Hochschule Offenburg
offenburg.university

# Anwendungsschicht-Protokolle für das Internet der Dinge

**Master Thesis**

im Studiengang
Medien und Kommunikation

vorgelegt von

**Johannes Kässinger**

Wintersemester 2016/17
an der Hochschule Offenburg - Fakultät M+I

| | |
|---|---|
| Erstprüfer: | Prof. Dr. Tom Rüdebusch |
| Zweitprüfer: | Prof. Dr. Sänger |

# Kurzfassung

Gegenstand der hier vorgestellten Arbeit ist eine Untersuchung der gängigsten Anwendungsschicht-Protokolle für das Internet der Dinge. Sie umfasst zum einen die Recherche und den theoretischen Vergleich der Protokolle anhand vorher festgelegter Untersuchungskriterien und zum anderen einen praktisch implementierten Versuchsaufbau, der das Zusammenwirken verschiedener Protokolle veranschaulichen soll.

Zunächst wird eine kurze Einführung in das Thema „Internet der Dinge“ gegeben. Dabei wird auf die Entwicklung der letzten Jahre sowie den Stand der Technik eingegangen. Zum einen wird aufgezeigt, inwieweit sich das Thema auf die verschiedenen Bereiche des täglichen Lebens ausgebreitet hat und zum anderen welche verschiedenen Anwendungspakete diverser Software-Hersteller auf dem Markt sind. Bei der Untersuchung dieser Pakete werden die Protokolle ermittelt, die verwendet werden. Diese bilden die Menge an zu untersuchenden Protokollen für den späteren Vergleich. Um diese in einem wissenschaftlichen Rahmen miteinander vergleichen zu können, werden Kriterien als Basis der Untersuchung definiert. Anhand derer werden alle Protokolle untersucht und anschließend in einer Tabelle zusammengefasst.

Im zweiten Teil werden dann einige der untersuchten Protokolle mithilfe von diversen MCUs (Micro Controller Unit) verwendet und deren Funktionsweise/Handhabung miteinander verglichen. Dabei entsteht ein Aufbau, der zukünftigen Studierenden die Möglichkeit bietet, schnell einen Einblick in den Bereich „Internet der Dinge“ zu erhalten.

Es folgen eine Zusammenfassung und ein Ausblick.

**Schlagwörter:** Master Thesis, Internet der Dinge, IoT, Anwendungsschicht

# Versicherung/Erklärung

| | | | |
|---|---|---|---|
| Name: | Kässinger | Vorname: | Johannes |
| Matrikel-Nr.: | 171801 | Studiengang: | MuK |

Hiermit versichere ich, Johannes Kässinger, dass ich die vorliegende Master Thesis mit dem Titel „Anwendungsschicht-Protokolle für das Internet der Dinge“ selbständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.

Ort, Datum

Unterschrift

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Abkürzungsverzeichnis

| | |
|---|---|
| HSO | Hochschule Offenburg |
| M+I | Fakultät Medien und Informationswesen |
| IoT | Internet of Things (Internet der Dinge) |
| IIoT | Industrial Internet of Things |
| MCU | Micro Controller Unit |
| PaaS | Platform as a Service |
| QoS | Quality of Service |
| MQTT | Message Queue Telemetry Transport |
| REST | Representational State Transfer |
| CoAP | Constrained Application Protocol |
| XMPP | Extensible Messaging and Presence Protocol |
| DDS | Data Distribution System |
| GND | Erdung (Ground) |
| VCC | Positive Spannung |

# Vorwort

Im Jahre 1999 wurde der Begriff „Internet of Things“ von Kevin Ashton erstmals verwendet. Er sagte damals eine Zukunft mit nahtlos verbundenen Geräten voraus, die der Menschheit Zeit und Geld sparen wird [1]. Im Jahre 2009 ergänzte er seine Aussage dahingehend, dass nahezu der gesamte Inhalt, den das Internet bereithält, bisher von Menschen erstellt wurde und hob hervor, um welches Vielfache die Effektivität und Automatisierung durch die verbundenen Geräte wäre, würden diese selbst Informationen sammeln [2].

> *We need to empower computers with their own means of gathering information, so they can see, hear and smell the world for themselves, in all its random glory. RFID and sensor technology enable computers to observe, identify and understand the world – without the limitations of human-entered data. [2]*

Aufgrund vom immer kostengünstigeren Microcontrollern und WiFi-Adaptern, die teils direkt mit einem kleinen Microcontroller versehen sind, können diese Daten einfach und in großer Zahl gesammelt werden. Zur Kommunikation dieser Elemente stehen verschiedene Protokolle zur Verfügung, die sich in ihrem Funktionsumfang und ihrer Kompatibilität von anderen Elementen/Systemen unterscheiden. Diese Unterschiede sowie deren Vor- und Nachteile sollen in dieser Arbeit untersucht und zusammengefasst werden.

Um den praktischen Teil umzusetzen, ist eine ausgiebige Einarbeitung in die Hardware erforderlich. Die zu verwendenden Bauteile müssen zunächst beschafft und ihre Handhabung erlernt werden. Die bereits getätigte Vorrecherche lässt in diesem Bereich ein spannendes Gebiet erwarten, welches jedoch durch den Umfang der Thesis eingegrenzt wird.

> *“Ich fürchte den Tag, an dem die Technologie die wichtigsten Elemente menschlicher Verhaltensweisen strukturiert. Die Welt wird nur noch aus einer Generation von Idioten bestehen.” (Albert Einstein)*

Bei allen neuen Technologien sollte aber auch darauf geachtet werden, dass sich Albert Einsteins Befürchtung der Generation von Idioten nicht bewahrheitet. Die neuen Möglichkeiten sollen vielmehr den Menschen, wie von Kevin Ashton vorhergesagt, unterstützen und nicht vollends dessen Denkvermögen ersetzen.

# 1 Überblick

In den folgenden Kapiteln werden zunächst die Begrifflichkeiten, mit der sich diese Arbeit beschäftigt, vorgestellt und kurz erläutert, welche Stufen diese in ihrer Entwicklung durchlaufen haben. Allem voran steht hierbei die Digitalisierung, gefolgt vom Internet der Dinge, der vierten Revolution in der Industrie (Industrie 4.0) sowie der Definition und Einordnung der Anwendungsschicht und ihrer Protokolle im Protokoll-Stack.

Wichtig für die Kommunikation im Internet der Dinge ist auch die Art der Übertragung. Dafür sind verschiedene Standards vorhanden, die kurz vorgestellt und in einer Übersicht zusammengefasst werden.

Es folgt ein Einblick in einige der derzeit angebotenen IoT-Lösungen verschiedener Software-Hersteller. Dabei wird auch darauf eingegangen, welche Protokolle diese unterstützen. Diese werden dann gesammelt, um sie zu einem späteren Zeitpunkt zu untersuchen.

Anschließend werden die Kriterien, anhand welcher die Protokolle miteinander verglichen werden sollen, festgelegt und definiert. Diese sind wichtig, um am Ende des Vergleichs eine aussagekräftige Übersicht erstellen zu können, welche die jeweiligen Eigenschaften sowie Vor- und Nachteile der einzelnen Protokolle beinhaltet.

Auf dieser Grundlage werden die gängigen Anwendungsschicht-Protokolle kurz vorgestellt und auf die Kriterien hin untersucht. Dabei wird auch darauf geachtet, wie diese in unterschiedlichen Umgebungen funktionieren und inwieweit diese für zukünftige Anwendungen sinnvoll sein können.

Die dabei entstandenen Erkenntnisse bilden zusammen mit der Problemstellung die Basis für die Anforderungen an ein entsprechendes Beispielprojekt, bei dem einige der Protokolle sinnvoll miteinander kombiniert werden. Zudem soll ein kleiner Versuchsaufbau zur einfachen Veranschaulichung der Funktionalitäten vorbereitet werden.

Im weiteren Verlauf folgt die Konzeptionierung für ein solches Beispielprojekt und den Versuchsaufbau. Zu berücksichtigen sind dabei der verständliche Aufbau, die einfache Handhabung sowie der größtmögliche Informationstransfer an die Studierenden.

Am Ende folgt eine Zusammenfassung über die Ergebnisse und ein Ausblick/Einschätzung auf den weiteren Verlauf in diesem Bereich.

Bereits im Vorfeld hat sich gezeigt, dass das Thema rund um das Internet der Dinge und die damit verbundenen Bereiche schon länger untersucht wird, die (Automobil-)Industrie jedoch erst vor kurzer Zeit deren Potential erkannt hat und nun einen großen Fokus auf die Entwicklung dieser Technologien legt.

# 2 Problemstellung

## 2.1 Ausgangssituation

Seit vielen Jahren bietet das Internet Informationen, Dienste und Anwendungen auf der Basis digitalisierter Daten an. Seit einiger Zeit werden auch physische Objekte mit digitaler Kommunikationsfähigkeit ausgestattet und in die Welt des Internets integriert. Die vom Menschen real erlebte Welt verschmilzt mit dem Abbild der Welt im Internet: Das sogenannte „Internet der Dinge“ (engl. Internet of Things, „IoT“) entsteht.

Physische Objekte mit digitaler Kommunikationsfähigkeit auszustatten, ermöglicht kostengünstige Microcontroller wie das Arduino System oder System-on-a-Chip-Computer mit Linux-Betriebssystem wie der Raspberry Pi.

Um komplexe Anwendungen im Internet der Dinge zu realisieren, müssen verteilte physische Objekte und Software-Instanzen miteinander kommunizieren: Automatisch öffnende Fenster benötigen die Daten von Temperatursensoren und Regenfühlern und lassen sich auch manuell über ein Smartphone bedienen; Temperaturverläufe und Lüftungsmaßnahmen werden mit Hilfe von Plattform-Diensten visualisiert; Grenzwerte lösen Alarmaktionen aus.

Die notwendige Kommunikation auf Anwendungsschicht-Ebene realisieren Protokolle, die auf die besonderen Anforderungen im Internet der Dinge ausgerichtet sind, z.B. CoAp, MQTT, REST über HTTP/1.1 oder /2.

## 2.2 Ziel der Arbeit

Im Rahmen der Master Thesis soll zunächst eine vergleichende Übersicht von Anwendungsschicht-Protokollen für das Internet der Dinge erstellt werden. Dazu sind jeweils die Funktionalitäten für die Kommunikation von Dingen und SW-Instanzen zu untersuchen, Kommunikationsarchitekturen (Client/Server, Publish/Subscribe usw.) und -formen (synchron, pipelining, asynchron usw.) zu vergleichen und die Unterstützung durch Plattform-Dienste und Programmbibliotheken (Sprachen, HW-Plattformen) sowie die Verarbeitung zu bewerten.

Schließlich ist eine Idee für ein Beispielprojekt zu entwickeln und auf Basis der Vorarbeiten mit ausgewählten Protokollen zu realisieren.

# 3 Das Internet der Dinge: Infrastruktur und Systeme

## 3.1 Begriffsdefinitionen

### 3.1.1 Digitalisierung

Für den Begriff der Digitalisierung gibt es mehrere Definitionen. Zum einen kann damit die digitale Umwandlung und Darstellung von Information, Bildern, Texten, Videos usw. gemeint sein und zum anderen die digitale Modifikation/Erweiterung von Geräten und Fahrzeugen. Ebenso kann es die (dritte) Industrielle Revolution, bei der die IT in die Automatisierung der Produktion miteinbezogen wurde, beschreiben. Die Digitalisierung hat verschiedene Neudefinitionen von Begriffen sowie komplett neue Begriffe mit sich gebracht. Güter können demnach nicht mehr nur physische Dinge sein, sondern ebenso digitale. Hervorzuheben sind dabei auch die neu entstandenen Branchen, die sich mit dem Sammeln, Analysieren und eventuellen Verarbeiten von sogenanntem User-generated Content beschäftigen. Aber nicht nur der aktiv und bewusst von den Nutzern erstellte Content ist im Rahmen der Digitalisierung von Bedeutung; ebenso Dinge wie Bewegungsdaten oder das Kaufverhalten, welche alle im Begriff Big Data vereint werden können. Diese gesammelten Daten sowie neue Ein- und Ausgabegeräte verändern das tägliche Leben vom Beruf (Industrie 4.0) über Freizeit (Future Mobility, Smart City) bis hin zum Heim (Smart Home). Als Überbegriff der über das Internet verbundenen und miteinander kommunizierenden Geräte steht das „Internet der Dinge“. [3]

### 3.1.2 Internet der Dinge (Internet of Things IoT)

Das Internet der Dinge entwickelte sich aus der stetig wachsenden Anzahl von Geräten, die mit dem Internet verbunden sind. Ein geläufiger Vertreter davon sind die Mobiltelefone bzw. die Smartphones. Vor allem durch die Einführung des iPhones bekam die Verbreitung internetfähiger Endgeräte einen gewaltigen Schub. Mittlerweile werden aber nicht nur Smartphones miteinander verbunden, sondern auch immer mehr Geräte des täglichen Lebens. Laut den Experten des IEEE (Institute of Electrical and Electronics Engineers) wird sich die Zahl der vernetzten Gegenstände bis zum Jahr 2020 auf rund 100 Mrd. belaufen. Dazu gehören dann vor allem auch Maschinen aus der Industrie, aber auch alles, was man sich vorstellen kann: vom Kühlschrank über die allgemeine Haustechnik, Fahrzeuge, Sensoren aller Art bis hin zu sogar Warenverpackungen. [4]

