des IoT's ist praktisch nur durch die eigene Vorstellungskraft begrenzt.
Diesem Trend folgt auch die Firma Triopan AG in Rorschach. In einem ersten Schritt statten sie ihre Aufpralldämpfer mit einer intelligenten Elektronik aus. Diese bereits vorhandene Hardware sammelt bestimmte Daten wie GPS-Position, Fahrtrichtung, Temperatur und Luftfeuchtigkeit. Zuletzt detektiert sie wenn es am Aufpralldämpfer zu einem solchen Aufprall kommt. Über das GSM-Modul werden diese Daten versendet.
Für die Übertragung der Daten wurden drei Protokolle genau untersucht: HTTP, WebSockets und MQTT. Letztere zwei sind sich dabei ähnlich. Beide sind auf TCP/IP aufgebaut. Trotz eines grossen Headers bei der Übermittlung seitens HTTP, ist es über einen langen Zeitraum gesehen der Gewinner wenn es um das verbrauchte Datenvolumen geht. Denn dies bestimmt nicht zuletzt auch einen Teil der anfallenden Kosten.
Beim Datenvolumen sind sich die WebSockets und MQTT ebenfalls sehr ähnlich. Beide Protokolle werden in Betracht zum aufgebrauchten Datenvolumen durch den TCP-Keep-Alive bestimmt. Dieser stellt in regelmässigen Zeitabständen fest ob die Verbindung noch besteht.
Jedoch überwiegen die Nachteile von HTTP gegenüber MQTT und WebSockets. Wird die Elektronik eingeschaltet erhält sie evtl. jedes Mal eine neue IP. Somit muss sich das Gerät immer zuerst registrieren indem es angibt welche IP es gerade zugeteilt bekommen hat. Ebenfalls besteht keine konstante Verbindung zwischen Gerät und Webapplikation.
Bei den WebSockets wird eine bidirektionale Verbindung zwischen Client und Server aufgebaut. Steht diese Verbindung können beide Seiten senden und empfangen. Jedoch müssen auch beide Seiten zur selben Zeit online sein. Um möglichst energiesparend zu sein wird die Elektronik jedoch nicht ständig online sein sondern regelmässig in einen Low-Power-, oder Sleep-Modus gehen. Zudem kann auch der Empfang abbrechen wenn der Aufpralldämpfer beispielsweise durch einen Tunnel fährt.
Beim MQTT-Protokoll ist das gar kein Problem. Denn zwischen Client und Server steht ein sogenannter Broker. Dieser empfängt alle Nachrichten und leitet sie anhand Ihres Topics weiter zum Endempfänger. Somit ist es der Webapplikation egal wenn das Gerät gerade nicht online ist. Sie sendet Ihre Daten an den Broker und sobald die Elektronik wieder online geht, holt sie sich alles was für sie bestimmt ist beim Broker ab, und umgekehrt.
Da das MQTT-Protokoll speziell für Machine-to-Machine (M2M)–Anwendungen entwickelt wurde bietet es auch viele nützliche Optionen an. So kann über den „Quality of Service" (QoS) eine Übermittlung wichtiger Daten garantiert werden. Des Weiteren lassen sich die oben genannten TCP-Keep-Alives einfach verzögern oder abstellen. Ebenfalls kann bei Netzverlust eine auf dem Broker vordefinierte Funktion ausgeführt werden. Somit lässt sich ohne grossen Aufwand, auf beinahe jede Situation eine Lösung finden.
Durch diese gegebenen Vorteile des MQTT Protokolls ist dieses, in Gebrauch von IoT-Anwendungen, ganz klar den WebSockets vorzuziehen.

Nicht zuletzt müssen diese Daten in einer Webapplikation gesammelt und visualisiert werden. Damit sich der Entwickler nicht selbst um die Wartung eines Webservers kümmern muss, wird in diesem Projekt auf eine sogenannte „Plattform as a Service" (PaaS) gesetzt. Die Google App Engine ist eine solche PaaS. Die Software lässt sich wie auf einen Webserver einfach per Mausklick hochladen. Je nachdem wie viele Requests der Webserver gerade verarbeiten muss, stellt Google automatisch mehr Rechenleistung zur Verfügung. Man spricht hierbei von Skalierung.

Inhalt

1 Einleitung

Dieser Bericht wurde im Rahmen des Fachmoduls der Bachelorarbeit «Ferndiagnose und Fernwartung von „intelligenten" Geräten mit dem Internet of Things (IoT)» erarbeitet. Er dient als Einarbeitung in das Thema, und soll grob angeben, in welche Richtung die Arbeit verlaufen soll. Das heisst es können schon Entscheidungen getroffen werden, welche einen Einfluss auf das weitere Vorgehen haben.

Die Firma Triopan AG in Rorschach ist ein Hersteller von verschiedenen Produkten für die Strassensignalisierung. Von Faltsignalen bis zu kompletten Warnwänden mit Aufpralldämpfer für Fahrzeuge ist alles im Sortiment. In einem nächsten Schritt sollen diese mobilen Geräte „intelligent" werden. Konkret heisst das, sie sollen mit einer kleinen elektronischen Leiterplatte ausgestattet werden, welche es ermöglicht, eine Internetverbindung aufzubauen. So können verschiedene Informationen, z.B. Position, Temperatur oder gegebenenfalls Unfälle (beim Aufpralldämpfer), direkt und schnell ins Internet gesendet werden, wo die Daten von einer Webapplikation ausgewertet werden.

1.1 Aufgabenstellung

Die Leiterplatte welche für diese Anwendung eingesetzt werden soll besteht bereits. Sie kann aus einer vorherigen Bachelorarbeit übernommen werden. Diese wurde von Marc Figliuolo von der Firma Computechnic AG in Goldach erarbeitet. Dabei wurde auch schon ein wenig Software entwickelt, welche Daten ans Internet schickt.
Folgende Punkte sind Aufgabe des Fachmoduls:

- Einarbeitung in das Gebiet des „Internet of Things" (IoT), durch Literaturstudien
- Einarbeitung in die Google App Engine (GAE)
- Einarbeitung in die von der Firma Computechnic AG vorgegebene Hard- und Software
- Evaluation geeigneter Kommunikationsprotokolle (HTTP, WebSocket, MQTT)
- Erstellung einer ersten Systemarchitektur

2 Internet of Things (IoT)

Das Internet of Things, oder im deutschen das Internet der Dinge ist heutzutage in aller Munde. Doch was genau versteckt sich dahinter?

2.1 Einleitung

Die Vernetzung des Internets hat in den letzten Jahren stark zugenommen. Smartphones, Tablets oder herkömmliche Computer haben dazu beigetragen, dass man heutzutage praktisch immer mit dem Netz verbunden ist. Doch dies war erst der Anfang. Beim IoT spricht man schon von einer globalen Wirtschaftsbewegung, welche diese Vernetzung auf ein ganz neues Level bringen will. In Zukunft werden Toaster, Blumentöpfe, Kühlschränke, ja einfach alles eine Verbindung zum Internet besitzen. Die alltäglichsten Dinge werden Informationen und Daten sammeln und diese verwenden um unser Leben möglichst zu vereinfachen.

Aus einem Bericht der NZZ:[1]

„Der Elektroauto-Hersteller Tesla sammle über eingebaute Sensoren 1,6 Gigabyte Daten pro Monat von jedem Fahrzeug. Tesla stellte fest, dass Kunden in der Schweiz Hilfe beim Anfahren am Berg gebrauchen könnten. Die Firma entwickelte ein Software-Update, testete es ein paar Tage und überspielte es dann über Nacht auf alle Tesla-Fahrzeuge weltweit. Am nächsten Morgen hatten die Tesla-Kunden einen neuen Knopf auf ihrem digitalen Armaturenbrett."

Die Anzahl der Geräte mit Internetanschluss wird in Zukunft rasant ansteigen. Die Schätzungen darüber gehen recht weit auseinander. Es ist von ca. 10-35 Milliarden Geräten die Rede, welche bis zum Jahr 2020 neu ans Netz angeschlossen werden sollen. Mit einem solchen Zuwachs an intelligenten Gegenständen wird auch ein dementsprechend riesiger Datenaustausch erwartet. Die Kommunikationen und Interaktionen werden mit neuen Applikationen auf ein bislang undenkbares Level angehoben.
In naher Zukunft wird es nicht mehr ungewöhnlich sein, wenn man per Smartphone vom Blumentopf erfährt, dass dessen Erde etwas trocken ist und man doch etwas Wasser nachgiessen sollte. Die Möglichkeiten sind immens, und eigentlich nur durch die Fantasie der Entwickler begrenzt.

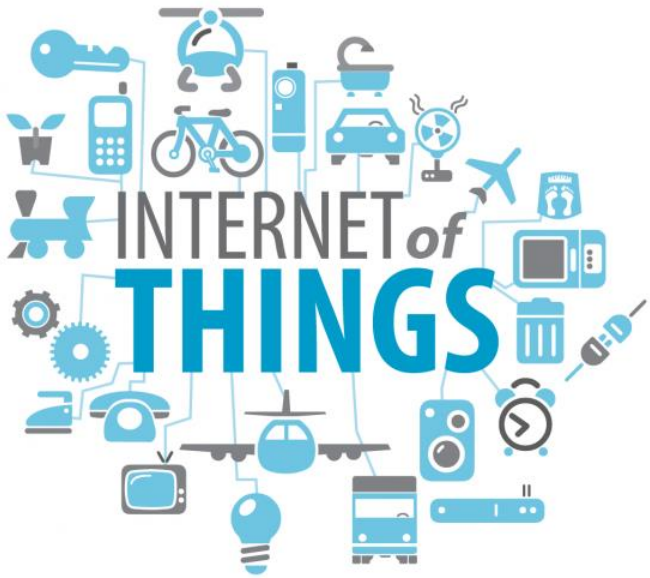

Abbildung 1: Internet of Things[2]

[1] Quelle: (Sander, 2015)
[2] Quelle: (Hsieh, Abbott, & Jazayeri, 2015)

2.2 Reference Model[1]

Ein IoT-System unterscheidet sich ein wenig von den herkömmlichen Geräten, welche mit dem Internet verbunden sind.