Durch die derzeit mit dem Internet verbundenen Geräte werden sehr viele Daten gesammelt, die bisher nicht oder nur in geringem Maße verwendet werden. Als Beispiel dafür, was bisher schon verwendet wird und viele nutzen, ohne dass ihnen der Zusammenhang bekannt ist, kann man z.B. Google Maps nennen. Hierbei werden die Position- und Bewegungsdaten der Android Benutzer, die Google Maps aktiviert und der Positionsbestimmung zugestimmt haben, übertragen und ausgewertet. Auch andere Hersteller wie TomTom nutzen die Daten, der mit den sogenannten Live-Diensten verbundenen Geräten. [5]

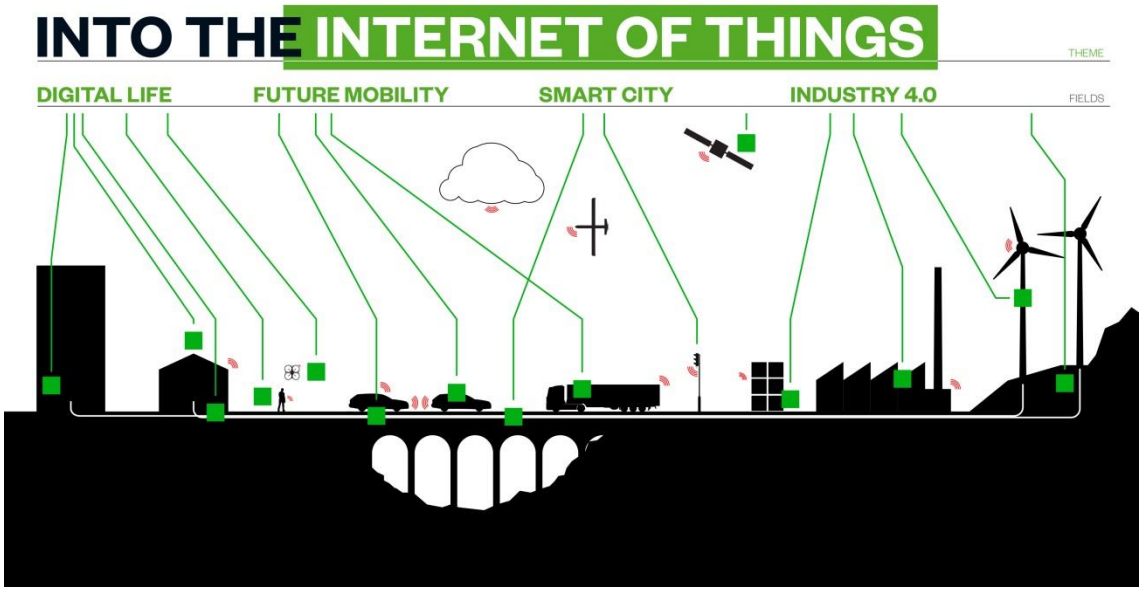


Abbildung 3-1: Internet der Dinge und dessen Einsatzfelder [6]

In Abbildung 3-1 ist eine stark vereinfachte Übersicht zu sehen, inwieweit das Internet der Dinge Einzug in diverse Bereiche des täglichen Lebens gefunden hat. Bei der Verbindung von physikalischen Geräten mit der digitalen Welt spricht man von sogenannten Cyber-Physical Systems.

### 3.1.3 Cyber-Physical Systems

Der Begriff Cyber-Physical Systems beschreibt die Verbindung eingebetteter Systeme in physikalischen Geräten, die sowohl als Sensoren als auch Aktuatoren fungieren, mit den globalen digitalen Netzen (Cyberspace). Dies ermöglicht durch die Nutzung der heute verfügbaren digitalen Netzinfrastruktur eine Vielfalt von Anwendungen. Das dabei entstehende Potential kann jedoch erst dann sinnvoll genutzt werden, wenn entsprechende neue Systeme entwickelt werden. Vor allem die umfangreichen Interaktionsmöglichkeiten dieser Systeme schaffen neue Einsatzmöglichkeiten. Dabei ist die Interaktion der Systeme mit ihrer Umwelt über Sensoren, Aktuatoren oder eine Benutzerschnittstelle ebenso zu berücksichtigen wie die Interaktion der Systeme in die entsprechenden Netze und untereinander. In diesem Zusammenhang liegt die Herausforderung darin, die unterschiedlichen Teilnetze optimal miteinander zu verbinden. Aus diesem Grund werden übergreifende Lösungen nötig sein, um die Möglichkeiten der Cyber-Physical Systems umfangreich nutzen zu können. [7]

### 3.1.4 Ubiquitous Computing

Der Begriff des Ubiquitous Computing (Ubicomp), also der Rechnerallgegenwärtigkeit, beschreibt die rechnergestützte Informationsverarbeitung, die in den letzten Jahren, bedingt durch die immer kleiner und günstiger werdenden Mikrocontroller, enorm zugenommen hat. Diese Kleinstcomputer können aufgrund ihres teils sehr geringen Energiebedarfs in viele Alltagsgegenstände eingebaut werden, um diesen mithilfe von einem zugewiesenen Verhalten einen Mehrwert zu verschaffen. Dies kann so weit gehen, dass selbst Gegenstände, die auf den ersten Blick nicht einmal als elektronisches Gerät zu erkennen sind, damit ausgestattet sind. Dadurch entsteht nicht nur die Kommunikation zwischen Menschen und weiteren Geräten, sondern auch eine teils nicht wahrgenommene Kommunikation der Geräte untereinander. Diese Idee und Entwicklung ist nichts Neues: Bereits 1988 wurde der Begriff schon von Mark Weiser verwendet, der ihn in seinem Aufsatz „The Computer for the 21st century“ letztendlich geprägt hat. [8]

> *„In the 21st century the technology revolution will move into the everyday, the small and the invisible“ „Im 21. Jahrhundert wird die technologische Revolution das Alltägliche, Kleine und Unsichtbare sein.“ – Mark Weiser [8]*

Dieses ubiquitäre Einbinden von elektronischen Geräten in den Alltag zeichnet sich nicht nur durch offensichtlich elektronische Geräte aus. Es können auch einfach nur Aufkleber sein, die mit einem Lesegerät/Empfänger Daten auslesen und diese weiterverarbeiten. Hierzu zählen die RFID-Chips (Radio Frequency Identification). Mit deren Hilfe lassen sich sämtliche Gegenstände ausstatten, um sie identifizieren. Wären alle Verpackungen damit ausgestattet, würde dies z.B. dazu führen, dass der Kühlschrank automatisch weiß, was er beinhaltet und sämtliche Details wie Mindesthaltbarkeitsdatum der Gegenstände kennt. [9]

Mittlerweile hat auch die Industrie den Vorteil von selbständig untereinander kommunizierenden Maschinen entdeckt. Die sogenannte vierte industrielle Revolution, auch Industrie 4.0 genannt, beschäftigt sich derzeit in erheblichem Maße mit der Entwicklung neuer Systeme und Anwendungen.

### 3.1.5 Industrie 4.0

Der Begriff Industrie 4.0 steht für die vierte industrielle Revolution. Als die erste industrielle Revolution wird die Einführung mechanischer Produktionsanlagen, die mit Hilfe von Wasser- und Dampfkraft betrieben wurden, betitelt. Die Hinzunahme bzw. die Verwendung von elektronischer Energie zur arbeitsteiligen Massenproduktion führte zur zweiten Industriellen Revolution. Bei der dritten Revolution kam die Elektronik und IT zum Einsatz, die eine weitere Automatisierung der Produktion ermöglichte. Nun, in einem etwas kürzeren Abstand zur vorherigen Revolution, findet man den Begriff der Cyber-Physical Systems wieder. Bei der nun vierten Revolution wird die Automatisierung um vernetzte und kommunizierende Systeme erweitert. [10]

Für den Begriff Industrie 4.0 besteht keine genau festgelegte Definition. Armin Roth versucht diese in seinem Buch „Einführung und Umsetzung von Industrie 4.0“ wie folgt zu festzuhalten.

> *„Industrie 4.0 umfasst die Vernetzung aller menschlichen und maschinellen Akteure über die komplette Wertschöpfungskette sowie die Digitalisierung und Echtzeitauswertung aller hierfür relevanten Informationen, mit dem Ziel die Prozesse der Wertschöpfung transparenter und effizienter zu gestalten, um mit intelligenten Produkten und Dienstleistungen den Kundennutzen zu optimieren.“ [10]*

Die verschiedenen industriellen Entwicklungsstufen sind in folgender Abbildung noch einmal kurz zusammengefasst. Dabei ist zu sehen, dass zwischen den ersten drei Stufen immer ca. 100 Jahre lagen, wobei hingegen von der dritten zur vierten lediglich ca. 50 Jahre lagen. Dies zeigt zudem, wie schnell sich bisherige Verfahren in der Industrie aufgrund der Entwicklungen im IT-Bereich ändern, bzw. sich daran anpassen. Aus diesem Grund ist es wichtig, die aktuellen Entwicklungen dahingehend zu untersuchen, welche Technologien zukunftsorientiert sind.

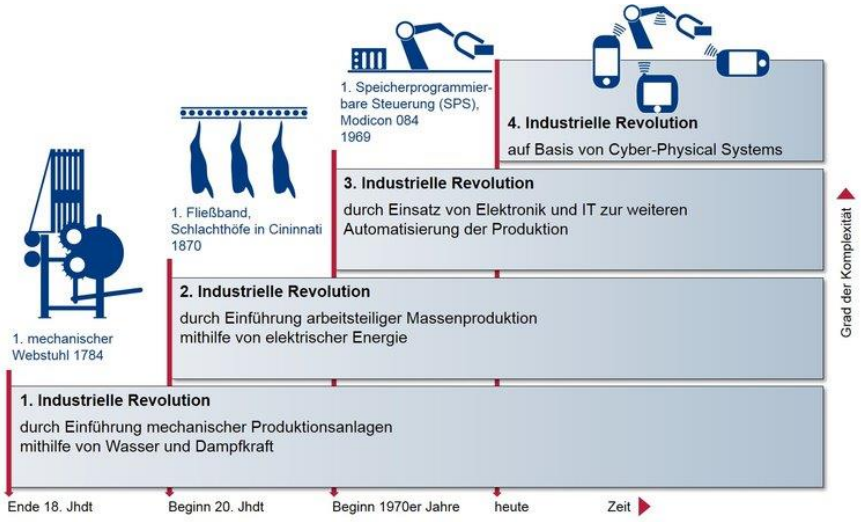


Abbildung 3-2: Übersicht der industriellen Entwicklung [11]

Dabei bietet die vierte Revolution zwei unterschiedliche Ansätze, wie mit den neuen Möglichkeiten die komplette Wertschöpfungskette neu gestaltet werden kann. Zum einen besteht die Möglichkeit, bestehende Modelle mit Hilfe der neuen Technologie zu optimieren und zum anderen gänzlich neue Strukturen zu schaffen und völlig neue Geschäftsmodelle zu entwickeln. Derzeit liegt der Fokus vor allem auf ersterem, bei dem die aktuellen Informations- und Kommunikationstechnologien mit der Produktion- und Automatisierungstechnik kombiniert werden. Daraus resultiert eine neue Stufe der Organisation und Steuerung der gesamten Wertschöpfungskette, die eine Flexibilisierung und Verbesserung dieser anstrebt. Mit der Digitalisierung der Wertschöpfungskette gewinnen digitale Plattformen an ökonomischer Bedeutung. [10]

### 3.1.6 Plattformen

Durch die Digitalisierung und Entwicklung des Internets der Dinge ergeben sich neue Geschäftsfelder. Die Masse an Daten, die aus den verschiedensten Quellen stammen können, müssen gesammelt, geeignet gespeichert und verarbeitet werden. Vor allem PaaS (Platform as a Service) und Cloud Computing spielen dabei eine große Rolle. Zum einen sind diese reine PaaS Angebote, teils mit rudimentärer SaaS (Software as a Service), im Angebot, zum anderen bieten große Softwarehersteller Komplettpakete (Suiten) an. Einige dieser IoT-Lösungen werden unter *3.2 Proprietäre IoT-Lösungen* vorgestellt. Bei der Untersuchung der Anwendungsschicht-Protokolle wird auch geprüft, welche von den Komplettlösungen verwendet werden und welche PaaS-Dienste die Protokolle unterstützen.