- Viele Geräte, welche unabhängig vom Menschen Daten ins Internet senden
- Oft begrenzte Rechenleistungen, was eine andere Kommunikation mit dem Netz benötigt
- Menge an generierten Daten übersteigt alles bisherige

Für solche Systeme wurde also ein neues Reference Model erstellt, mit welchem die einzelnen Schichten genauer erläutert werden können.

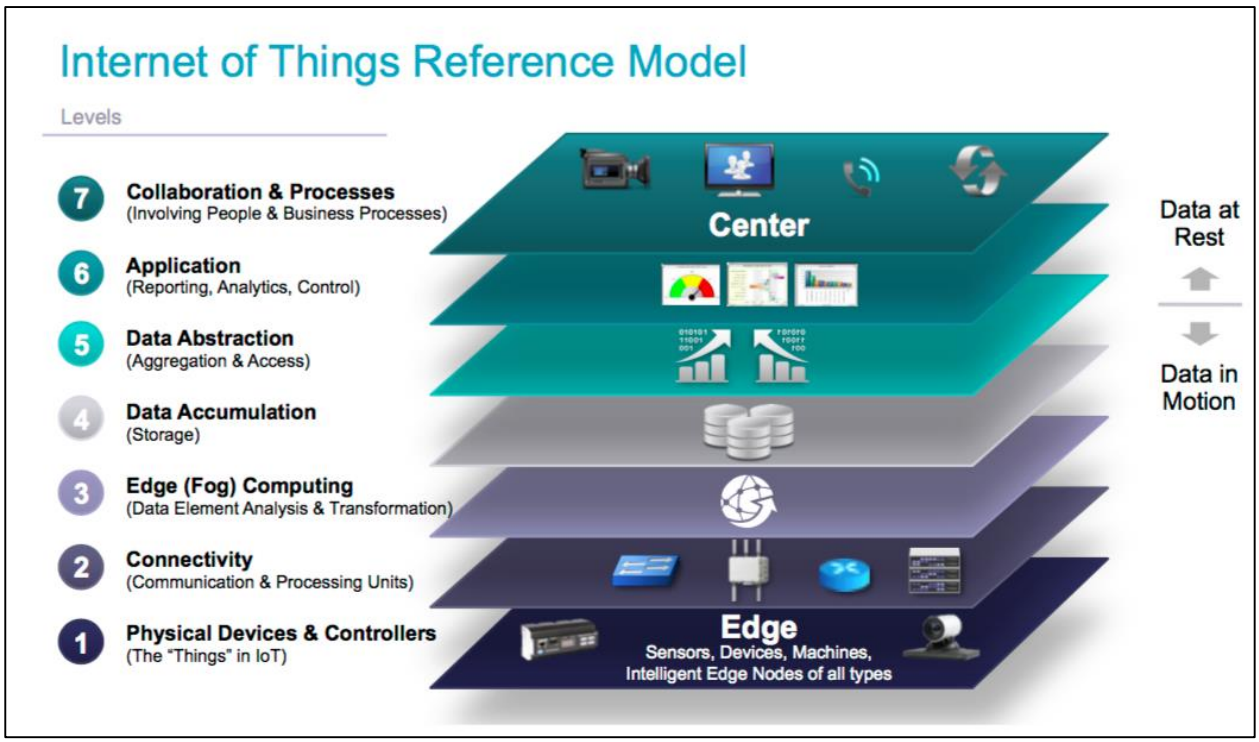

Abbildung 2: Internet of Things Reference Model

2.2.1 Level 1: Physical Devices

In der untersten Schicht befinden sich die Sensoren und Controller der Geräte oder Maschinen. Hier werden die Daten generiert, welche später über die anderen Levels versendet werden. Man spricht auch von einem sogenannten „Endpoint-Device", also sozusagen das eine Ende der Kommunikation.

2.2.2 Level 2: Connectivity

Die zweite Schicht liegt zwischen dem Device und dem Netzwerk. Sie ist für die Verbindung zum Internet verantwortlich. Das beinhaltet die Datenübertragung zwischen:

- Level 1 Devices und dem Netzwerk
- verschiedenen Level 1 Devices
- zwischen dem Netzwerk und Level 3

Dazu gehört die Implementierung eines oder mehreren Übertragungsprotokolle. Damit sollen die Daten zuverlässig versendet und auch empfangen werden.

[1] Quelle: (Cisco, 2015)

2.2.3 Level 3: Edge (Fog) Computing

Die dritte Schicht in diesem Reference Model wandelt die Daten in ein geeignetes Format um, sodass sie von der nächst höheren Schicht abgespeichert werden können. Sie ist also darauf fokussiert den ganzen Datenstrom vom Device zu analysieren und so zu formatieren, dass die Informationen von der Anwendung verarbeitet werden können.

- Falls nötig, zusätzlichen Kontext an Daten hängen, oder Teile entfernen
- Daten zusammenfassen

2.2.4 Level 4: Data Accumulation

Bis zu dieser Schicht sind die vom Device generierten Daten „in motion". Das heisst, sie werden mittels einem Übertragungsprotokoll in Netzwerk geschickt, ohne zu wissen ob der Endanwender die Daten wirklich braucht. Die Daten sind in Bewegung. In dieser Schicht werden die Daten nun „in rest" gesetzt. Das heisst hier werden die Informationen abgespeichert, sie sind also nachher fest abgelegt und in Ruhe. Level 4 bestimmt also:

- ob die Daten für eine höhere Schicht von Interesse sind, dann werden sie abgespeichert
- welche Daten auf einem nicht-flüchtigen Speicher gespeichert werden müssen, und welche nur für den kurzzeitigen Gebrauch nötig sind
- was für ein Typ von Datensystem eingesetzt wird (Bsp. Datenbank, File-System)

2.2.5 Level 5: Data Abstraction

Hier soll der Zugriff auf die abgespeicherten Daten abstrahiert werden. Es kann gut sein, dass bei einer IoT-Anwendung verschiedene Datenspeichertypen zum Einsatz kommen, und oder mehrere Speicher desselben Typs. Es muss also eine Schnittstelle zwischen sie und die Anwendung implementiert werden, um einen einfachen Datenzugriff zu gewährleisten. Dazu gehört:

- Vereinheitlichen von verschiedenen Datenformaten von verschiedenen Quellen
- Konsistent Semantik der Daten
- Schützen der Daten mittels Authentifizierung und Autorisierung
- Schnellen Datenzugriff gewährleisten

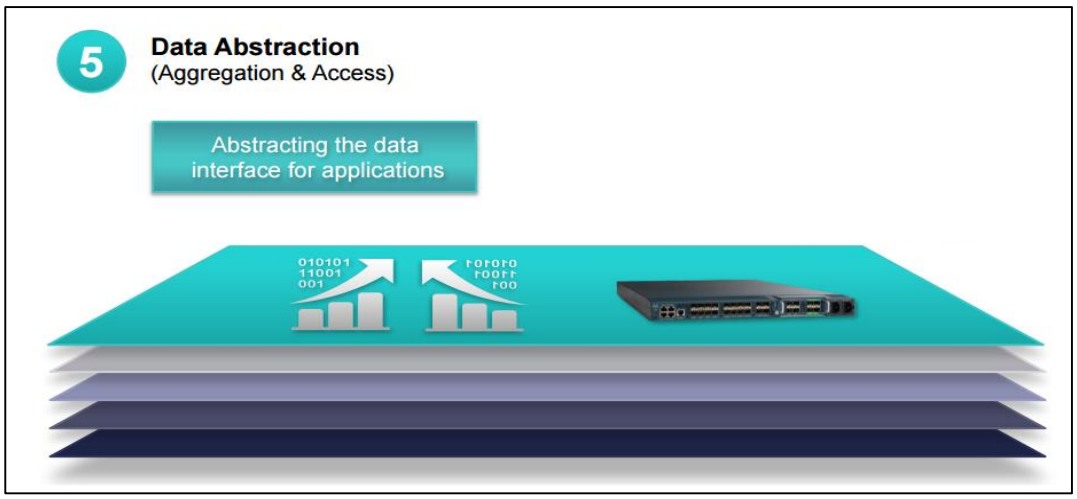

Abbildung 3: Level 5, Data Abstraction

2.2.6 Level 6: Application

Wie der Name schon sagt, wird in dieser Schicht die eigentliche Applikation implementiert. Über den Data-Abstraction Layer wird auf die gespeicherten Daten zugegriffen. Hier muss nicht mehr mit Netzwerkgeschwindigkeit gerechnet werden.
Die eigentliche Anwendung ist sehr spezifisch und unterscheidet sich je nach Markt stark. Beispielsweise können hier:

- Informationen visualisiert werden
- Devices gesteuert werden

Es kann sich um eine Mobile-Applikation handeln, die beispielsweise auf einfache Interaktionen reagiert. Eine komplexe Business-Anwendung die riesige Datenmengen auswertet und Statistiken erstellt ist ebenso denkbar.