### 3.1.7 Protokollschichten

Damit eine Kommunikation von elektronischen Geräten überhaupt stattfinden kann, bedarf es einer Verbindung dieser. Die Kommunikation funktioniert durch den Austausch von Nachrichten. Voraussetzung für eine Verständigung ist die klare Definition der Kommunikationssprache (Protokoll). Diese findet nicht nur horizontal, also mit der gleichen/korrespondierenden Ebene (Schicht) des anderen Gerätes statt, sondern auch vertikal innerhalb des Gerätes. Die unterschiedlichen Sichten haben jeweils eine bestimmte Funktion für den Nachrichtenaustausch. Dies beginnt bei der Bitübertragung auf der physikalischen Ebene und endet bei der Darstellung am Bildschirm. Über Schnittstellen kommunizieren die Schichten mit den darüber- und darunterliegenden Ebenen. Da jede Schicht in sich abgeschlossen ist, können diese problemlos verändert oder ersetzt werden. Auch im Internet der Dinge werden verschiedene Protokolle verwendet. Zur besseren Übersicht der verschiedenen Schichten bestehen unterschiedliche Schichtenmodelle. Im Folgenden wird das für diese Arbeit relevante Modell, das TCP/IP-Referenzmodell, kurz erklärt sowie das OSI-7-Schichtenmodell vorgestellt. [12]

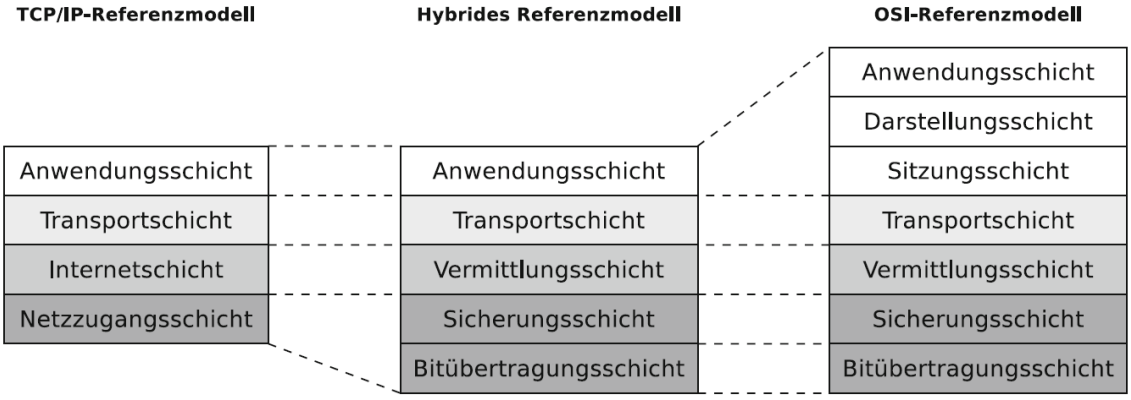


Abbildung 3-3: Übersicht der verschiedenen Referenzmodelle [12]

#### 3.1.7.1 TCP/IP-Referenzmodell

Das TCP/IP-Referenzmodell, auch unter dem Namen DoD-Schichtenmodell bekannt, wurde ab 1970 vom Department of Defense (DoD) der USA entwickelt. Dabei wurden die jeweiligen Aufgaben, die bei der Kommunikation anfallen, in übereinanderliegende Schichten aufgeteilt. Für das TCP/IP-Referenzmodell wurden dabei vier Schichten definiert (siehe Abbildung 3-3, links). Dabei wurde festgelegt, welche Anforderungen diese in Bezug auf die Kommunikation erfüllen müssen. Die genaue Umsetzung wurde nicht vorgegeben, weshalb für jede Schicht unterschiedliche Protokolle existieren. Von jeder Schicht werden zusätzliche Informationen, die vom Empfänger der gleichen Schicht ausgewertet werden, hinzugefügt. [12]

#### 3.1.7.2 Hybrides Referenzmodell

Da die Netzzugangsschicht zwei unterschiedliche Aufgabenbereiche abzudecken hat, wird das Referenzmodell oft auf fünf Schichten erweitert, bei dem die Netzzugangsschicht in die Sicherungsschicht und die Bitübertragungsschicht aufgeteilt wird (siehe Abbildung 3-3, Mitte). Die Internetschicht unterscheidet sich von der Vermittlungsschicht nur im Namen. [12]

#### 3.1.7.3 ISO/OSI-7-Schichtenmodell

Das ISO/OSI-7-Sichtenmodell wurde 1983 von der Internationalen Organisation für Normierung (ISO) standardisiert. Dieses Modell ist dem hybriden Referenzmodell sehr ähnlich, unterteilt die Anwendungsschicht jedoch noch einmal in drei Schichten. Dabei werden manche Aufgaben der Anwendungsschicht der Sitzungsschicht und der Darstellungsschicht zugeordnet (siehe Abbildung 3-3, rechts). [12]

#### 3.1.7.4 Aufgaben der einzelnen Schichten

In diesem Abschnitt werden die wesentlichen Aufgaben der einzelnen Schichten aus dem hybriden Referenzmodell beschrieben.

**Anwendungsschicht**

Die Anwendungsschicht (Application Layer) beinhaltet die Protokolle, die von den Anwendungsprogrammen verwendet werden. Dies kann z.B. ein Browser, ein E-Mail Programm oder ein MQTT-Broker (näheres dazu in Kapitel 0) sein. Hier werden die Nachrichten, die die eigentliche Kommunikation darstellen, verarbeitet. Hier wird auch die Steuerung (Auf-/Abbau) der Verbindungen mit anderen Anwendungen gesteuert und die Darstellung der Informationen verarbeitet. [12]

Beispiele sind hier: HTTP, POP, FTP sowie MQTT oder CoAP.

**Transportschicht**

Die Transportschicht (Transport Layer) ist für den Transport der Nachrichten zwischen den Prozessen der unterschiedlichen Geräte zuständig. Dies geschieht über eine sogenannte Ende-zu-Ende-Kommunikation. Dabei werden die Pakete der Anwendungsschicht in Segmente verpackt und mit zusätzlichen Daten der Transportschicht ergänzt (Header). Auf der Seite des Empfängers erkennt die korrespondierende Schicht diese Segmente in der Vermittlungsschicht. Dabei geschieht die Adressierung über Portnummern der Anwendungen, um die Segmente korrekt zuordnen zu können. Unterschiedliche Transportprotokolle bieten verschiedene Formen der Kommunikation: **verbindungslose** und **verbindungsorientierte Kommunikation**. Die Unterschiede dabei sind auch für die Anwendungsschicht-Protokolle für das Internet der Dinge wichtig, da dabei unterschiedliche Stufen der Zuverlässigkeit entstehen. Bei der verbindungslosen Kommunikation werden, wie bei einem Brief, die Nachrichten verschickt, ohne zuvor eine Verbindung aufgebaut zu haben. Es gibt demzufolge auch keine Kontrolle darüber, ob eine Nachricht angekommen ist oder nicht. Soll eine entsprechende Kontrolle über eine verbindungslose Kommunikation hinzugefügt werden, ist dies Aufgabe der Anwendungsschicht. Dieses Fehlen einer Zustellgarantie ist ein Nachteil dieser Kommunikationsform, der höhere Datendurchsatz im Vergleich zur verbindungsorientierten Kommunikation ein Vorteil. Bei der verbindungsorientierten Kommunikation ist die Zustellgarantie möglich. Diese funktioniert ähnlich wie ein Telefon: Vor dem Austausch der Nachrichten wird zunächst eine Verbindung mit dem Empfänger hergestellt und nach Beendigung der Übertragung wieder beendet. Zusätzlich kann bei dieser Übertragungsform eine Datenflusskontrolle verwendet werden, bei der der Empfänger die Sendegeschwindigkeit des Senders steuern kann. Neben der Zustellgarantie werden die Segmente auch reihenfolgegetreu übertragen. [12]

Beispiele sind hierfür: UDP (User Datagram Protocol) bei der verbindungslosen Kommunikation, TCP (Transport Control Protocol) bei der verbindungsorientierten Kommunikation.

### Internetschicht/Vermittlungsschicht

Die Vermittlungsschicht (Network Layer) – oder auch Internetschicht (Internet Layer) beim TCP/IP-Referenzmodell – ist für die Weitervermittlung der Daten (Pakete) in einem logischen Netz über physikalische Übertagungsabschnitte zuständig, dem sogenannten Routing. Für die Kommunikation werden in einem TCP/IP-Netzwerk logische Adressen (IP-Adressen) definiert. Dabei werden logische Subnetze durch Router begrenzt. Auch hier werden den Daten der Transportschicht weitere Informationen zur Vermittlung angefügt (Header). [12]

Beispiele sind hierfür: IP (Internet Protocol), ein verbindungsloses Protokoll, und XIP (Extended Internet Protocol), welches jedoch kaum verwendet wird.

### Netzzugangsschicht – Sicherungsschicht

Die Sicherungsschicht (Data Link Layer) der Netzzugangsschicht (Network Access Layer) ist für die Fehlererkennung zuständig. Dazu werden die Pakete der Vermittlungsschicht in Rahmen (sogenannte Frames) verpackt. Zur Übertragung sind physische Adressen (MAC-Adressen) notwendig. Diese werden zusammen mit einer Prüfsumme zur Fehlererkennung an jeden Rahmen angefügt. Die Übertragung findet zuverlässig statt. Wird ein Fehler erkannt, wird der fehlerhafte Rahmen vom Empfänger verworfen, jedoch von der Sicherungsschicht nicht wieder neu angefordert. Dies ist Aufgabe der Transportschicht. Eine Verbindung findet auf dieser Schicht ausschließlich innerhalb von physischen Netzen statt, welche über Bridges und Switches miteinander verbunden sein können. [12]

### Netzzugangsschicht – Bitübertragungsschicht

Die Bitübertragungsschicht (Physical Layer) der Netzzugangsschicht ist für die Übertragung über verschiedene Übertragungsmedien zuständig. Hier findet die letztendliche Übertragung der Nullen und Einsen statt. Dabei wird die verwendbare Übertragungsgeschwindigkeit definiert und festgelegt, ob die Übertragung nur in eine Richtung oder gleichzeitig in beide Richtungen stattfinden kann. [12]

### 3.1.8 Übertragungsprotokolle und -techniken im Internet der Dinge

Im Bereich des Internets der Dinge werden noch weitere Protokolle und Techniken neben TCP/IP und UDP unterhalb der Anwendungsschicht bzw. im kompletten Protokollstack verwendet. Diese sind dann oft nicht direkt, sondern nur indirekt über einen Broker oder Gateway mit dem Internet verbunden. Aber gerade im Bereich der Sensoren, die einen hohen Anspruch an einen niedrigen Energiebedarf stellen, sind diese Technologien unverzichtbar und werden deshalb auch in dieser Arbeit vorgestellt. Im gesamten Spektrum der Sensoren besteht eine Vielzahl an verschiedenen Übertragungstechniken von verschiedenen Herstellern. Vermehrt wird jedoch versucht, diese über Gateways zusammenzuführen. Zudem arbeiten einige Hersteller im Verbund an Protokollen und Standardisierungen. Durch die folgende Einführung soll verständlich gemacht werden, wo die Unterschiede und Gemeinsamkeiten der verschiedenen Techniken liegen.

#### 3.1.8.1 TCP/IP und UDP

Die Aufgaben von TCP/IP und UDP wurden unter 3.1.7.4 schon beschrieben. Zum besseren Verständnis im Vergleich mit den anderen hier aufgeführten Übertragungsprotokollen ist in Abbildung 3-4 ein vereinfachter Aufbau des Protokollstacks zu sehen.

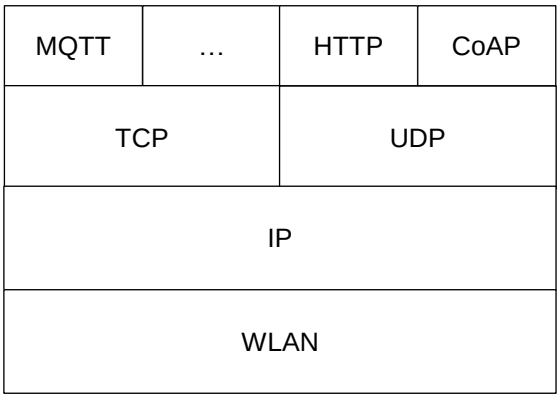


Abbildung 3-4: TCP/IP Protokollstack

#### 3.1.8.2 WLAN

WLAN steht für Wireless Local Area Network und bezeichnet damit die lokalen Funknetze. Diese basieren in der Regel auf den Standards der Normenfamilie IEEE 802.11 [12]. Der größte Teil der auf TCP/IP basierenden Anwendungen im Internet der Dinge benutzen diese Art der Übertragung. Ein Vorteil bei der Nutzung dieser Technik ist, dass die Sensoren und Aktuatoren direkt mit dem Internet verbunden werden können, ein Nachteil ist jedoch der verhältnismäßig hohe Stromverbrauch. Will man aber z.B. Messwerte eines Sensors auf einer Internetplattform sammeln, so müssen die Daten spätestens über ein sogenanntes Gateway (meist eben auch im WLAN eingebunden) ins Internet transportiert werden. Die vom Internet unabhängigen Sensornetzwerke werden WSN (Wireless Sensor Network) genannt.

#### 3.1.8.3 Wireless Sensor Network

Wireless Sensor Networks (WSN) oder manchmal auch Wireless Sensor and Actuator Networks (WSAN) beschreiben flächendeckend verteilte, autonome Sensoren, die diverse Umweltbedingungen wie Temperatur oder Feuchtigkeit erfassen und ihre Daten an eine zentrale Einheit weitergeben. Modernere Netzwerke arbeiten mittlerweile oft bidirektional: Es besteht eine gewisse Kontrolle über die Sensoren, die somit auch Aufgaben von Aktuatoren übernehmen können. Die Entwicklung von drahtlosen Sensornetzwerken wurde von militärischen Einsatzgebieten wie der Gefechtsfeldüberwachung vorangetrieben. Heutzutage werden solche Sensornetzwerke in vielen industriellen und verbraucherorientierten Anwendungen eingesetzt. In vielen Anwendungsbereichen sind die Sensornetzwerke über ein Gateway an ein lokales Netzwerk angebunden, das die Daten an andere Geräte mit mehr Ressourcen zur Weiterverarbeitung weitergibt. In diesem Zusammenspiel können diese Netzwerke auch für das Internet der Dinge berücksichtigt werden. [13] Ein mittlerweile vielseitig eingesetztes Produkt, vor allem im Konsumentenbereich, ist ZigBee, welches zur Übertragung den IEEE 802.15.4 Standard nutzt.