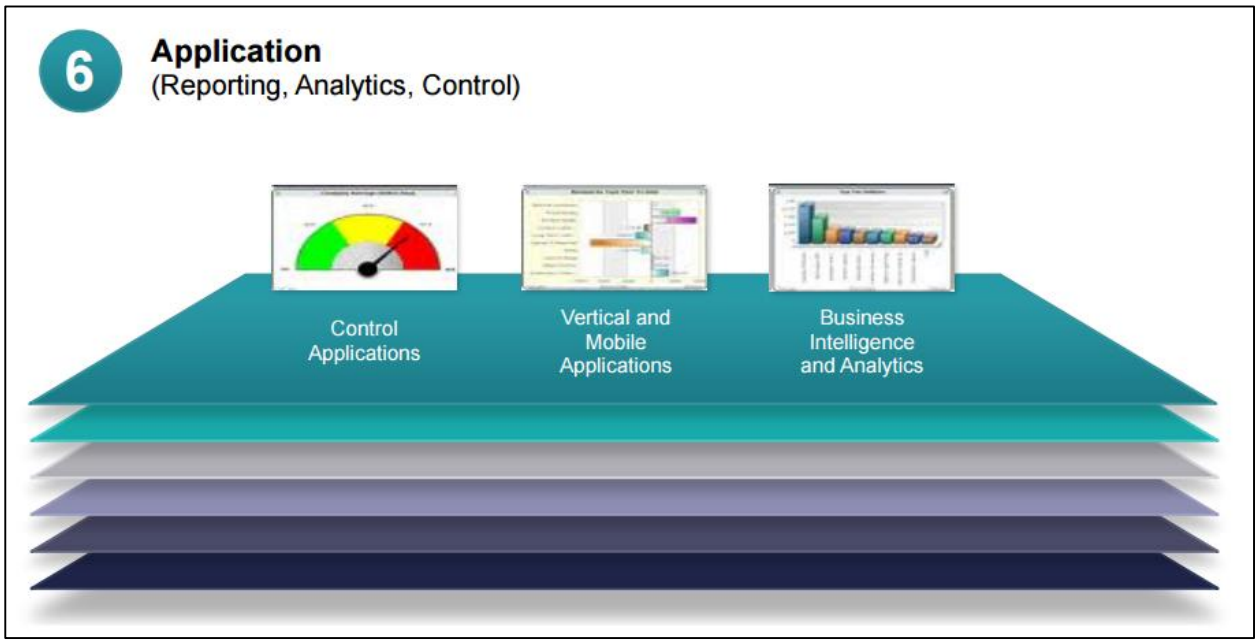

Abbildung 4: Level 6, Application

2.2.7 Level 7: Collaboration and Processes

Beim Internet der Dinge werden oft Personen oder Prozesse mit eingebunden. Die erzeugten Daten sind nur etwas wert, wenn sie richtig verarbeitet werden.
Anwendungen führen Geschäftsprozesse durch um Mitarbeiter zu ermächtigen. Sie werden von Arbeitern für ihre speziellen Bedürfnisse gebraucht. Allgemein sollten die Leute unterstützt werden ihre Arbeit besser erledigen zu können.
Anwendungen (Level 6) geben den Geschäftsleuten die richtigen Daten zum richtigen Zeitpunkt. Was daraus gemacht wird, also welche Entscheidungen getroffen werden, was für Zusammenarbeiten nötig sind und welche Prozesse eingeleitet werden, ist Teil der obersten Schicht.

3 Google App Engine

Die Google App Engine, kurz GAE, ist eine Plattform für Entwickler von Webanwendungen. Sie gehört somit zum Bereich der PaaS, den sogenannten Platform-as-a-Service. Als PaaS wird eine Computerplattform in der Cloud genannt. Sie dient Entwicklern von Webanwendungen dazu mit geringerem Aufwand und ohne Anschaffung von Hardware eine Webapplikation zu schreiben. Webanwendungen lassen sich so unkompliziert auf die Server von Google laden. Der Entwickler muss sich somit weder um die Beschaffung und Wartung der Server kümmern. Zusätzlich muss er sich keine Gedanken über die Dimensionierung der Server machen. Die Webserver von Google skalieren automatisch, je nachdem wie viel Rechenleistung gerade benötigt wird.
Dazu wird eine Vielzahl von nützlichen Diensten angeboten.

3.1 Programmiersprachen

Google hat angekündigt, dass die GAE entwickelt wurde um sprachunabhängig zu sein. Deshalb wird neben Phyton und Java auch Go und PHP unterstützt. In Zukunft sollen zudem noch mehr Programmiersprachen unterstützt werden.

3.2 Datenspeicher

Die GAE unterstützt diverse Datenspeicher, die wichtigsten sind unter anderem:

3.2.1 Google Cloud SQL

Hierbei handelt es sich um eine MySQL Datenbank im Inneren der Google Cloud. Dabei unterstützt sie nicht nur alle Funktionen von MySQL, sondern hat auch noch ein paar Erweiterungen.

3.2.2 App Engine Datastore.

Ein NoSQL Datenspeicher für Webanwendungen. Dieser Datenspeicher hält Objekte in Form von Einträgen. Ein Eintrag hat einen oder mehrere Eigenschaften. Somit unterscheidet er sich stark von der Tabellenstruktur eines SQL Datenspeichers.
Ein grosser Vorteil gegenüber der SQL Cloud ist dabei die Skalierbarkeit. Während der App Engine Datastore automatisch skaliert, muss sich der Entwickler bei der SQL Cloud selbst um die Skalierung kümmern.

3.2.3 Datastore Backup/Restore

Dadurch lassen sich Einträge vom Datastore wiederherstellen. Dieser Dienst befindet sich jedoch noch in der Beta.

3.3 Kosten

Die anfallenden Kosten sind sehr übersichtlich. Man muss sich nicht für ein Abo entscheiden, sondern zahlt lediglich was man braucht. Anfallende Kosten sind zum Beispiel eine benötigte Instanz pro Stunde und der benötigte Datenverkehr. Die laufenden Kosten sind jederzeit im Google App Engine Konto einsehbar.

4 Kommunikationsprotokolle

4.1 HTTP[1]

Das HTTP (Hypertext Transfer Protocol) ist im Bereich des World-Wide-Web das meist verwendete Protokoll zum Laden von Webseiten.

4.1.1 Allgemein

Mit dem HTTP-Protokoll werden vorwiegend Daten aus einem Browser (Internet Explorer, Google Chrome etc.) geladen, um eine Webseite darin darzustellen. Es handelt sich um ein sogenanntes zustandsloses Protokoll. Jede Anfrage wird eigenständig betrachtet, es kann kein Zusammenhang zu vorherigen Anfragen gemacht werden.

Application	**HTTP**
Transport	**TCP**
Internet	**IP (IPv4, IPv6)**
Network Access	**Ethernet etc.**

Tabelle 1: Protokollstapel HTTP

4.1.2 Funktionsweise

Der Client sendet eine Anfrage, ein HTTP-Request, an einen Webserver. Dafür wird eine TCP-Verbindung aufgebaut. Anschliessend antwortet der Server mit einer Response. Diese enthält immer einen Status-Code, mit welchem mitgeteilt wird, ob die Anfrage richtig verstanden wurde. Zusätzlich werden die angeforderten Daten mitübertragen. Nach dieser Nachricht wird die Verbindung wieder getrennt.

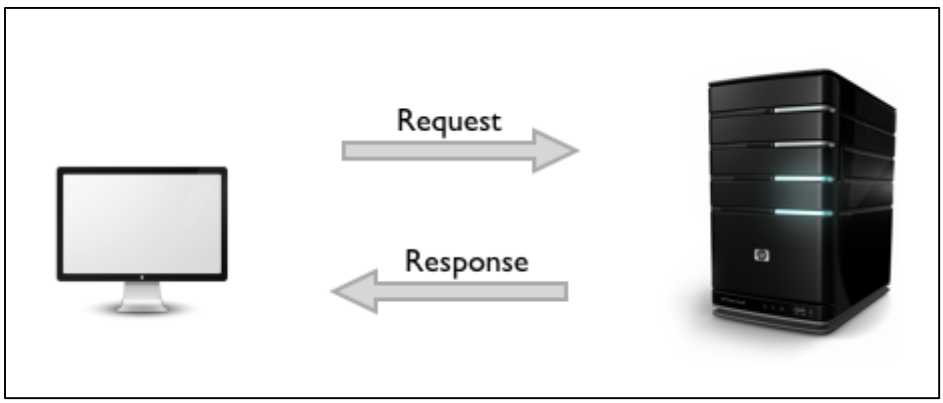

Abbildung 5: Funktionsweise HTTP-Protokoll

4.1.3 Aufbau

HTTP ist ein typisches Client-Server Protokoll. Wie man in Abbildung 5 sehen kann schickt ein Client einem Server eine Anfrage, und erhält darauf eine Antwort. Danach ist die Kommunikation abgeschlossen bis die nächste Information angefordert wird.

4.1.3.1 HTTP-Request

So kann eine typische Anfrage eines Clients aussehen:

```
GET /homepage.html HTTP/1.1
Accept: text/html, image/jpeg, image/gif, */*
Accept-Charset: ISO-8859-1
User-Agent: Mozilla/5.0 (Windows; Win 9x 4.9; de-DE;
rv:1.7.12) Gecko/20050919 Firefox/1.0.7
Host: www.beispiel.de
```

[1] Quelle: (Jens Franke, 2015)

Eine Anfrage beginnt immer mit einer ***Request-Line***. Diese beinhaltet folgende Informationen:

- Methode (Siehe 4.1.3.1.1)
- Ziel-URL: sie gibt an welche Webseite oder Datei aufgerufen werden soll (hier: „homepage.html") In einem Nachfolgenden Header muss der genaue Host angegeben werden, da heutzutage für eine IP-Adresse mehrere Hostnamen verwendet werden können
- HTTP-Version: z.B. „HTTP/1.1"

Jetzt folgen die ***Request-Header***, hier können zusätzliche Informationen bezüglich der Anfrage an den Server übermittelt werden. Beispielsweise können folgende Header angehängt werden:

- Accept: gibt an, welche Dateiformate der Client akzeptiert
- Accept-Charset: Zeichenkodierung, die der Client verstehen kann
- Host: gibt den genauen Zielhost an (hier: „www.beispiel.de") Falls der Port vom Standard HTTP-Port (80) abweicht, muss dieser ebenfalls angegeben werden

4.1.3.1.1 Request Methoden

Hier werden die meist verwendeten Methoden kurz erläutert.

GET
Diese wird verwendet, wenn ein Client Daten von einem Server anfordert.

POST
Ähnlich wie die GET-Methode, zusätzlich können aber Daten übermittelt werden. Daten könnten auch bei einem GET übertragen werden (als Parameter im URL), die Datenmenge dort ist aber viel geringer. Beim POST werden die Daten im Body der Nachricht übertragen.

PUT
Wird verwendet, um eine Datei unter der angegebenen URL zu speichern.