#### 3.1.8.4 IEEE 802.15.4

Der IEEE 802.15.4 Standard ist eine Spezifikation für einfache drahtlose Netzwerke, die nur eine geringe Übertragungsrate benötigen. Neben den bereits vorhandenen drahtlosen Übertragungsarten wie WLAN und Bluetooth (Classic), welche durch ihre hohen Übertragungsarten, den komplexen Aufbau und dem damit einhergehenden hohen Energieverbrauch für Sensornetzwerke eher ungeeignet sind, bietet dieser Standard eine sinnvolle Ergänzung. Durch die Vermeidung unnötiger Steuerungsinformationen kann die Übertragungszeit sehr gering gehalten werden. Aufgrund des einfacheren Aufbaus wird auch weniger komplexe Hardware benötigt, was die Herstellung nicht nur vereinfacht, sondern auch günstiger macht. Durch diese Optimierung können die Funkmodule je nach Anwendungsfall, bei dem z.B. nur in großen Intervallen gesendet wird, über mehrere Jahre hinweg betrieben werden, ohne dass die Batterie getauscht werden muss. Der IEEE 802.15.4 Standard besteht aus zwei Schichten, der PHY-Schicht und der MAC-Schicht, welche der Bitübertragungsschicht und der Sicherungsschicht aus dem unter 3.1.7.2 beschriebenen Referenzmodell entsprechen. [14]

#### 3.1.8.5 ZigBee

Bei der im Jahre 2002 gegründeten ZigBee-Allianz handelt es sich um einen Zusammenschluss von verschiedenen Industrieunternehmen, die offene, weltweite Standards für Funknetze bereitstellen wollen, die für Überwachungs- und Steuerungsaufgaben einsetzbar sind. Im Folgenden werden Grundlagen zu der ZigBee-Übertragungstechnik vorgestellt. [14]

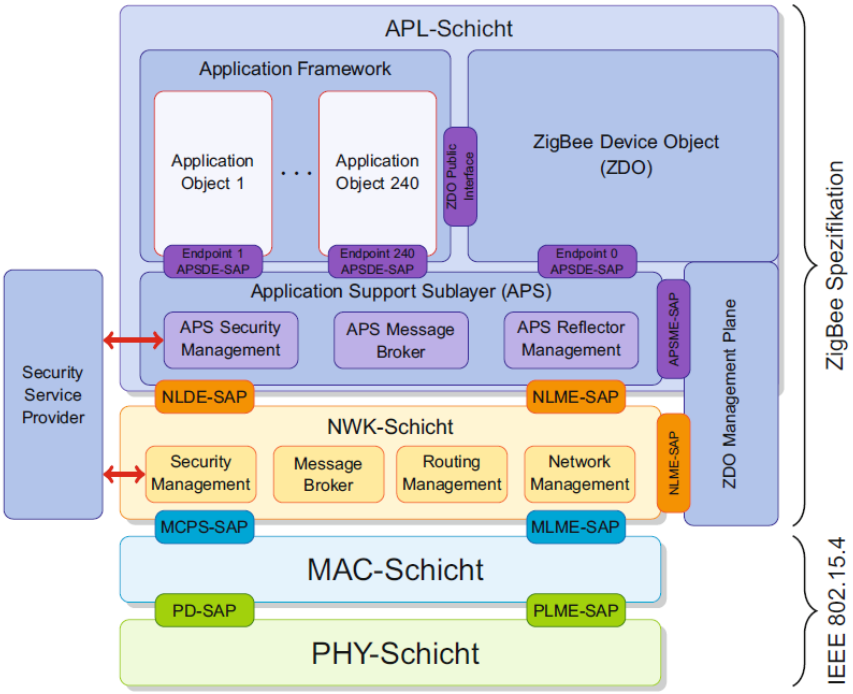


Abbildung 3-5: ZigBee Architektur [14]

Genau wie der TCP/IP-Protokollstack besteht die ZigBee-Architektur auch aus vier Schichten. Jedoch wurde hier die Netzzugangsschicht, wie beim hybriden Referenzmodell, bereits in zwei Schichten aufgeteilt. Somit bestehen oberhalb der Bitübertragungsschicht (**PHY-Schicht**) und der Sicherungsschicht (**MAC-Schicht**) lediglich nur zwei Schichten, von denen die obere aber intern wiederum nochmal unterteilt ist. Auf der Sicherungsschicht liegt die Netzwerkschicht (**NWK-Schicht**) auf, deren Hauptaufgaben die Konfiguration, also die Rollenzuweisung eines Funkmoduls, das Routing, also die Zustellung der Pakete, die Adresszuweisung sowie das Netzwerkmanagement sind. Die Anwendungsschicht wiederum besteht aus drei Bausteinen: Der Anwendungsunterstützungsschicht (**APS-Schicht**), dem ZigBee Device Object (ZDO) inklusive ZDO Management Plane und dem Anwendungsframework. Eine zu entwickelnde Anwendung wird in diesem Framework eingebettet, indem sie in mehrere, bis zu 240 Anwendungsobjekte aufgeteilt wird, bei denen jedes Objekt eine bestimmte Aufgabe übernimmt. Das ZDO initialisiert die APS-Schicht, die NWK-Schicht und den Sicherheitsserviceprovider (Security Service Provider). [14]

In einem ZigBee-Netzwerk können die Funkmodule jeweils eine von drei verschiedenen Rollen einnehmen. In jedem Netzwerk gibt es genau ein Funkmodul, das die Rolle des **ZigBee-Koordinators** übernimmt. Dieser ist für den Start und die Festlegung der wichtigsten Parameter zuständig und übernimmt zudem auch die Aufgaben des Routers. Über den **ZigBee-Router** können andere Funkmodule dem Netzwerk beitreten. Er übernimmt dabei die Wegentdeckung und die Weiterleitung der Daten. Diese erhalten sie wiederum von den **ZigBee-Endgeräten**, welche ausschließlich mit dem zugeordneten Router kommunizieren und zwischen dem Senden in einen Schlafmodus versetzt werden können. Dies führt zu erheblich längeren Batterielaufzeiten. [14]

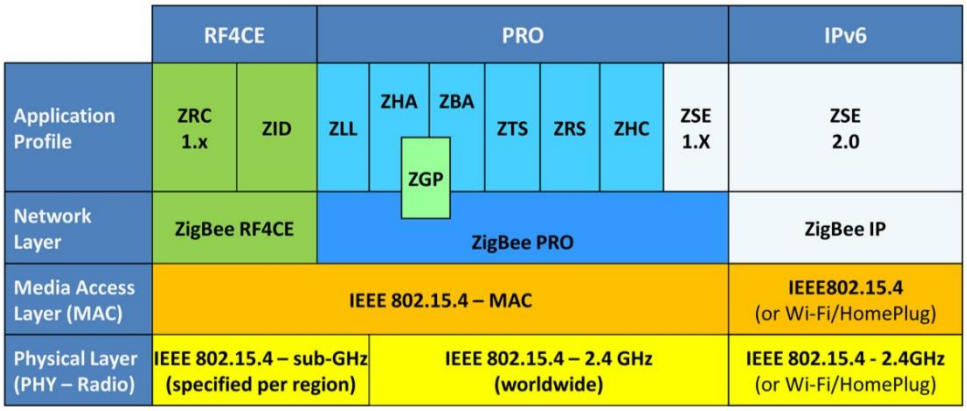


Abbildung 3-6: Übersicht über die ZigBee-Standards [15]

Unter ZigBee 3.0 wurden drei verschiedene Standards auf der Vermittlungsschicht (Network Layer) definiert, welche unterschiedliche ZigBee-Profile unterstützen. Interessant ist dabei vor allem der ZigBee IP Standard, welcher es möglich macht, nicht nur über IEEE 802.15.4 zu kommunizieren, sondern auch über WiFi. Dabei wird das Anwendungsprofil ZSE 2.0 (ZigBee Smart Energy) verwendet. ZigBee IP basiert dabei auf dem 6LoWPAN-Protokoll (IPv6 over Low power Wireless Personal Area Network). Dadurch können die Komponenten direkt an das Internet und somit auch an das Internet der Dinge angebunden werden. [16]

#### 3.1.8.6 X-Bee

Oft wird X-Bee fälschlicherweise als Synonym für ZigBee verstanden. X-Bee ist jedoch kein Standard, sondern ein Produkt der Firma Digi International, welche Funkmodule herstellt. Die Verwechslung kommt auch daher, dass X-Bee neben erweiterten, eigenen Standards auch den ZigBee-Standard unterstützt und in diesem Zusammenhang deshalb oft verwendet wird.

#### 3.1.8.7 Thread

Der Thread-Stack, der sich über die Sicherungsschicht, die Transportschicht und die Vermittlungsschicht erstreckt, ist ein offener Standard für eine zuverlässige, kosteneffektive, energiesparende, drahtlose Kommunikation zwischen Geräten. Dabei liegt das Hauptaugenmerk auf „Connected Home“-Applikationen, in IP-basierten Netzwerken und mit der Möglichkeit, von diversen Anwendungsschichtprotokollen genutzt zu werden. Zu den grundlegenden Charakteristiken des Standards zählt die einfache Installation und Benutzung des Netzwerks, welches selbstkonfigurierend ist und auftretende Routingprobleme selbst löst. Dabei werden neue Geräte erst nach der Authentifizierung dem Netzwerk hinzugefügt, in dem die Kommunikation verschlüsselt stattfindet. Die Netzwerkgröße kann von einigen wenigen bis hin zu hunderten Geräten variieren. Die Netzwerkschicht ist so entwickelt, dass sie sich den Bedürfnissen der Benutzung anpasst, um eine nahtlose Kommunikation aller Geräte sicherzustellen. [17]

Im Thread Netzwerk wird zwischen drei bzw. vier Geräten unterschieden (drei Geräte, von denen eines auch als ein anderes eingesetzt werden kann). Diese werden im Folgenden kurz beschrieben.

### Border Router

Ein Border Router ist ein spezieller Router, der die Verbindung zwischen dem 802.15.4 Netzwerk und dem angrenzenden Netzwerk auf einer anderen physikalischen Schicht herstellt (Gateway). Dabei kann es sich dann z.B. um ein Ethernet oder WLAN handeln. Innerhalb des Netzwerks kann es mehrere dieser Border Router geben, was dem Single Point of Failure entgegenwirkt und das Netzwerk damit robuster macht. [17]

### Router

Die Router wiederum bilden die Schnittstelle zwischen den Geräten und den Border Routern und übernehmen dabei das Routing sowie die Sicherheitsdienste innerhalb des Netzwerks. Die Router sind in ihrer Funktion nicht dafür ausgelegt, in eine Art Standby-Modus (Sleep) zu wechseln. Sie können jedoch auch auf die Funktionalität eines Endgerätes zurückgestuft werden. In diesem Fall bezeichnet man die Router als REED (Router-eligible End Device). [17]

### Router-eligible End Device

Ist ein Router als REED eingerichtet, fungiert dieser nur als normales Endgerät ohne irgendwelche Routerfunktionalitäten zu übernehmen. Abhängig von den Bedingungen im Netzwerk und dessen Topologie ist es jedoch möglich, dass diese völlig selbständig, ohne Benutzerinteraktion, kurzzeitig als Router verwendet werden können. [17]

### Sleepy End Device

Die Sleepy End Devices sind die eigentlichen Host-Geräte (Sensoren, Aktuatoren). Diese kommunizieren lediglich mit ihrem übergeordneten Router und können keine Nachrichten zu anderen Geräten weiterleiten. [17]

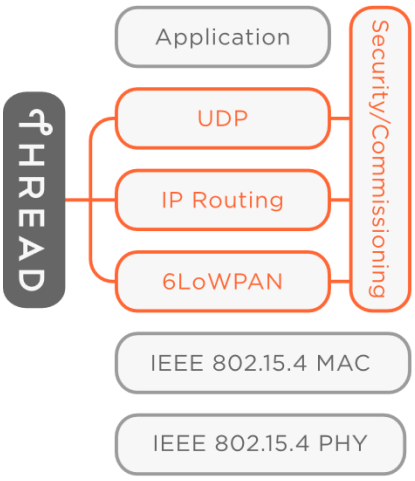


Abbildung 3-7: Der Thread Protokollstack [18]

Wie in Abbildung 3-7 zu sehen ist, übernimmt Thread die gesamte Verwaltung des Netzwerks unterhalb der Anwendungsschicht und über der Netzzugangsschicht. Ziel des Ganzen war es, die Adressierung in einem allgemeineren Verfahren vorzunehmen. Durch die Verwendung von 6LoWPAN erhält jedes Endgerät eine IP-Adresse (IPv6). Dies erlaubt eine direkte und einfache Kommunikation eines Plattformdienstes mit einem Endgerät. Thread definiert zwar keine eigene Anwendungsschicht und somit auch nicht, auf welche Art und Weise die Geräte mit dem Netzwerk interagieren sollen, bietet jedoch eine generische Art der Kommunikation, mit der einfache Nachrichten ausgetauscht werden können. Deshalb ist Thread optimal, wenn man mit verschiedenen Anwendungen auf das Netzwerk zugreifen möchte. [18]

#### 3.1.8.8 Bluetooth LE (Smart/Smart Ready)

Bluetooth Classic ist ein Standard für Kommunikation über eine geringe Bandbreite und wird von der Bluetooth Special Interest Group (SIG) gewartet und weiterentwickelt. Ursprünglich wurde Bluetooth 1994 als drahtlose Alternative der Datenübertragung über Funkwellen entwickelt. Seit sich auf diesem Gebiet das WiFi durchgesetzt hat, fokussiert sich SIG auf die leichtgewichtige und energiesparende Kommunikation. Derzeit besteht Bluetooth aus vier verschiedenen Technologien: Bluetooth, Bluetooth EDR, Bluetooth HS und Bluetooth Low Energy, welche mit der Version 4.0 von Bluetooth eingeführt wurde und auf den Markt für energiesparende, drahtlose Kommunikation abzielt. [19]

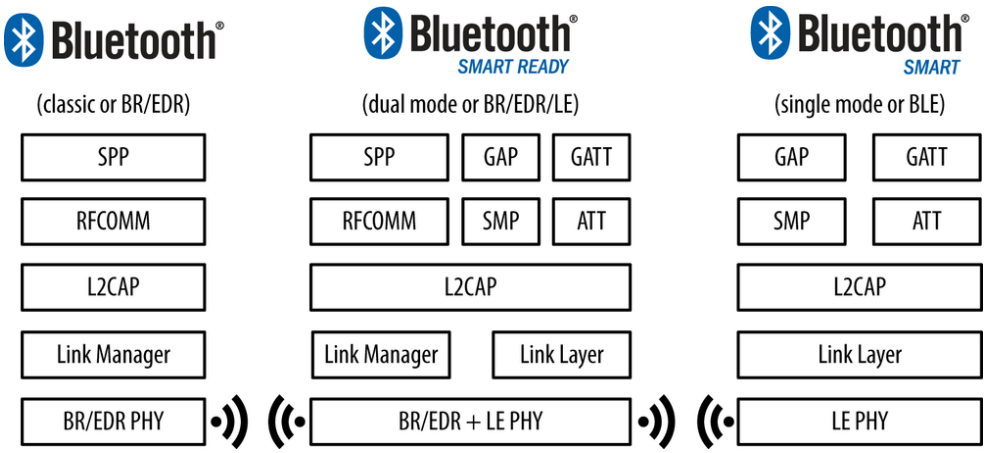


Abbildung 3-8: Übersicht über die Bluetooth Standards [20]

Wie in Abbildung 3-8 zu sehen, bestehen drei Arten von Bluetoothgeräten: Bluetooth, Bluetooth Smart Ready und Bluetooth Smart. Geräte, die nur mit „Bluetooth“ gekennzeichnet sind, können nur im klassischen Bluetoothnetz eingesetzt werden. Bluetooth Smart Ready Geräte hingegen können sowohl mit Bluetooth Classic Geräten kommunizieren als auch mit als Bluetooth Smart gekennzeichneten, die für sich nur Bluetooth LE als Übertragung verwenden. [19] Viele Smart Ready Geräte (wie z.B. Smartphones) haben neben Bluetooth auch noch andere Netzwerkanbindungen wie z.B. WiFi und können dadurch die Daten der Smart Endgeräte an das Internet weiterleiten. [20] Für diese Funktion können auch sogenannte Gateways verwendet werden.

Ein Endgerät sendet zur Verbindungsaufnahme ein sogenanntes Advertising, welches für alle verfügbaren Geräte empfangbar ist. Dadurch kann dann eine Verbindung zwischen beiden Geräten aufgebaut werden. Ein Vorteil dieses Advertisings ist, dass darin auch Daten enthalten sein können, die also abgerufen werden können, ohne dass eine Verbindung aufgebaut werden muss [21]. Diese Eigenschaft wird z.B. bei sogenannten Beacons verwendet, die lediglich ihre ID senden und damit zur Identifikation (und Entfernungsbestimmung über die Signalstärke) genutzt werden kann.

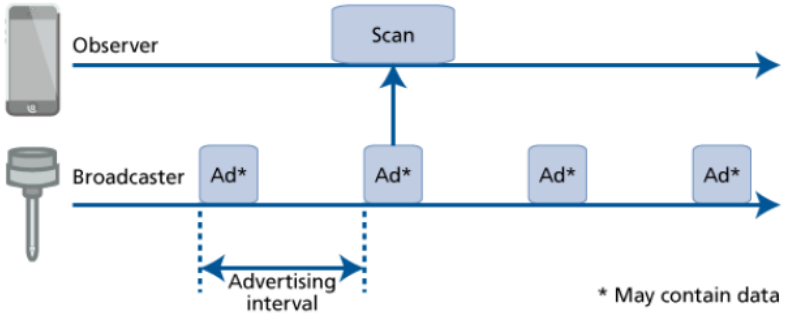


Abbildung 3-9: Bluetooth LE Advertising [21]

Im Fall, dass zwei Geräte miteinander verbunden sind, fungiert das Endgerät als Server und das damit verbundene Gerät als Client, welches die Daten vom Server in bestimmten Verbindungsintervallen erfragt. [21]

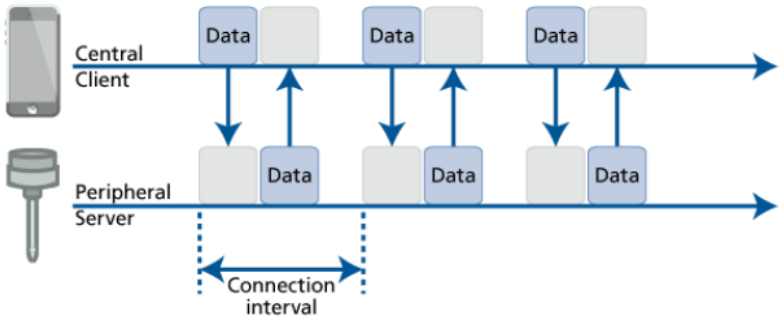


Abbildung 3-10: Bluetooth LE verbundene Geräte [21]

Der BLE Standard erlaubt neben der klassischen Adressierung durch GATT auch die Adressierung mit IP-Adressen (IPv6) durch die Verwendung von 6LoWPAN, wie auch Thread und ZigBee 3.0. Dadurch ist es möglich, über Gateways, alle Endgeräte direkt mit dem Internet zu verbinden, wobei diese dann auch selbst vom Internet aus ansteuerbar sind.

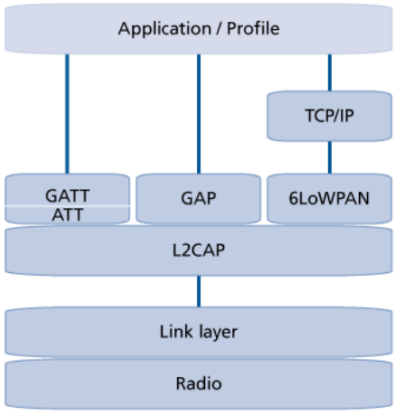


Abbildung 3-11: Bluetooth LE Protokollstack [21]

#### 3.1.8.9 6LowPAN

6LowPAN (IPv6 over Low-power wireless Personal Area Network) ist ein Kommunikationsprotokoll zur Funkdatenübertragung und schlägt damit die Brücke zwischen low-power Geräten im klassischen WSN, wie IEEE 802.15.4 oder Bluetooth LE, und anderen in IP basierten Netzwerken. Dabei werden die Sensornetzwerke für die Benutzung bzw. Adressierung mit IP Adressen (IPv6) erweitert. Aufgrund des hohen Adressraums von IPv6 kann jedem Gerät bereits in der Herstellung eine Adresse vergeben werden. Dabei müssen 6LowPAN basierte Anwendungen die Einschränkungen des Protokolls berücksichtigen. Da IP-Pakete im Vergleich zu den Paketen und WSNs viel größer sein können, kann es dazu kommen, dass diese vor der Übertragung fragmentiert werden. Wenn dabei ein einzelnes Fragment verloren geht, müssen alle Fragmente erneut übertragen werden. Dies kann wiederum zu einem schlechten Netzwerkverhalten führen. Deshalb müssen diese Paketverluste bei der Verwendung von 6LowPAN vor allem bei größeren Netzwerken berücksichtigt werden. Ein weiteres Merkmal ist die Header-Komprimierung. Dieser offene Standard wurde von der Internet Engineering Task Force (IETF) definiert. [22]

#### 3.1.8.10 Z-Wave

Bei Z-Wave handelt es sich um ein proprietäres Verfahren, welches von der dänischen Firma Zensys, gegründet 1999 um einen drahtlosen Standard für die Gebäudeautomatisierung zu definieren, entwickelt wurde. Die Z-Wave Alliance besteht aus über hundert Herstellern, die Produkte im Bereich der Heimautomatisierung herstellen. Dies führt dazu, dass im Vergleich zu anderen Technologien eine Vielzahl an Geräten auf dem Markt ist, vor allem im amerikanischen Markt. Das Protokoll ist im Vergleich zu ZigBee, was die direkte Konkurrenz darstellt, deutlich einfacher und erreicht zudem eine Batterielebensdauer von bis zu zehn Jahren. [23] Theoretisch könnten auch hier IPv6 Pakete über die verwendete Übertragungstechnik (ITU-T G.9959) versendet werden. [24] Da Z-Wave jedoch kein offener Standard ist und die verantwortlichen Schichten, die dies ermöglichen würden nicht veränderbar sind, fällt diese Möglichkeit hier weg.

### 3.1.8.11 Gesamtübersicht

Die folgende Abbildung ist als vereinfachte Übersicht zu sehen, in der der Zusammenhang und die Überschneidung der verschiedenen Techniken erkennbar sein soll. Die Grafik ist vertikal von oben nach unten zu lesen.

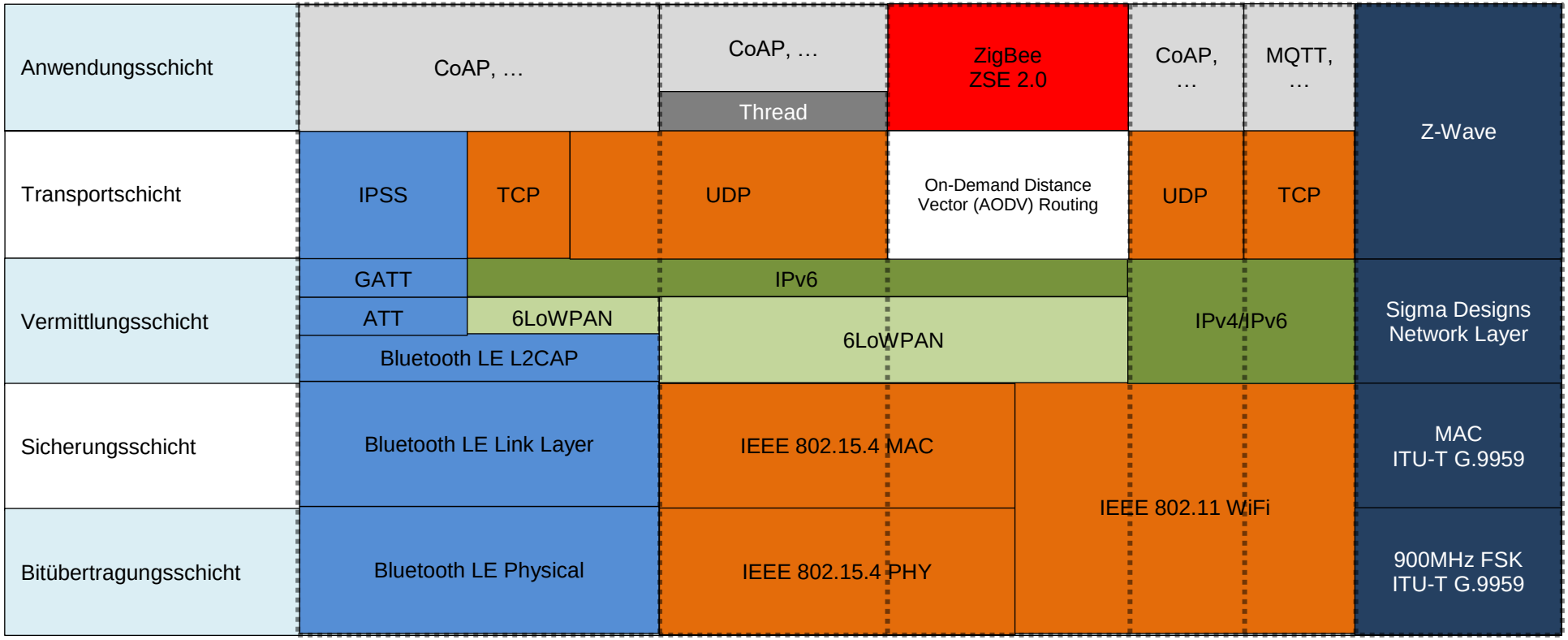


Abbildung 3-12: Gesamtübersicht der Übertragungstechniken

## 3.2 Proprietäre IoT-Lösungen

Bei den proprietären IoT-Lösungen besteht keine gleiche Vorgehensweise bei der Umsetzung. Es geht immer um die Vernetzung von Geräten und Objekten mit dem Internet und dem Verarbeiten der ausgetauschten Daten. Dabei kann es sich um Zustandsdaten handeln, die zwischen zwei Maschinen ausgetauscht werden, oder einfachen Identifizierungsdaten von mit Sendern versehenen Lager- und Logistikobjekten. Teils werden auch bereits bestehende Angebote aus dem Bereich Embedded Systems umgelabelt, ohne dass sich dabei viel am System verändert hat. In diesem Markt finden sich neben den großen IT-Firmen auch klassische Industriekonzerne wie Siemens oder Bosch. Gerade diese müssen sich auch mit der Digitalisierung, der nächsten Stufe der industriellen Entwicklung, beschäftigen, um zukunftsfähig zu sein. Die wichtigsten Aufgaben bei der Entwicklung des IoT liegen bei der Verwaltung, der Verarbeitung sowie der Sicherheit der Daten. Dies geschieht über sogenannte Plattformen. Diese Plattformen sind aber je nach Hersteller unterschiedlich in ihrer Funktionalität und der Verwendung von Protokollen. [25] Dies macht eine Standardisierung in diesem Bereich schwierig. Um einen kleinen Überblick über die derzeitige Entwicklung zu geben, werden im Folgenden einige Lösungen vorgestellt und auch geprüft, welche Anwendungsschichtprotokolle dabei verwendet bzw. als Input unterstützt werden. Diese Sammlung dient der Festlegung der zu untersuchenden Protokolle.