4.1.3.2 HTTP-Response

Eine Antwort eines Servers kann folgendermassen aussehen:

```
HTTP/1.1 200 OK
Date: Sat, 17-March-01 11:45:13 GMT
Server: Apache/1.3
Content-type: text/html
Content-length: 91
<html>
    <title>Hello</title>
    <body>
        Welcome to my world.
    </body>
 </html>
```

Die erste Zeile stellt die ***Status-Line*** dar. Diese stellt sich aus folgenden Informationen zusammen:

- HTTP-Version: z.B. „HTTP/1.1"
- Status-Code: gibt an, inwiefern die Anfrage bearbeitet werden konnte
    - Codebereich 1xx: Allgemeine Informationen
    - Codebereich 2xx: Anfrage des Clients verstanden und erfüllt
    - Codebereich 3xx: Anfrage des Clients verstanden, jedoch nicht erfüllbar
    - Codebereich 4xx: Anfrage des Clients unvollständig oder fehlerhaft
    - Codebereich 5xx: Fehler im Server

Danach folgen die ***Response-Header-Felder.*** Sie enthalten die nötigen Informationen über den Server oder die angeforderten Daten. Pro Zeile steht jeweils ein Header, nach dem letzten folgt eine Leerzeile. Danach folgt der Body der Antwort, in welchem die Daten, welcher der Client gewünscht hat, übertragen werden.

- Content-Type: gibt den Medientyp der enthaltenen Daten an
- Server: enthält Informationen zum Server (Typ, Serversoftware)

4.2 WebSocket[1]

Das WebSocket-Protokoll bietet eine bidirektionale Verbindung zwischen einer Webapplikationen und einem Webserver.

4.2.1 Allgemein

Dieses Protokoll, welches noch relativ jung ist, basiert auf TCP. Es funktioniert jedoch nicht mit allen Webservern, denn dieser muss WebSockets unterstützen. Ist eine solche Verbindung erstmals aufgebaut können Server und Client jederzeit mit dem Senden von Daten beginnen. Beide Parteien sind persistent miteinander verbunden.

Application	**WebSocket**
Transport	**TCP**
Internet	**IP (IPv4, IPv6)**
Network Access	**Ethernet etc.**

Tabelle 2: Protokollstapel WebSocket

4.2.2 Funktionsweise

Um eine Verbindung zwischen Client und Server herzustellen ist ein Handshake nötig. Die Verbindung wird mittels eines HTTP-Request aufgebaut. In der Anfrage ist der Request-Header „Upgrade" enthalten, welcher den Vorschlag macht auf das WebSocket-Protokoll zu wechseln. Sofern der Server das unterstütz wird darauf gewechselt.

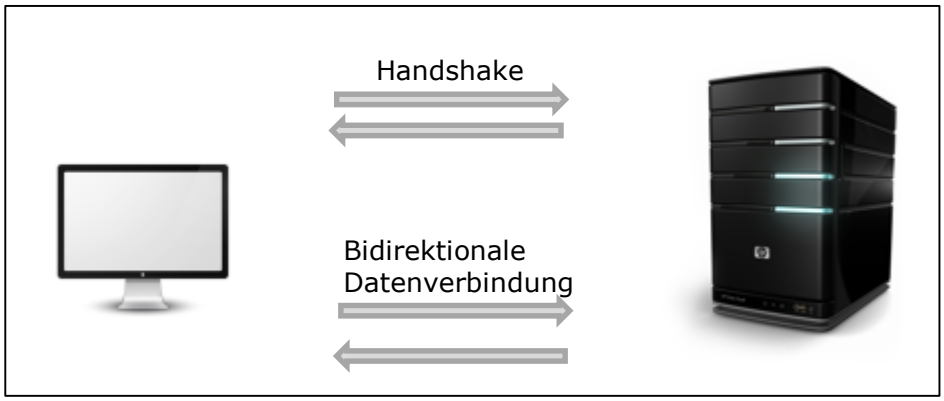

Abbildung 6: Funktionsweise WebSocket

4.2.3 Aufbau

Die Verbindung muss auf- und abgebaut werden. Dazwischen können gleichzeitig Daten ausgetauscht werden.

4.2.3.1.1 Verbindungsaufbau (Opening Handshake)

So sieht eine typische Anfrage eines Clients aus, der auf das WebSocket-Protokoll wechseln möchte.