### 3.2.1 Microsoft

Microsoft hat bereits früh erkannt, dass es im Bereich IoT offene Protokollstandards geben sollte. Deshalb war Microsoft lange Zeit Mitglied in der Allianz AllSeen [26], wechselte aber Anfang 2016 zu der neu formierten Open Connectivity Foundation (OCF) [27], deren Ziel die Entwicklung von Spezifikationen und Sets an Protokollen ist. Noch 2016 sollen erste OCF-zertifizierte Produkte auf den Markt kommen [25]. Um die breite Verwendung bzw. einen möglichst reibungsloses Zusammenspiel verschiedener Geräte zu ermöglichen, hat sich die OCF mit der Thread Group zusammengetan [27]. Microsofts Windows-10-Geräte sollen den OCF-Standard nativ nutzen können. Auf der Client-Seite soll im Bereich IoT das Betriebssystem Windows 10 IoT Core, welches ohne Display auskommt und unter anderem auf dem Raspberry Pi läuft, eingesetzt werden. Im Bereich der Verwaltung und Verarbeitung der Daten entwickelt Microsoft die **Azure IoT Suite**, welche aus verschiedenen cloud-basierten Anwendungen besteht. Die für diese Arbeit interessantesten Anwendungen sind das Azure IoT Hub, welches die verschiedenen Geräte miteinander verbindet und Azure IoT Protokollgateway, über das Protokollanpassungen durchgeführt werden können. Durch zertifizierte Lösungen von Drittanbietern aus dem Microsoft Partnernetzwerk lässt sich die IoT-Plattformlösung erweitern. [25]

Das Azure IoT Hub bietet nativ die Unterstützung für die Anwendungsschicht-Protokolle **AMQP**, **MQTT** und **HTTP/1.1**. Da manche verwendeten Geräte diese Protokolle eventuell nicht unterstützen, bietet die Azure IoT Suite mit dem Protokollgateway, wie bereits erwähnt, die Möglichkeit der Protokollanpassung. [28]

### 3.2.2 Cisco Jasper

Seit der Übernahme von Jasper Technologies durch Cisco für 1,4 Milliarden Dollar spielt Cisco Jasper eine wichtigere Rolle im Bereich IoT [29]. Cisco übernimmt damit von Jasper die Plattform **Jasper Control Center**, eine vollständige und sofort nutzbare SaaS-Lösung (Software as a Service) für eine weltweite Umsetzung von IoT-Projekten. Durch die einheitliche, jedoch kundenspezifisch anpassbare Codebasis, auf der die Software beruht, können verschiedene Geschäftsmodelle abgebildet werden. Von Seiten Ciscos wird die starke Stellung als Netzausrüster genutzt und bietet dadurch Network Connectivity, Fog Computing, Data Analytics, Security, Management and Automation und Application Platform. Mit Fog Computing ist ein Verfahren gemeint, welches von Cisco entwickelt wurde und darauf abzielt, rechenintensive Transaktionen direkt im Netz vorzunehmen, wodurch dringende Verarbeitungen von Sensordaten zeitnah bearbeitet werden können. Dadurch laufen Operationen nicht nur schneller und effizienter als über entfernte Clouds, sondern entlasten auch die Übertragungswege und sparen Speicherplatz in der Cloud ein [25].

Laut der Internetseite von Cisco kann per **REST** (Representational State Transfer) auf die CMX Mobility Services zugegriffen werden. [30]

### 3.2.3 Amazon

Durch seine Cloud-Infrastruktur hat **Amazon Web Services** (AWS) perfekte Voraussetzungen für den IoT-Markt. Einer der Dienste ist z.B. Amazon Machine Learning, welcher für die Analyse von IoT-Sensordaten geeignet ist. Beim Machine Learning können anhand bestimmter Algorithmen Muster in diesen Daten erkannt, Prognosemodelle entwickelt und falls nötig vorsorgliche Veränderungen vorgenommen werden. Zusätzlich zu den eigenen Diensten hat Amazon das Startup 2lementry übernommen, welches über eine Plattform verfügt, mit der Daten von Maschinen und Geräten getrackt und verwaltet werden können. Diese umfasst auch Möglichkeiten zum Einsatz der sogenannten Beacon-Technologie, bei der unter anderem über Übertragungstechniken wie Bluetooth LE per Push-Nachrichten ortsgebundene Sonderangebote an ein Smartphone gesendet werden können. [25]

Speziell für den Bereich des Internets der Dinge ist AWS IoT seit Mitte Dezember 2015 verfügbar. Über diese Cloud-Plattform können die verbundenen Geräte untereinander oder mit speziellen Cloud-Anwendungen kommunizieren. Solche Anwendungen sind unter anderem AWS Lambda, Amazon Kinesis, Amazon S3, das bereits erwähnte Amazon Machine Learning und Amazon DynamoDB. Diese können zum Aufbau von IoT-Anwendungen eingesetzt werden, um die Daten der verbundenen Geräte zu sammeln, zu verarbeiten und zu analysieren [25]. AWS IoT unterstützt zum Nachrichtenaustausch die Anwendungsschichtprotokolle **MQTT**, **HTTP/1.1** [31] und **HTTP/2** [32].

### 3.2.4 IBM

Im März 2015 hatte IBM mitgeteilt, über die nächsten vier Jahre rund drei Milliarden Dollar in den Aufbau einer IoT-Division zu investieren, welche innerhalb des Unternehmensbereichs IBM Analytics angesiedelt sein wird. Hier sollen neue Produkte und Services speziell für IoT entwickelt werden. Gleichzeitig wurde auch die "**IBM IoT Cloud Open Platform for Industries**" angekündigt. Über diese soll es möglich sein, eine Vielzahl von Daten aus unterschiedlichen Quellen zu sammeln und auszuwerten. Neben dieser Plattform bietet IBM auch eine weitere IoT-Lösung innerhalb der PaaS-Plattform Bluemix: die **Watson IoT Platform**. Hier können Entwickler Lösungen erstellen, die mit den Daten aus den IoT-Quellen agieren und diese auswerten lassen. Darüber hinaus können Apps entwickelt werden, die wiederum mit den Resultaten dieser Lösungen arbeiten. IBM versucht damit auch zu ermöglichen, dass vorhandene Geschäftsanwendungen mit diesen Echtzeitdaten erweitert werden, um IoT-Prozesse automatisieren und verbessern zu können. Dabei sollen auch vernetzte Geräte über Schnittstellen zwischen Cognitive Computing und IoT um kognitive Fähigkeiten erweitert und damit intelligent gemacht werden, um neue Marktchancen zu erschließen. [25]

Die Daten der Sensoren können über das offene Lightweight **MQTT**-Nachrichtenprotokoll oder per **HTTP** zur Cloud übertragen werden. Auf die in der Plattform verfügbaren Sensoren und Daten kann dann wiederum über diverse APIs zugegriffen werden (z.B. über **REST**). [33]

### 3.2.5 Intel

Intel ist mit seinen Ein-Prozessor-Computern „Galileo“ und „Edison“ im Bereich der Endgeräte für IoT schon sehr gut aufgestellt, möchte aber neben diesen Endgeräten auch im Bereich der zu verwendenden Software tätig werden. Dabei entwickelt es das **Intel IoT Gateway**, welches eine Art Roadmap für intelligente Hard- und Software-Lösungen bietet. Unter anderem sind ein API-Management, Software-Services, Data Analytics, Cloud-Konnektivität, intelligente Gateways und Funktionen zur IT-Sicherheit enthalten. Der Fokus beim Intel IoT Gateway liegt vor allem darauf, IoT-Technologien weitestgehend ohne Programmierkenntnisse auf den darunterliegenden Schichten entwickeln zu können. Die Security-Tochter McAfee stellte zusätzlich das **Enhanced Security for Intel IoT Gateways** zur Verfügung. Intel arbeitet auch mit Siemens zusammen, um Firmen anzusprechen, die ihre Geräte erstmals an das Internet anschließen.

Für die Kommunikation nutzt das Intel IoT Gateway **MQTT** und **MODBUS** [34] sowie weitere gängige Protokolle wie **REST**, **DSS**, **CoAP**, **XMPP** oder **HTTP** [35]. Dadurch wird das angestrebte Ziel, möglichst viele Geräte problemlos miteinander zu verknüpfen, stark unterstützt.

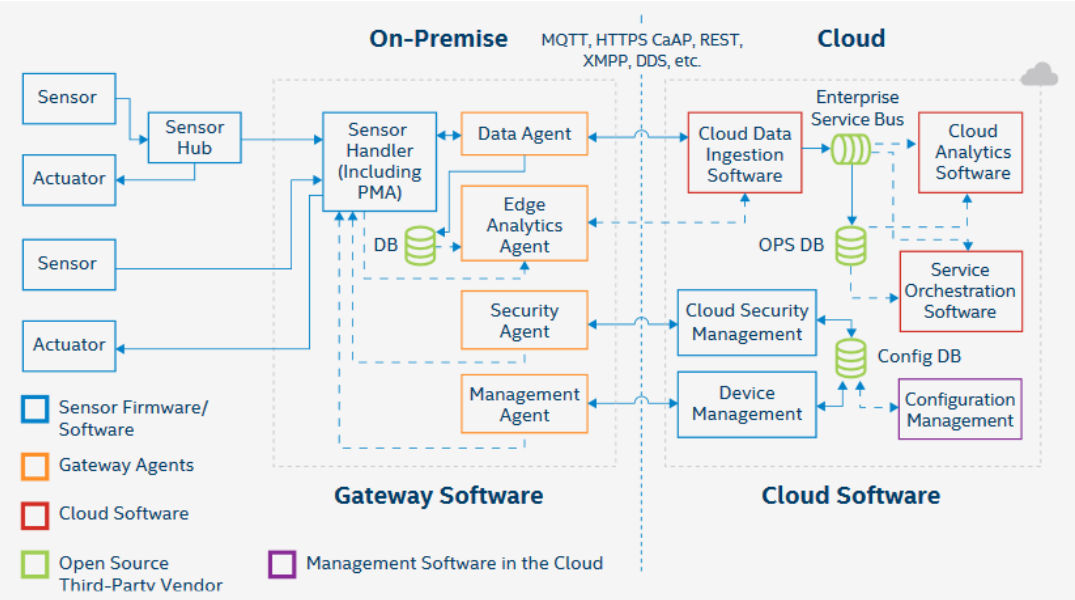


Abbildung 3-13: Software-Komponenten und Schnittstellen der Intel Reference Architecture [35]

### 3.2.6 SAP

SAP hat ihre Cloud Platform HANA für das Verbinden und Managen von Geräten sowie Datenintegration und -analyse im Bereich Internet der Dinge erweitert. Darin enthalten sind auch SAP Predictive Maintenance and Service, SAP Connected Logistics und Connected Manufacturing. Diese sollen auch dabei helfen, Maschinen und Anlagen über größere Entfernungen hinweg zu überwachen und drohende Ausfälle zu erkennen und zu vermeiden. Durch die Verbindung der einzelnen Herstellungsabschnitte bis hin zur Auslieferung soll das Unternehmen dabei unterstützt werden, den kompletten Produktionsprozess zu überwachen, zu analysieren und effizienter gestalten zu können. Dies geschieht durch die Verwendung bisheriger Daten- und Anwendungsservices aus HANA. [25]