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Origin: http://example.com
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Version: 13
Sec-WebSocket-Protocol: chat, superchat
```

Die erste Zeile ist dabei aufgebaut wie die ***Request-Line*** bei einem normalen HTTP-Request. Methode, Ressource, Protokoll und dessen Version werden darin angegeben. Da-

[1] Quelle: (Ullrich, 2015),(de.wikipedia.org, 2015)

nach folgt der Host, optional kann zusätzlich ein anderer Port angegeben werden, falls von Standardport 80 abgewichen werden soll.
Zusätzlich folgen einige Felder mit welchen die Verbindung spezifiziert wird.

- Upgrade: fordert den Server auf, das Protokoll zu wechseln
- Connection: die Verbindung muss ebenfalls auf das neue Protokoll angepasst werden
- Origin: schickt der Webbrowser mit, um vor Cross-Domain-Anfragen in Browsern zu schützen
- Sec-WebSocket-Key: enthält einen 16-Byte-Wert der base64-kodiert wurde
- Sec-WebSocket-Version: enthält nach aktueller Spezifikation Wert 13
- Sec-WebSocket-Protocol: Anwendungsschicht-Unterprotokolle die der Client unterstützt

Die Antwort des Servers sieht folgendermassen aus:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

Die erste Zeile ist wie die ***Status-Line*** beim HTTP aufgebaut. Mit dem Status 101 wird bestätigt, dass das Protokoll gewechselt wird. Die Werte der Felder „Upgrade" und „Connection" müssen den gleichen Inhalt besitzen wie die der Anfrage um die Verbindung zu vervollständigen. Weiter Felder sind:

- Sec-WebSocket-Accept: speziell abgeänderter Wert des „Sec-WebSocket-Key" Client soll wissen, dass Server Nachricht gelesen und verstanden hat, falscher Key wird als Ablehnung interpretiert und Verbindung kommt nicht zu Stande
- Sec-WebSocket-Protocol: dieser optionale Header gibt an, welches vom Client Unterstützte Unterprotokoll verwendet wird

4.2.3.2 Verbindungsabbau (Closing Handshake)

Möchte eine Seite die Verbindung schliessen, so sendet es ein Frame mit dem Befehlscode „0x8". Optional kann ein Grund für das Abbauen der Verbindung im Body der Nachricht angegeben werden, dann beschreiben die ersten zwei Bytes des Bodys einen Statuscode für den Grund.
Wird ein solches Close-Frame empfangen, ohne vorher selber eines verschickt zu haben, so sendet man als Antwort ebenfalls so ein Frame. Nach dem dieses gesendet ist, schliesst man die TCP-Verbindung. Mit diesem System stellt es auch kein Problem dar, wenn beide Seiten gleichzeitig die Verbindung schliessen wollen. In diesem Fall senden also auch beide ein Close-Frame als Antwort und beide schliessen die TCP-Verbindung.

4.3 MQTT[1]

MQTT steht für Message Queue Telemetry Transport. Dabei handelt es sich um ein offenes Nachrichten-Protokoll, welches speziell für Machine-to-Machine-Kommunikation (M2M) geeignet ist.

4.3.1 Allgemein

Das MQTT Protokoll wurde bereits 1999 entwickelt. MQTT ist ein kleines Protokoll und somit sehr gut für „Internet of Things" Anwendungen geeignet. Oft ist man dort durch eine limitierte Bandbreite oder limitierte Daten beschränkt. Das MQTT Protokoll basiert auf TCP/IP.

Application	**MQTT**
Transport	**TCP**
Internet	**IP (IPv4, IPv6)**
Network Access	**Ethernet etc.**

4.3.2 Funktionsweise

Speziell beim MQTT Protokoll ist, dass zwischen Sensor und Datenbank, bzw. Client, noch ein sogenannter Broker verwendet wird. Dieser empfängt die Daten des Sensors und leitet sie an den Client oder an die Datenbank weiter. Vorteil dabei ist, dass weder Sensor noch Datenbank zur selben Zeit aktiv sein müssen. Das Senden über den Broker funktioniert nach dem Publish/Subscribe-Prinzip. Die Nachrichten, welche über den Broker gesendet werden, sind in Topics unterteilt. So teilt ein Subscriber bei seiner Anmeldung beim Broker mit welche Topics er abonnieren möchte. Dieser leitet die Nachrichten dann korrekt an alle Abonnenten/Subscriber weiter. In der Abbildung 7 sieht man die Funktionsweise des MQTT Protokolls mit einem Broker.

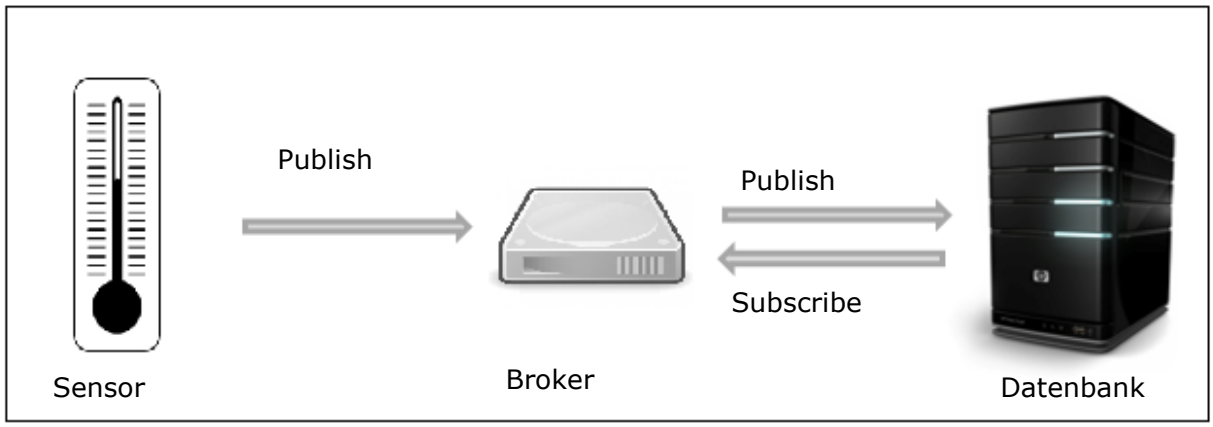

Abbildung 7: Funktionsweise MQTT

4.3.3 Aufbau

Durch das Abonnieren von verschiedenen Topics lässt sich so entscheiden welche Daten man empfangen möchte. In der Abbildung 8 sieht man ein Beispiel mit drei Clients und drei Sensoren. Je nachdem welche Topics man abonniert, erhält man die dazugehörigen Daten vom Broker.

[1] Quelle: (Sic-software.com, 2015)

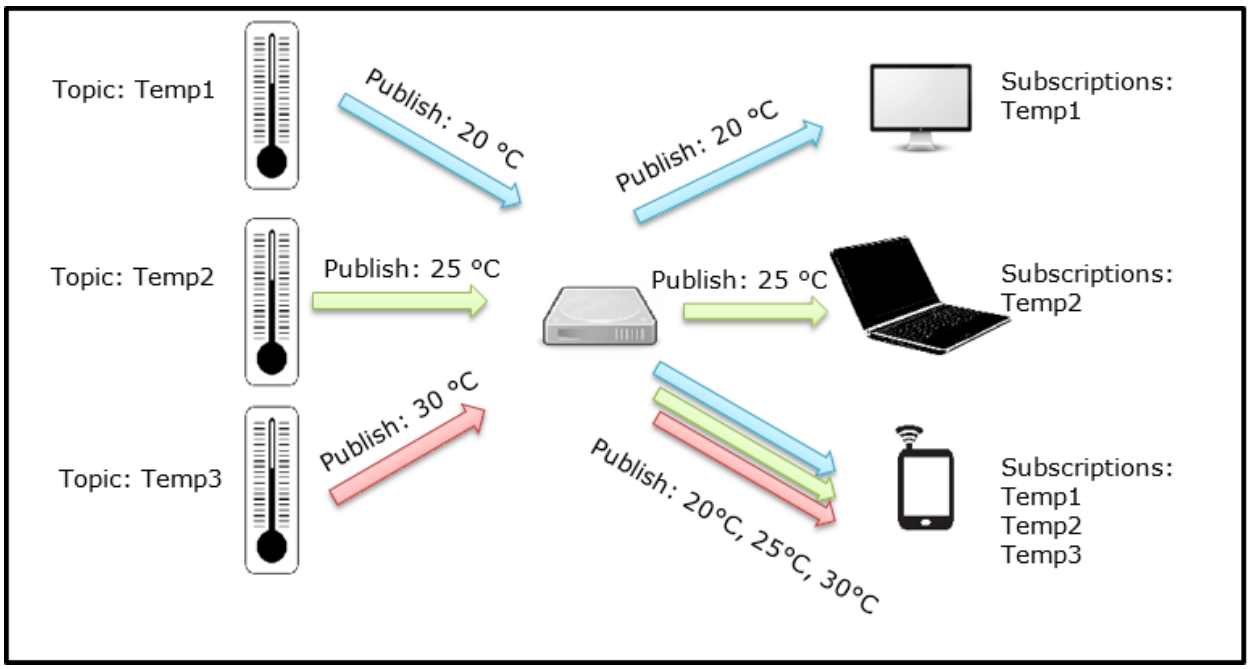

Abbildung 8: MQTT Beispiel

4.3.4 Verbindungsoptionen

Mit dem MQTT Protokoll lassen sich zudem noch diverse Einstellungen vornehmen.

4.3.4.1 Quality of Service (QoS)

Im mobilen Datenaustausch kommt es immer wieder zu Verbindungsunterbrüchen. Mittels Quality of Service lässt sich so die zuverlässige Datenübertragung garantieren. Dabei wird in drei Stufen unterschieden.

QoS = 0 (fire and forget).
Die Daten werden gesendet ohne Überprüfung ob sie erfolgreich angekommen sind. Daten können somit verloren gehen.

QoS = 1
Nachrichten kommen mindestens einmal an. Der Empfänger schickt eine Bestätigung bei erhalt der Nachricht. Geht diese Bestätigung des Empfängers jedoch verloren, wird die Nachricht erneut gesendet. Somit können die Nachrichten mehrmals beim Empfänger ankommen.

QoS = 2
Nachrichten kommen genau einmal an. Wie beim QoS1 wird hier der Erhalt der Nachricht vom Empfänger bestätigt. Jedoch wird diese Bestätigung noch einmal vom Sender Bestätigt. Nur so kann gewährleistet werden, dass die Nachricht genau einmal beim Empfänger ankommt.

Je nach Einsatzgebiet kann hier individuell entschieden werden für welche Sicherheit man sich entscheidet. Wird regelmässig die GPS-Position mit Temperatur, Ausrichtung, etc. gesendet ist es verkraftbar wenn ein Wert verloren geht. Kommt es an einem Aufpralldämpfer jedoch zu einem Unfall, wäre es fatal wenn diese Meldung nicht ankommt. Es muss daher garantiert werden dass diese Meldung mindestens einmal erfolgreich übermittelt wird.

4.3.4.2 Last Will and Testament

MQTT Clients können dem Broker mitteilen was geschehen soll wenn ein Client die Verbindung zum Broker verliert. Somit lässt sich zum Beispiel eine letzte Nachricht, welche als „Letzter Wille" auf dem Broker gespeichert wird, beim Verbindungsunterbruch an alle Abonnenten versenden.

4.3.4.3 Retained Messages

Der Broker speichert die letzten Nachrichten welche über ihn versendet werden. Somit kann auch ein Abonnent, welcher sich später erst anmeldet, noch die letzte Nachricht vom Broker abrufen. Sendet das Device beispielsweise alle 15 Minuten seinen Standort, muss ein neuer Client nicht warten bis das Device den Standort wieder sendet. Er erhält direkt den letzten Standort welchen der Broker als „retained Message" zurückhält.

4.4 Vergleich Protokolle

Es wurden verschiedene Kriterien festgelegt, um die drei behandelten Protokolle zu beurteilen. Anhand dieser soll entschieden werden, welche der drei Varianten für diese Anwendung am besten geeignet wäre. Natürlich kann es innerhalb der Projektarbeit für verschiedene Teilbereiche auch Sinn machen, unterschiedliche Protokolle zu verwenden.

4.4.1 Datenmenge

Die Menge an Daten die übertragen wird ist für diese Anwendung sehr entscheidend. Schliesslich muss später der gesamte Datenstrom den die Geräte ans Internet schicken auch bezahlt werden.