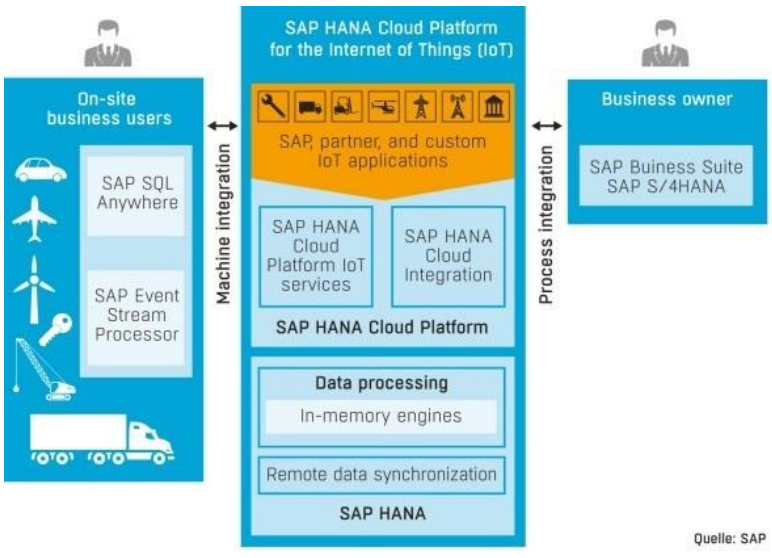


Abbildung 3-14: SAP HANA Cloud Platform [25]

SAP arbeitet im Bereich IoT auch mit anderen Konzernen zusammen, um gemeinsame IoT-Lösungen zu entwickeln. Darunter Siemens, die HANA als Grundlage ihrer Lösung gewählt haben, Intel, um die Integration des Intel IoT Gateways in HANA zu ermöglichen und auch T-Systems. [25]

SAPs Message Management Service (MMS) API unterstützt derzeit **HTTP**, **WebSocket** und **MQTT** über Websocket. Dadurch können Nachrichten an die Plattform gesendet oder abgerufen werden. Über den SAP MMS ist es auch über HTTP möglich, Push-Nachrichten zu versenden, wobei diese zwischengespeichert werden und aktiv vom Gerät angefragt werden müssen. [36]

### 3.2.7 HP (Enterprise)

Im Februar 2015 wurde die **HP Internet of Things Platform** vorgestellt, welche sich an Unternehmen im Bereich Communication Services richtet. Diese sollen damit in ihren Netzen große Mengen an vernetzten Produkten und Endgeräten verwalten und analysieren können. Dabei handelt es sich um einen indirekten Marktzugang, da die Service Provider mit dem IoT-Stack von HP neue Geschäftsmodelle entwickeln können. Mit dem HP Energy Management Pack hat HP die erste branchenspezifische Anwendung für die HP IoT Platform eingeführt. Damit können Telekommunikations- und Versorgungsunternehmen sowohl Verbrauchern als auch der Industrie eine Hausautomatisierung und Energiesteuerung ermöglichen. [25]

HP verwendet zur Kommunikation einen Standard der oneM2M, einer globalen Initiative für Machine to Machine (M2M) Communication [37]. Dieser Standard unterstützt unter anderem **HTTP** und **MQTT** [38].

### 3.2.8 Google

Im Bereich IoT konzentriert sich Google derzeit auf Smart Home. Deutlich macht dies auch die Übernahme des Smart Home Spezialisten Nest. Selbst entwickelt hat Google das Andriod-Derivat **Brillo**, ein Betriebssystem für das Internet der Dinge (wie auch das Microsoft Windows 10 IoT Core), welches als abgespeckte Android-Variante möglichst viele Prozessoren und Connectivity-Standards unterstützen soll. Zudem auch das Datenbanksystem BigQuery und die Applikationsplattform Firebase. Besonders zu erwähnen ist auch der Echtzeit-Messaging- und Streaming-Dienst **Cloud Pub/Sub** [25]. Der Aufbau des Anwendungsschicht-Protokolls dieser Middleware ist ähnlich aufgebaut wie MQTT, verfügt aber über erweiterte Funktionen. Zusätzlich können Nachrichten über eine **REST** API ausgetauscht werden [39].

### 3.2.9 Bosch Software Innovations

Neben anderen Industriekonzernen beschäftigt sich Bosch schon längere Zeit intensiv mit dem Thema Internet der Dinge und hat hier auch schon viel Geld investiert. In diesem Zusammenhang hat Bosch Innovations Software Technologies, Inubit und die Prosyst Software GmbH übernommen und zusammen mit Cisco und ABB das Joint Venture Mozaiq Operations GmbH gegründet, welches eine Open-Software-Plattform für den Bereich Smart Home entwickeln soll. Bosch hat hier, als Global Player im Bereich Sensortechnik, die Entwicklung rechtzeitig erkannt und sichert sich mit ihrer **Bosch IoT Suite** einen wichtigen Platz im internationalen Wettstreit um den Zukunftsmarkt IoT. Über die Zusammenarbeit mit PTC (siehe 3.2.11) wird deren IoT-Entwicklungsplattform ThingWorx über den M2M-Konnektor der Bosch IoT Suite an diese angebunden. [25]

Nativ verfügt die Bosch IoT Suite über Bosch IoT Things eine **REST** API [40].

### 3.2.10 Siemens

Auch Siemens beschäftigt sich im Rahmen seiner Digitalisierung längst mit dem Thema IoT und Industrie 4.0. Als einer der größten Hersteller im Bereich der Automatisierung legt Siemens jedoch andere Schwerpunkte als z.B. Bosch. Siemens nutzt hierfür seine Kompetenz im Bereich Product-Lifecycle-Management und stellt ihren Kunden mit der **Digital Enterprise Software Suite** eine Lösung zur Optimierung der industriellen Wertschöpfungskette zur Verfügung. Wie unter 3.2.6 angemerkt, nutzt Siemens zudem die Basis der SAP HANA Cloud Platform, um eine eigene offene Cloud-Plattform für Big-Data-Analysen anzubieten. [25]

Durch die Verwendung der SAP HANA Cloud Platform in der Basis unterstützt Siemens‘ Cloud-Plattform ebenso **HTTP**, **WebSocket** und **MQTT**. [36]

### 3.2.11 PTC

Nach der Übernahme von **ThingWorx** bietet PTC eine Plattform für die Entwicklung und Inbetriebnahme von IoT-Anwendungen in Unternehmen an. ThingWorx umfasst Konnektivität, Geräte-Clouds, Geschäftslogik, Big Data, Analysen und Remote-Service-Anwendungen. In Zukunft möchte das Unternehmen vor allem Lösungen für Application- und Services-Lifecycle-Management (ALM, SLM) sowie Aftermarket-Services anbieten. Dabei liegt der Fokus also auf Dienstleistungen, die rund um ein verkauftes und in Betrieb genommenes Produkt entstehen. [25]

Neben dem eigenen Nachtrichten-Protokoll ThingWorx Communication Protocol besteht eine **REST** API für die Kommunikation mit der Plattform. [41]

### 3.2.12 Deutsche Telekom

Die Deutsche Telekom ist Mitglied der Global M2M Association (GMA) [42], zu der auch andere Mobilfunkhersteller wie Orange, Telecom Italia, TeliaSonera, Bell Canada, SoftBankMobile und Swisscom gehören. Die GMA will weltweit erweiterte und nahtlose M2M-Services bereitstellen. Das daraus entstandene Produkt ist die **Multi-Domestic Service-Plattform**, über die Unternehmen eine zentrale Lösung für das Echtzeit-Management der Konnektivität erhalten, mit der sie ihre vernetzten, weltweit betriebenen Geräte verwalten, überwachen, unterstützen und Fehler beheben können. Dabei kommen eingebettete SIM-Karten (eSIMs) zum Einsatz, mit denen dem Endnutzer durch ein lokalisiertes Profil, je nach Aufenthaltsort, lokale Angebote bereitgestellt werden können. [25]

### 3.2.13 Oracle

Oracle hat seine IoT-Plattform ganz nach dem Prinzip erstellt, dass der höchste Nutzen dann erreicht werden kann, wenn eine effektive Kommunikation zwischen allen Elementen gegeben ist. Aus diesem Grund bietet Oracle mit seiner IoT-Plattform ein Komplettpaket an, bei dem es alles von der Hardware über eine Middleware bis hin zur Datenbank und den Analysetechnologien mitliefert. Zusätzlich bietet die Firma einen Service zur Einrichtung und Unterstützung bei der Verwendung in allen Bereichen des Systems an. [25]

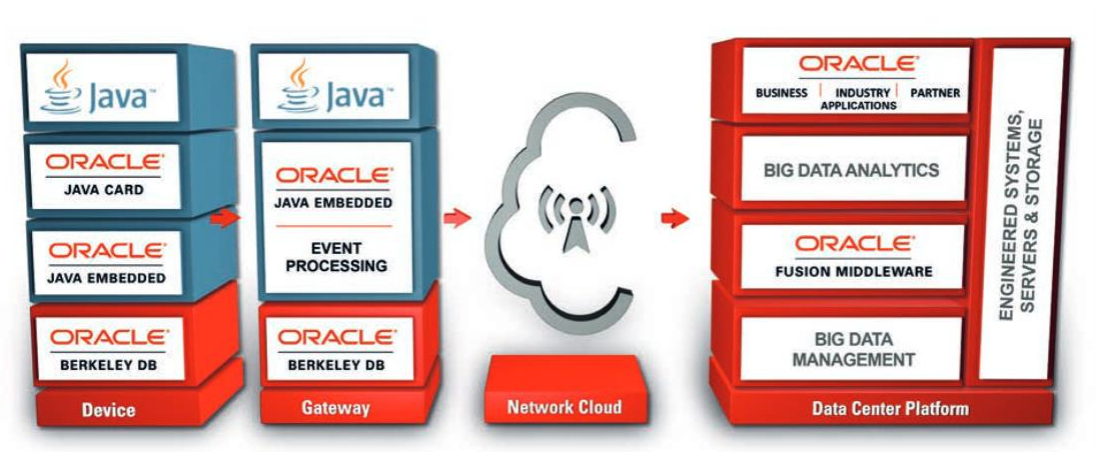


Abbildung 3-15: Oracle IoT-Plattform [43]

Der Oracle Data Center erlaubt als Webserver den Zugriff über **HTTP** und stellt zusätzlich über die Oracle REST Data Services eine Alternative über **REST** zur Verfügung [44]. Der Oracle IoT Cloud Service kann zusätzlich über die bereitgestellte MQTT-Bridge für die Verwendung von Geräten und Gateways mit **MQTT** erweitert werden [45].

### 3.2.14 Blackberry

Die BlackBerry IoT Plattform umfasst Gerätesoftware, Cloud-Services und Schnittstellen zur bestehenden IT. Mit den BlackBerry Cloud-Services werden Messaging-Dienste sowie Funktionen für Business-Logik und Anwendungsentwicklung zur Verfügung gestellt. Die Einsatzmöglichkeiten der IoT-Anwendungen liegen vor allem in den Bereichen Datenanalyse, Life-Cycle-Management, Gerätemanagement und Identitätsmanagement. Zudem bestehen weitere, branchenspezifische Anwendungen z.B. im Bereich Automotive. [25]

Die in der Plattform verwendeten Nachrichten und Notifications verwenden **REST** in Verbindung mit JSON-formatierten Objekten. [46]

### 3.2.15 Fazit

Ähnlich wie bei den Übertragungstechniken bestehen auch hier viele Kooperationen verschiedener Firmen, die zusammen an einer gemeinsamen Lösung arbeiten oder in der Basis die Technik eines anderen verwenden. Klar zu erkennen ist jedoch, dass mittlerweile alle erkannt haben, dass im Bereich IoT nicht nur ein mögliches neues Arbeitsfeld liegen könnte, sondern – gerade in der Industrie – ein verspätetes Befassen mit der Thematik starke Auswirkungen auf das Geschäft haben könnte. Auch zu erkennen ist, dass eine bidirektionale Kommunikation mit einem Server nicht für alle angebundenen und vernetzten Geräte von Nöten ist. Gateways jeglicher Art kommen fast überall zum Einsatz und vermitteln nicht nur zwischen Endgeräten und Servern sondern auch zwischen verschiedenen Protokollen. Deshalb wird auch im praktischen Beispiel eine Art Gateway zu Veranschaulichung vorkommen.

## 3.3 Verwendung von Anwendungsschicht-Protokollen

Die von den proprietären Lösungen verwendeten Protokolle (erster Abschnitt) und weitere Protokolle (zweiter Abschnitt) sind hier aufgelistet. Im weiteren Verlauf werden die fettgedruckten Protokolle näher untersucht. Die Auswahl beruht auf der Häufigkeit des Vorkommens bei der einführenden Recherche.

- **MQTT**
- **HTTP/1.1**
- **HTTP/2**
- **REST**
- **CoAP**
- AMQP

- SMQTT
- MQTT-SN
- SSI
- STOMP

- **XMPP**
- MODBUS
- **DDS**
- Google Cloud Pub/Sub
- WebSocket

- SOAP
- OFC
- OPC-UA

# 4 Vergleich von IoT Anwendungsschicht-Protokollen

In diesem Kapitel werden die gängigsten IoT Anwendungsschicht-Protokolle, die auch bei der bisherigen Recherche vorgekommen sind, untersucht und verglichen. Dazu werden zunächst die Untersuchungskriterien festgelegt und beschrieben. Im Anschluss erfolgt die Untersuchung mit einer zusammenfassenden Übersichtstabelle nach jeder Protokolluntersuchung, welche am Ende zu einer Gesamtübersicht zusammengefügt wird. Zum Schluss erfolgt ein Fazit zur Untersuchung.

## 4.1 Kriterien zur Untersuchung der Protokolle

Hier werden zunächst die Untersuchungskriterien festgelegt und beschrieben. Dies ist wichtig, damit in der Gesamtübersicht direkt zwischen den Protokollen verglichen werden kann.