Es wurden alle drei Protokolle lauffähig implementiert, um die Datenmengen direkt zu bestimmen. Dabei wurde vier Stunden lang, jede 15 Minuten die Zahl „25" (z.B. Umgebungstemperatur) übertragen.

HTTP
Senden:

	Datenmenge [Byte]	Anzahl Übertragungen	Gesamt [Byte]
Übertragung Nachricht	302	17	5134
TCP Verbindungsaufbau	282	17	4794
			9928

Tabelle 3: gesendete Datenmengen HTTP

Empfangen:

	Datenmenge [Byte]	Anzahl Übertragungen	Gesamt [Byte]
Antwort auf Nachricht	94	17	1598
TCP Verbindungsaufbau	343	17	5831
			7429

Tabelle 4: empfangene Datenmengen HTTP

WebSocket
Senden:

	Datenmenge [Byte]	Anzahl Übertragungen	Gesamt [Byte]
Handshake Anfrage	574	1	574
TCP Verbindungsaufbau	228	1	228
Übertragung Nachricht	64	17	1088
Keep-Alive Anfrage	55	304	16720
			18610

Tabelle 5: gesendete Datenmengen WebSocket

Empfangen:

	Datenmenge [Byte]	Anzahl Übertragungen	Gesamt [Byte]
Handshake Antwort	319	1	319
TCP Verbindungsaufbau	186	1	186
Acknowledge Nachricht	60	17	1020
Keep-Alive Acknowledge	66	304	20064
			21589

Tabelle 6: empfangene Datenmengen WebSocket

MQTT, QoS0

Senden:

	Datenmenge [Byte]	Anzahl Übertragungen	Gesamt [Byte]
TCP Verbindungsaufbau	202	1	202
Übertragung Nachricht	75	17	1275
Keep-Alive Acknowledge	110	240	26400
			27877

Tabelle 7: gesendete Datenmengen MQTT, QoS0

Empfangen:

	Datenmenge [Byte]	Anzahl Übertragungen	Gesamt [Byte]
TCP Verbindungsaufbau	126	1	126
Keep-Alive Acknowledge	120	240	28800
			28926

Tabelle 8: empfangene Datenmengen MQTT, QoS0

MQTT, QoS1

Senden:

	Datenmenge [Byte]	Anzahl Übertragungen	Gesamt [Byte]
TCP Verbindungsaufbau	202	1	202
Übertragung Nachricht	77	17	1309
Acknowledge Nachricht	54	17	918
Keep-Alive Acknowledge	110	240	26400
			28829

Tabelle 9: gesendete Datenmengen MQTT, QoS1

Empfangen:

	Datenmenge [Byte]	Anzahl Übertragungen	Gesamt [Byte]
TCP Verbindungsaufbau	126	1	126
Acknowledge Nachricht	60	17	1020
Keep-Alive Acknowledge	60	240	14400
			15546

Tabelle 10: empfangene Datenmengen MQTT, QoS1

MQTT, QoS2

Senden:

	Datenmenge [Byte]	Anzahl Übertragungen	Gesamt [Byte]
TCP-Verbindungsaufbau	202	1	202
Übertragung Nachricht	77	17	1309
Acknowledge Nachricht	112	17	1904
Keep-Alive Acknowledge	110	240	26400
			29815

Tabelle 11: gesendete Datenmengen MQTT, QoS2

Empfangen:

	Datenmenge [Byte]	Anzahl Übertragungen	Gesamt [Byte]
TCP-Verbindungsaufbau	126	1	126
Acknowledge Nachricht	120	17	2040
Keep-Alive Acknowledge	60	240	14400
			16566

Tabelle 12: empfangene Datenmengen MQTT, QoS2

4.4.2 Verhalten bei Netzverlust

Ebenfalls eine wichtige Eigenschaft, ist das Verhalten bei einem kurzzeitigen Netzverlust. Dies ist vor allem beim WebSocket-Protokoll wichtig, was geschieht wenn die bidirektionale Datenleitung unterbrochen wird?

HTTP

Bei einer HTTP-Kommunikation hat ein kurzzeitiger Netzverlust keine grösseren Probleme zur Folge. Bei jeder Anfrage die gestellt wird, bekommt der Client eine Antwort zurück. Falls jedoch keine Verbindung zum Server möglich ist, kommt auch ganz sicher keine HTTP-Response zurück. Man bemerkt also, dass etwas nicht in Ordnung ist, und kann die Anfrage nochmals schicken.
Da gewisse Daten wahrscheinlich in einem regelmässigen Zeitabstand an den Server gesendet werden, würde dieser einen Netzverlust ebenfalls bemerken. Oder aber auch, wenn er dem Device eine Anfrage sendet und keine Antwort bekommt.

WebSocket

Eine WebSocket-Verbindung basiert auf TCP, und diese benützt ein „FIN"-Packet, um eine Verbindung zu schliessen. Im Falle eines Internetverlusts kann also auch dieser Befehl nicht übertragen werden. Der WebSocket-Endpoint wird somit auch nicht benachrichtigt, dass die TCP-Verbindung tot ist. Um solche Probleme zu beheben gibt es das „TCP-keep-alive". Es generiert zwar zusätzlichen Netzwerkverkehr, jedoch können so Unterbrechungen detektiert werden. Dabei sendet ein TCP-Socket regelmässige Keep-Alive-Anfragen und bekommt eine Bestätigung, falls die Leitung in Ordnung ist.

MQTT

Ähnlich wie bei den WebSockets werden ebenfalls TCP-Keep-Alive Anfragen gesendet. Der Client sendet dabei in regelmässigen Abständen einen Ping Request zum Broker. Dieser Antwortet mit einer Ping Response. Somit lässt sich ein Netzverlust feststellen und dem entsprechen darauf reagieren. Wie in Kapitel 4.3.4.2 beschrieben kann mittels definiertem Letzen Willen noch eine Aktion vom Broker ausgeführt werden falls die Verbindung zu einem Client verloren geht. Der MQTT Client ist ebenfalls in der Lage sich wieder mit dem Broker zu verbinden wenn die Verbindung wieder steht. Dabei holt er sich beim Broker alle Nachrichten welche inzwischen für ihn Angekommen sind ab. Siehe auch Kapitel 4.3.4.3 Retained Messages.

4.4.3 Auswertung Protokollvergleich

	HTTP	WebSocket	MQTT (QoS0)	MQTT (QoS1)	MQTT (QoS2)
Datenmenge (gesendet)	9.9 kB	18.6 kB	27.8 kB	28.8 kB	29.8 kB

Bei der Auswertung der Datenmengen muss man einige Faktoren berücksichtigen, um keine voreiligen Schlüsse zu ziehen. Beim Test wurde im Abstand von 15min jeweils eine kurze Zeichenkette übertragen. Dazwischen war Funkstille. Deshalb schloss das HTTP-Protokoll erstaunlich gut ab. Es hat zwar extrem viel Overhead, da jedoch nur in sehr grossen Abständen Daten verschickt werden, halten sich die gesendeten Mengen in Grenzen.

Beim WebSocket sieht es etwas anders aus. Zwar brauch das Senden der Nachricht sehr wenig Datenvolumen, dafür das Aufrechterhaltung der Datenleitung umso mehr. Der Test ist also nicht unbedingt zum Vorteil von WebSockets. Dieses Protokoll ist für eine kontinuierliche Kommunikation mit vielem hin und her besser geeignet.

Auch beim MQTT Protokoll wird das Datenvolumen in diesem Test hauptsächlich durch die TCP-Keep-Alive anfrage bestimmt. Bei den unterschiedlichen Quality of Services ist der Unterschied minimal.

Das Intervall der TCP-Keep-Alives könnte man jedoch bei beiden Protokollen anpassen. Zum Beispiel indem man den Zeitabstand zwischen den Anfragen vergrössert. Standardmässig wird bei den WebSockets alle 45 Sekunden, bei MQTT alle 60 Sekunden eine Keep-Alive-Anfrage versendet. Verändert man diese Zeit geht es allerdings auch länger, bis eine unterbrochene Verbindung detektiert werden kann. Hier muss ein sinnvoller Kompromiss zwischen benötigtem Datenvolumen und der Zeit bis zur Feststellung eines Netzunterbruchs gefunden werden.

Um zu entscheiden welches Protokoll für die Anwendung am besten geeignet ist, muss mit dem Industriepartner die Spezifikationen ganz genau geklärt werden. Es muss klar sein in welchen Intervallen mit wie vielen Daten gerechnet werden kann. Anhand der Untersuchungen zeigt sich jedoch, welches Protokoll für welche Art von Kommunikation die beste Variante wäre.

4.5 Alternative Protokolle

Im Rahmen des Fachmoduls wurden lediglich die vorangehenden drei Protokolle genau untersucht. Es gibt jedoch noch alternative Kommunikationsprotokolle, welche nicht ganz ausser Acht gelassen wurden.

4.5.1 MQTT-SN[1]

Das MQTT-SN-Protokoll (Sensor Networks) ist eine Version des MQTT-Protokolls, welches auf die Besonderheiten einer kabellosen Übertragung adaptiert wurde. Bei einer solcher Kommunikation sind die Fehlerraten viel höher, als bei einer drahtgebundenen Datenübertragung. Ebenfalls sind die Übertragungsraten tiefer. MQTT-SN ist ebenfalls für die Implementierung auf low-cost Geräten optimiert, welche nur limitierte Rechen- und Speicherressourcen zur Verfügung haben.
Dieses Unterprotokoll wurde so designt, dass es möglichst nahe am Original liegt, jedoch möglichst schlank ist. So ist es besser geeignet für kabellose Sensor-Netzwerke, mit begrenzten Ressourcen.

4.5.2 CoAP[2]

CoAP steht für Constrained Application Protocol. Es kommt vor allem bei internetfähigen Geräten mit beschränkten Ressourcen zum Einsatz. Da es einen sehr schlanken Aufbau besitzt, wird es in eingebetteten Systemen oft als Alternative zum HTTP verwendet. Es wird vor allem zur Übertragung von Daten in Netzwerken mit geringen Übertragungs- und hohen Verlustraten benutzt. CoAP wird meist für Anwendungen mit UDP entwickelt, man ist aber nicht zwingend daran gebunden. Auch Lösungen mit TCP sind möglich.

[1] Quelle: (Stanford-Clark & Truong, 2015)
[2] Quelle: (Trapickin, 2015)

5 Systemarchitektur

Für die verschiedenen Übertragungsprotokolle wurde jeweils eine erste Systemarchitektur erstellt.

5.1 Architektur mit HTTP

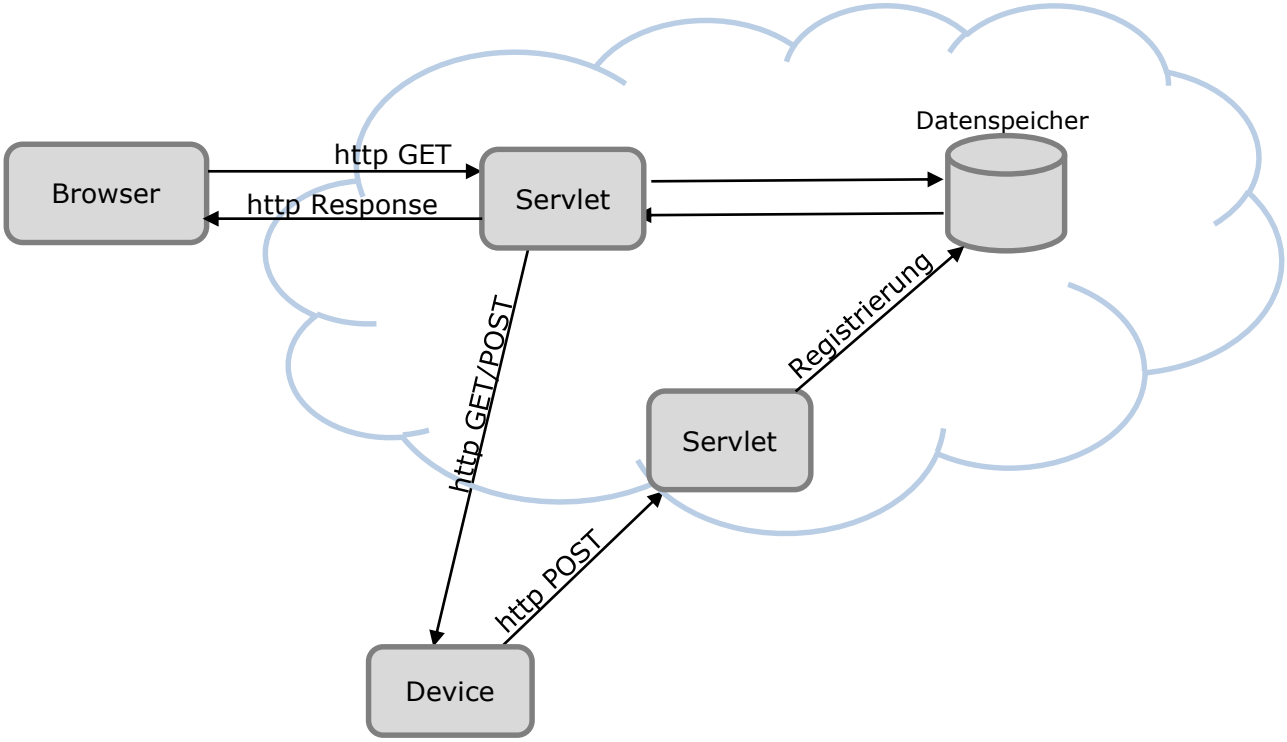

Abbildung 9: Systemarchitektur mit HTTP

In Abbildung 9 sieht man, wie eine Systemarchitektur aussähe, wenn man auf das HTTP setzten würde. Jedes Device hat einen festgelegten Name. Um sich bei der Anwendung anzumelden schickt es einen HTTP Post. Die Absender IP-Adresse wird danach zusammen mit dem Device-Namen in einem Datenspeicher abgelegt, so können alle aktiven Geräte registriert werden.
Vom Browser aus kann die Applikation mittels eines HTTP Get angezeigt werden. Diese holt die nötigen Informationen zur Visualisierung aus dem Datenspeicher, oder steuert das Device direkt über die IP-Adresse an.
Eventuell muss eine solche Adresse bei einem Provider registriert werden, ansonsten funktioniert der Zugriff auf das Device nur wenn keine Firewall dazwischen besteht.

5.2 Architektur mit WebSocket

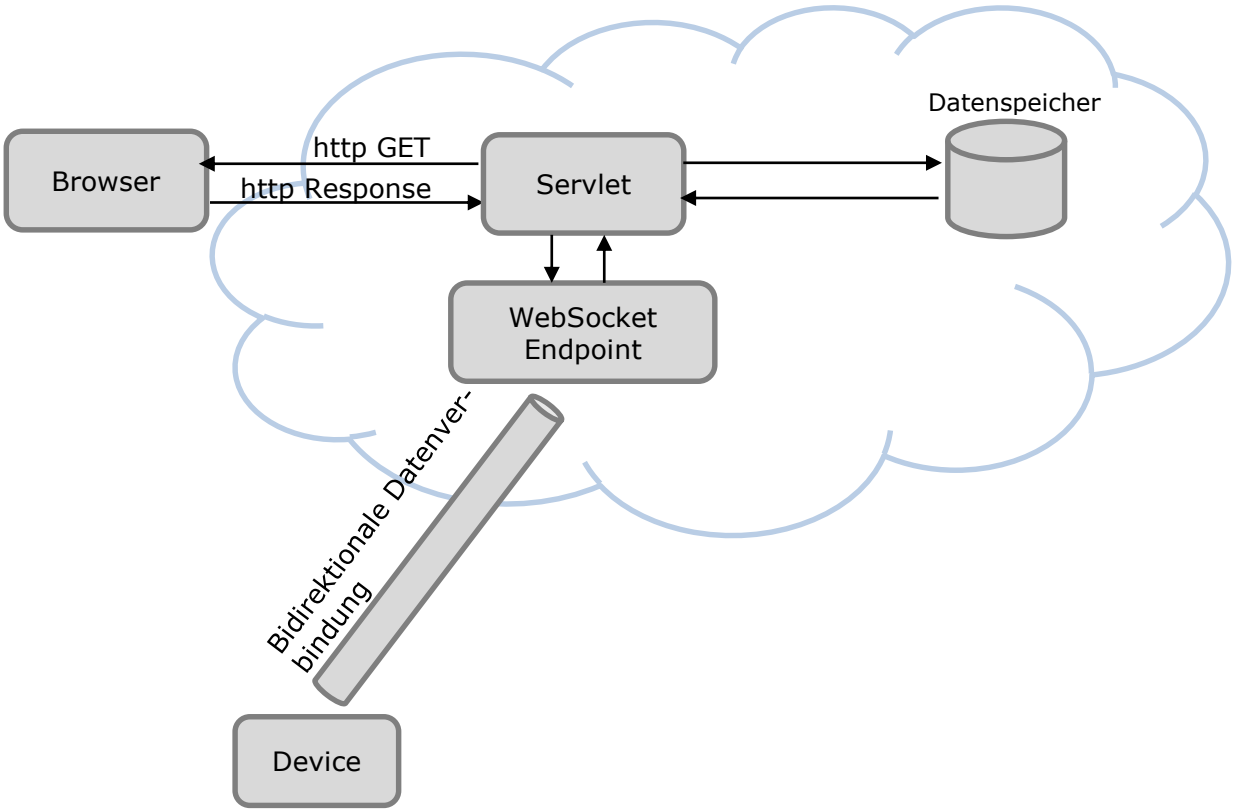

Abbildung 10: Architektur mit WebSocket

Das Device schickt eine HTTP Get-Anfrage, um auf das WebSocket-Protokoll zu wechseln. Die Anwendung speichert ebenfalls Name und IP des Devices in einem Datenspeicher, und akzeptiert den Verbindungsaufbau falls es sich um ein autorisiertes Gerät handelt. Danach steht eine bidirektionale Datenverbindung zur Verfügung, sodass beide Parteien Daten senden können.

5.3 Architektur mit MQTT

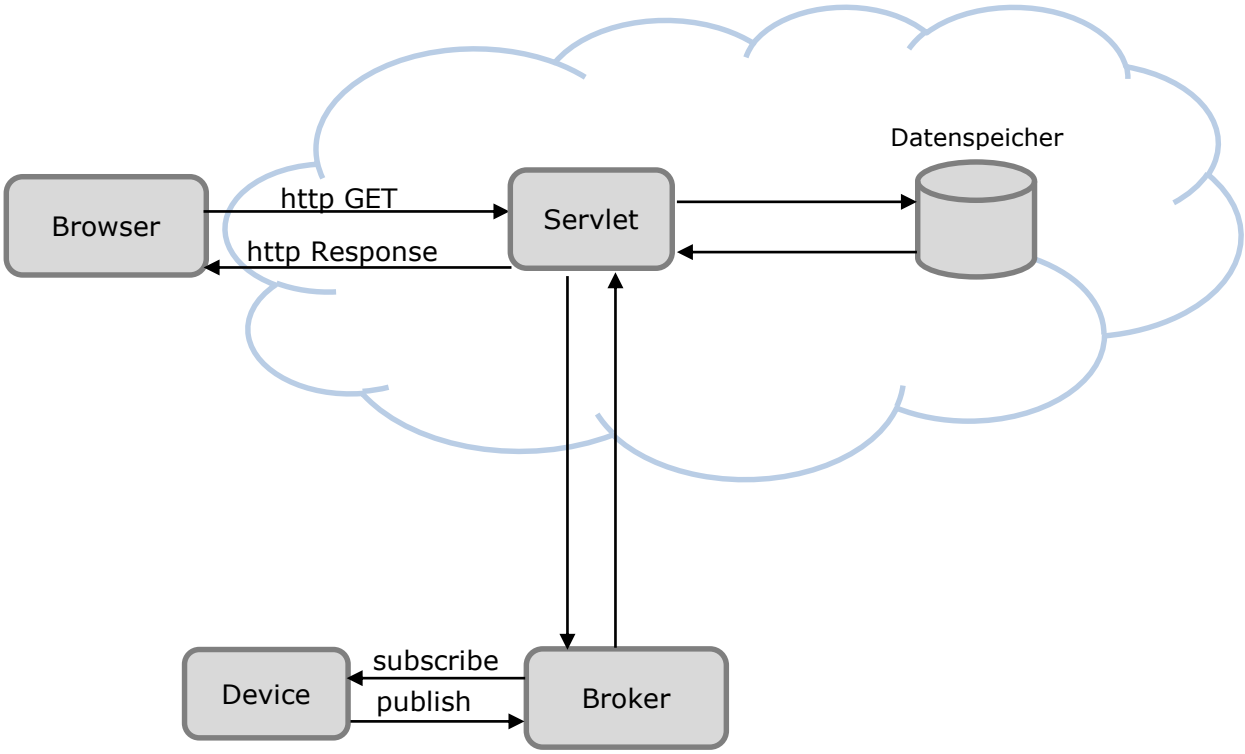

Abbildung 11: Systemarchitektur mit MQTT

Da sich das Device beim Broker mit seinem Namen, bzw. Topic, anmeldet muss hier weder Name noch IP im Datenspeicher abgelegt werden. Wie in Abbildung 11 zu sehen kommunizieren Client und Webapplikation über den Broker miteinander.

6 Hardware

Die Leiterplatte wurde im Rahmen einer Bachelorarbeit entwickelt, und steht fertig zur Verfügung.[1]

6.1 CPU

Auf der Leiterplatte ist ein STM32F207 verbaut.

Technische Details	
Core ARM 32-bit Cortex-M3 CPU 120 MHz maximum frequency	**Low power** Sleep, Stop and Standby modes VBAT supply for RTC,
Clock, and supply management 1.8 to 3.6 V application supply and I/Os 4 to 25 MHz crystal oscillator 32 kHz oscillator for RTC with calibration	**General-purpose DMA** 16-stream DMA controller Up to 17 timers Up to twelve 16-bit and two 32-bit timers
Debug mode Serial wire debug (SWD) & JTAG interfaces Cortex-M3 Embedded Trace Macrocell	**Memories** Up to 1 Mbyte of Flash memory Up to 128 + 4 Kbytes of SRAM Flexible static memory controller
I/Os and Peripheral Interface 82 I/Os, all 5 V-tolerant Up to 3 CI2C interfaces Up to 6 USARTs 2 CAN interfaces	**Peripherie Peripheral Interface** USB 2.0 HS/FS device/host/OTG controller with dedicated DMA 10/100 Ethernet MAC with dedicated DMA

Tabelle 13: Technische Details CPU[2]

6.2 GSM-Modul

Auf der Leiterplatte wurde das GSM-Modul Sim928 der Firma SimCom verwendet. Es verfügt zusätzlich über ein verbautes GPS-Modul. Das GSM-Modul unterstützt vier Hauptfrequenzen. Damit ist es mit den meisten Netzen weltweit kompatibel.

6.3 Sensorik

Auf der Hardware sind zusätzlich noch weitere Sensoren verbaut.
Um festzustellen in welche Fahrtrichtung das Fahrzeug steht, wird ein Magnetfeldsensor benötigt. Dieser dient als digitaler Kompass. Um einen Aufprall zu erkennen wird zusätzlich ein Beschleunigungssensor benötigt. Ein Temperatur- und ein Feuchtesensor liefern dabei noch weitere Informationen. Sämtliche Sensoren kommunizieren über eine I^2C-Schnittstelle.

6.3.1 Temperatur- und Feuchtigkeitssensor

Verbaut wurde ein Si7020-A20 von Silicon Labs. Dieser hat einen Temperaturbereich von -20 °C bis +70 °C. Bei -20 °C beträgt der Fehler 0.55 °C, bei +70 °C 0.4 °C.

6.3.2 Magnetfeldsensor

Der verwendete Magnetfeldsensor ist ein LSM303 von ST mit einer Auflösung von 16 Bit. Zusätzlich zum Magnetfeldsensor ist noch ein 3D-Beschleunigungssensor verbaut. Dies ermöglich die Schräglage des Sensors softwaretechnisch zu korrigieren.

6.3.3 Beschleunigungssensor

Zur Unfalldetektion wird ein separater 3D-Beschleunigungssensor benötigt. Wird ein bestimmter Grenzwert überschritten, wird mittels einem Interrupt die CPU sofort über einen Aufprall informiert. Dieser Grenzwert wurde auf 8 g eingestellt. Für diese Anwendung wurde ein LIS331DLH, ebenfalls von ST Microelectronics, verwendet.

[1] Quelle: (Figliuolo & Gautsch, 2015)
[2] Quelle: (Figliuolo & Gautsch, 2015)

7 Software

Zu der zur Verfügung stehenden Leiterplatte wurde in derselben Bachelorarbeit ebenfalls eine Basis-Software entwickelt. Auf dieser kann aufgebaut werden. Die gesamte Software wurde mit C++ programmiert. In diesem Kapitel wird nur auf die schon vorhandene Implementierung eingegangen.

7.1 Entwicklungsumgebung Software

Die jetzige Software wurde mit der Entwicklungsumgebung von Keil programmiert. Auf der CPU ist ein Real-Time Operating System, ebenfalls von Keil installiert. Im MDK-ARM (Microcontroller Development Kit) sind alle benötigen Funktionalitäten enthalten, welche für die Programmierung von Cortex –M Chip gebraucht werden.
Für die Software, die im Rahmen der Bachelorarbeit entwickelt wird, sollen diese Gegebenheiten übernommen werden.

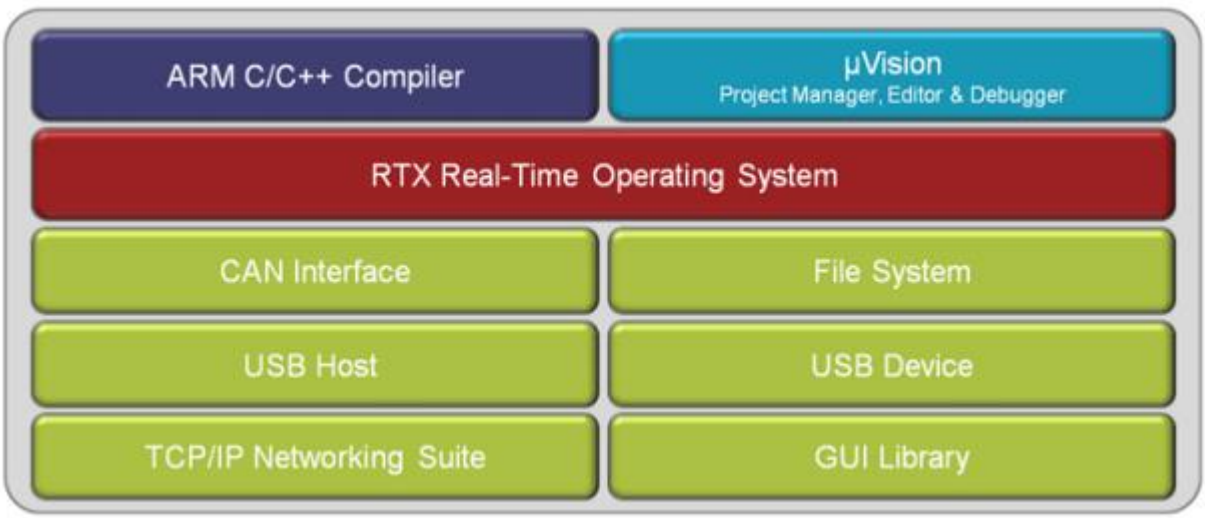

Abbildung 12: Microcontroller Development Kit[1]

7.2 Grobaufbau

Die Software wurde mit verschiedenen Threads aufgebaut. Deren Aufgabe wird hier kurz erläutert.

7.2.1 systemThread

Das Grundgerüst der Software wird mit dem *systemThread* implementiert. Er startet die anderen Threads.

7.2.2 sensorThread

Dieser Thread ist dafür zuständig, dass die Sensorwerte richtig ausgelesen werden. Dazu gehört das Konfigurieren der verschiedenen Schnittstellen für die einzelnen Sensoren, sowie das Auslesen der entsprechenden Werte. Dabei wird alles in einem Benutzerdaten-String zusammengefasst.

7.2.3 gsmThread

Der *gsmThread* ist für die Kommunikation zuständig. Das GSM-Modul wird mit dem Internet verbunden, und ein TCP-Socket wird erstellt.
Die Benutzerdaten werden regelmässig per HTTP an die Cloud „Nimbits.com" geschickt.

7.2.4 sensortrigger

Beim *sensortrigger* handelt es sich um kein Thread, lediglich um eine normale Klasse. Sie triggert nach einer definierten Zeit die Sensoren, damit diese wieder eingelesen werden.

[1] Quelle: (Figliuolo & Gautsch, 2015)

7.3 Senden der Daten

Der wichtigste Teil für die Erarbeitung der Bachelorarbeit ist sicherlich das Versenden der Daten mit dem GSM Modul. Bisher werden die Daten per HTTP an eine Cloud mit dem Name „Nimbits" gesendet. Der HTTP-Header wird dabei selber zusammengestellt.

```
//Definieren des HTTP Headers inklusive der zu versendenden Daten in der URL mit json
const CtString   GsmThread::s_REQUEST_VERSION  = "HTTP/1.1\r\n";
const CtString   GsmThread::s_HOST             = "Host: cloud.nimbits.com\r\n";
const CtString   GsmThread::s_CONTENT_TYPE     = "Content-Type: application/json\r\n";
const CtString   GsmThread::s_CONNECTION       = "Connection: close\r\n";
const CtString   GsmThread::s_CONTENT_LENGHT   = "Content-Length: 0\r\n";
const CtString   GsmThread::s_END              = "\r\n";
const CtString   GsmThread::s_POST             = "POST ";
const CtString   GsmThread::s_PAGE             = "/service/v2/value?";
```

Listing 1: Konstanten für HTTP-Header

Im *gsmThread* gibt es die Methode *sendData*. Wird diese aufgerufen, so werden vom sensorThread die gesamten Benutzerdaten geholt. Danach wird der HTTP-Post-Header aus verschiedenen String-Teilen (Listing 1) zusammengestellt. Es wird jeweils nur ein Sensorwert pro Sendevorgang verschickt (d.h. Temperatur, Luftfeuchtigkeit etc.). Diese werden in einer fixen Reihenfolge nacheinander gesendet.

8 Abbildungsverzeichnis

9 Tabellenverzeichnis

10 Literaturverzeichnis

[1] Cisco (Hrsg.). (06. 12 2015). Abgerufen am 6. 12 2015 von http://cdn.iotwf.com/resources/71/IoT_Reference_Model_White_Paper_June_4_2014.pdf

[2] *de.wikipedia.org*. (2015). Abgerufen am 21. 12 2015 von https://de.wikipedia.org/wiki/WebSocket

[3] *de.wikipedia.org*. (2015). Abgerufen am 23. 12 2015 von https://de.wikipedia.org/wiki/WebSocket

[4] Figliuolo, M., & Gautsch, N. (2015). *Bachelorarbeit - Mobile Datenerfassung.* Winterthur: zhaw.

[5] Hsieh, W., Abbott, M., & Jazayeri, R. (2015). *www.kpcb.com*. Abgerufen am 28. 12 2015 von Jazayeri: http://www.kpcb.com/blog/how-kleiner-perkins-invests-in-the-internet-of-things-picking-the-winners

[6] Jens Franke, J. J. (25. 11 2015). *Goessner.net*. Von http://goessner.net/download/learn/mwt/ws2005/presentations/HTTP.pdf abgerufen

[7] *MQTT*. (17. 12 2015). Von http://mqtt.org/ abgerufen

[8] Sander, M. (26. 3 2015). *NZZ*. Abgerufen am 6. 12 2015 von nzz.ch: http://www.nzz.ch/wirtschaft/was-hinter-dem-hype-um-das-internet-der-dinge-steckt-1.18510918

[9] *Sic-software.com*. (2015). Abgerufen am 21. 12 2015 von http://www.sic-software.com/das-mqtt-protokoll-1

[10] Stanford-Clark, A., & Truong, H. L. (2015). *mqtt.org*. Abgerufen am 26. 12 2015 von http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf

[11] Trapickin, r. (23. 12 2015). *http://www.net.in.tum.de/*. Von http://www.net.in.tum.de/fileadmin/TUM/NET/NET-2013-08-1/NET-2013-08-1_16.pdf abgerufen

[12] Ullrich, B. (10. 12 2015). *www.net.in.tum.de*. Abgerufen am 8. 12 2015 von http://www.net.in.tum.de/fileadmin/TUM/NET/NET-2012-04-1/NET-2012-04-1_08.pdf

[13] *www.triopan.ch*. (2015). Abgerufen am 14. 12 2015 von http://www.triopan.ch/index.php/de/

[14] cloud.google.com. (2015). Abgerufen am 21. 12 2015 von https://cloud.google.com/appengine/docs/ abgerufen

[15] cloud.google.com. (2015). Abgerufen am 21. 12 2015 von https://cloud.google.com/sql/docs/introduction abgerufen

[16] cloud.google.com. (2015). Abgerufen am 21. 12 2015 von https://cloud.google.com/datastore/docs/concepts/overview abgerufen