### 4.1.1 Beschreibung

In der Beschreibung wird das Protokoll kurz vorgestellt. Dabei sollte auf die Herkunft und Entstehung des Protokolls genauso eingegangen werden wie auf die Bereiche, in denen das Protokoll verwendet wird. Am Ende jeder Beschreibung solle eine stichwortartige Kurzbeschreibung für die Übersichtstabelle stehen.

### 4.1.2 Kommunikation

Da die Kernaufgabe der Anwendungsschicht-Protokolle die Kommunikation zwischen verschiedenen Geräten ist, muss hier etwas mehr ins Detail gegangen werden.

#### 4.1.2.1 Kommunikationsarchitektur

Einer der wichtigsten Punkte, von dem etliche andere abhängen, ist die Architektur. Dabei kann sowohl zwischen zentralisiert und dezentralisiert unterschieden werden, als auch zwischen der Request/Response- und Publish/Subscribe-Architektur [47]. Wichtig ist, hier die Kommunikationsarchitektur nicht mit der Netzwerkarchitektur zu verwechseln. Die Kommunikationsarchitektur des untersuchten Protokolls wird hier kurz beschrieben.

#### 4.1.2.2 Kommunikationsform

In einem Netzwerk im Internet der Dinge ist es auch wichtig, wie die Nachrichten ausgetauscht werden. Dabei unterscheidet man z.B. zwischen synchron, asynchron und pipelining. Hier werden die möglichen Formen der Kommunikation beschrieben.

#### 4.1.2.3 Kommunikationswege und Adressierung

Bei der Kommunikation ist es wichtig, dass die Nachrichten auch an die richtige Stelle gesendet werden. Hier kann eine Nachricht an einen einzelnen Empfänger gesendet werden oder, wenn es das Protokoll zulässt, auch an mehrere (unicast, multicast, broadcast). Dabei ist nicht nur zu beschreiben, inwieweit die einzelnen Geräte identifiziert werden, sondern auch, wie die Adressierung einer Nachricht an ein oder mehrere andere Geräte funktioniert. Die Kommunikationswege und die Art der Adressierung werden hier beschrieben.

#### 4.1.2.4 Kommunikationsrichtung

Je nach Art der Anwendung kann es sich um eine unidirektionale oder eine bidirektionale Kommunikation handeln. Deshalb wird hier kurz aufgelistet, welche Kommunikationsrichtungen möglich sind.

### 4.1.3 Funktionsumfang

Unter diesem Punkt wird der Funktionsumfang des untersuchten Protokolls in verschiedenen Bereichen beschreiben.

#### 4.1.3.1 Nachrichtenarten

Bei gesendeten Nachrichten handelt es sich nicht immer nur über direkten Datenaustausch. Teils muss zunächst eine Art Verbindung eingegangen werden oder die Nachrichtenart gibt an, ob eine Nachricht erfragt oder gesendet wird. Die verschiedenen Nachrichtenarten werden hier kurz beschreiben.

#### 4.1.3.2 Nachrichtenformate

Die Nachrichtenformate variieren bei den Protokollen von einfachen Textnachrichten über binäre Daten bis hin zu Objekten. Teils kann auch die Komplexität der Nachrichten durch standardisierte Formatierungen der Daten vereinfacht dargestellt werden. Ob dies mit dem untersuchten Protokoll gängige Praxis ist oder gar nicht möglich, wird hier beschrieben.

Da die Formatierung im JSON-Format (JavaScrip Object Notation) zum Senden von komplexen Strukturen bei jedem Protokoll verwendet werden kann, das die Übertragung von reinem Text erlaubt, wird sie hier kurz vorgestellt. Dargestellt werden können Variablen vom Typ NULL, Boolean, Zahl, Zeichenkette, Array oder einem Objekt. [48]

```
{                                       Objektanfang
  "Name": "Johannes",                   String-Wert
  "Kinder": [                           Arrayanfang
    { "Name": "Tom",  "Alter": 19,},    zwei Objekte mit String-
    { "Name": "Lisa", "Alter": 23,}     und Zahlenwert
  ]                                     Arrayende
}                                       Objektende
```

Quellcode 4-1: JSON Beispiel

#### 4.1.3.3 Quality of Service

Der Punkt QoS beschäftigt sich unter andrem mit der Zuverlässigkeit, mit der die Nachrichten übertragen werden können. Die unterschiedlichen Stufen werden hier, falls vorhanden, in ihrer protokollspezifischen Verwendung kurz beschrieben.

#### 4.1.3.4 Sicherheit

Hier wird angegeben, ob das Protokoll Sicherheitsmaßnahmen bei der Verbindungsaufnahme oder dem Senden der Nachrichten unterstützt.

#### 4.1.3.5 Sonstiges

Hier ist Platz um Besonderheiten im Funktionsumfang außerhalb der genannten Kategorien zu beschreiben.

### 4.1.4 Verwendungsmöglichkeiten

Die Protokolle sind teils an gewisse Dinge wie spezielle Hard- und Software gebunden. Um schnell zu erkennen, wofür ein Protokoll eingesetzt werden kann, wird hier die Hard- und Software, die das Protokoll unterstützt, aufgezeigt.

#### 4.1.4.1 Hardware

Mittlerweile stellen die verschiedensten Hersteller nicht nur Online-Dienste, sondern auch Hardware in Form von kleinen Chips bis hin zu komplexen Boards her. Die wichtigste Hardware, die das untersuchte Protokoll unterstützt, wird hier aufgelistet.

#### 4.1.4.2 Programmiersprachen

Anwendungen im IoT können mit diversen Programmiersprachen erstellt werden. Diejenigen Programmiersprachen, die das untersuchte Protokoll unterstützen bzw. für die es eine Bibliothek gibt, werden hier aufgelistet.

#### 4.1.4.3 Plattform-Dienste

Für die zentrale Speicherung und Verarbeitung bestehen diverse Plattformdienste (PaaS) für IoT. Diejenigen Plattformdienste, die das untersuchte Protokoll direkt (evtl. mit Zusatzinstallation) unterstützen, werden hier aufgelistet.

### 4.1.5 Aufbau des Protokolls und Overhead

Gerade im Bereich der Sensoren ist es wichtig, möglichst nur die Daten zu übertragen, die tatsächlich notwendig sind. Aus diesem Grund wird unter diesem Punkt gezeigt, wie das Protokoll aufgebaut ist und vor allem, wie hoch der Overhead, also der Anteil der Nicht-Nutzdaten, ist.

Die Beschreibung erfolgt in Textform und ergänzend für die Übersichtstabelle mit einer Einordnung in **hoch, mittel** oder **gering** in Bezug auf den Overhead.

### 4.1.6 Energiesparendes Protokolldesign

Das Energiesparverhalten des Protokolls ist zum einen von der verwendeten Hardware und zum anderen vom Overhead abhängig. Unter diesem Punkt wird kurz die energiesparendste Variante vorgestellt.

### 4.1.7 Verbreitung/Support/Foren

Ein gutes Protokoll ist nicht automatisch die beste Wahl. Für den Einsatz ist die Verbreitung von großer Bedeutung. Eine erste Einschätzung kann anhand Verwendungsmöglichkeiten vorgenommen werden. Jedoch ist es gerade für Entwickler wichtig, dass ein Support in Form von genügend Foren, die sich mit dem Protokoll beschäftigen, besteht. Zur Einschätzung werden die Erkenntnisse der Verwendungsmöglichkeiten sowie die Anzahl der Google- und StackOverflow-Suchergebnisse herangezogen.

Die Beschreibung erfolgt in Textform und ergänzend für die Übersichtstabelle mit einer Einordnung in **hoch, mittel** oder **gering** in Bezug auf die Verbreitung/Support/Foren.

### 4.1.8 Zukunftsfähigkeit

Manche Protokolle sind recht neu, andere sind bereits schon etliche Jahre alt und kommen teils aus einer Zeit, in der sie lediglich für die Kommunikation zwischen Maschinen eingesetzt wurden. Dies bedeutet jedoch nicht gleich, dass die neueren Protokolle sich durchsetzen werden. Manche alten Protokolle haben sich bereits bewährt und sind vielfältig eingesetzt. Deshalb soll hier kurz abgewogen werden, wie zukunftsfähig das untersuchte Protokoll ist.

Die Beschreibung erfolgt in Textform und ergänzend für die Übersichtstabelle mit einer Einordnung in **hoch, mittel** oder **gering** in Bezug auf die Zukunftsfähigkeit.

### 4.1.9 Vor- und Nachteile des Protokolls bezogen auf sein Einsatzgebiet

Jedes einzelne Protokoll hat seine Vor- und Nachteile. Oft ist dies abhängig vom Einsatzgebiet. Signifikante Merkmale und Eigenschaften können unter diesem Punkt jedoch aufgeführt werden, was den Vergleich der Protokolle untereinander nochmal etwas vereinfacht.

### 4.1.10 Zusammenfassung und Fazit

Nicht alles lässt sich einer Kategorie zuordnen oder verhält sich immer gleich. Deshalb wird hier kurz zusammengefasst, was das Protokoll ausmacht und wo der Einsatz Sinn macht.

### 4.1.11 Übersichtstabelle

| **Beschreibung** | |
|---|---|
| **Kommunikation** | |
| **Kommunikationsarchitektur** | |
| **Kommunikationsform** | |
| **Kommunikationswege und Adressierung** | |
| **Kommunikationsrichtung** | |
| **Funktionsumfang** | |
| **Nachrichtenarten** | |
| **Nachrichtenformate** | |
| **Quality of Service** | |
| **Sicherheit** | |
| **Sonstiges** | |
| **Verwendungsmöglichkeiten** | |
| **Hardware** | |
| **Programmiersprachen** | |
| **Plattform-Dienste** | |
| **Aufbau des Protokolls und Overhead** | hoch/mittel/gering |
| **Energiesparendes Protokolldesign** | hoch/mittel/gering |
| **Verbreitung/Support/Foren** | hoch/mittel/gering |
| **Zukunftsfähigkeit** | hoch/mittel/gering |
| **Zentraler Verwaltungsserver/Broker** | |
| **Vor- und Nachteile des Protokolls** | |

Tabelle 4-1: Übersichtstabelle Vorlage

## 4.2 Untersuchung der Protokolle

### 4.2.1 MQTT

#### 4.2.1.1 Beschreibung

Das MQTT (Message Queue Telemetry Transport)-Protokoll wurde ursprünglich bereits 1999 von IBM für die M2M (Machine to Machine)-Kommunikation entwickelt. Seit der Version 3.1 (2010) ist das Protokoll offen verfügbar. Bei MQTT handelt es sich um ein leichtgewichtiges Protokoll, das mit einem minimalen Overhead auskommt und somit eine einfach zu implementierende Kommunikation ermöglicht [49]. Dies macht das Protokoll besonders für eingeschränkte Netzwerke geeignet, bei denen die Bandbreite niedrig ist oder die Kosten für das Netzwerk hoch sind. Zudem benötigt das Protokoll nur wenige CPU-Ressourcen und eignet sich deshalb sehr gut für eingebettete Systeme [50].

**Kurzbeschreibung: leichtgewichtiges M2M-Protokoll, offener Standard, geringe CPU-Ressourcen**

#### 4.2.1.2 Kommunikation

##### 4.2.1.2.1 Kommunikationsarchitektur

MQTT basiert auf dem **Publish/Subscribe**-Modell, bei dem die Nachrichten von einem zentralen Nachrichtenvermittler (Broker) empfangen und weitergeleitet werden. Anders als beim Request/Response-Modell findet die Kommunikation nicht direkt zwischen dem Nachrichtensender und -empfänger statt. Die Clients sind komplett voneinander entkoppelt und müssen nichts voneinander wissen. Das Senden und Empfangen erfolgt ausschließlich über den Broker. [51]

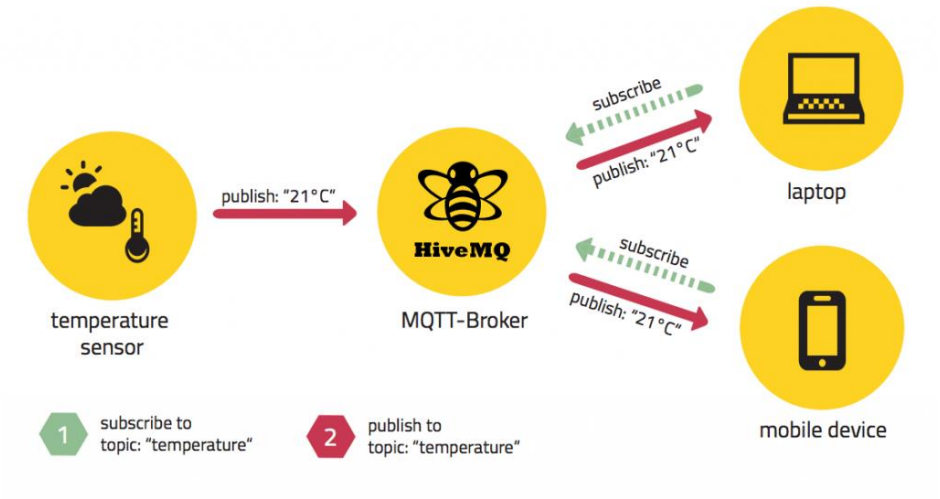


Abbildung 4-1: MQTT Kommunikationsarchitektur am Beispiel des HiveMQ MQTT-Brokers [51]

#### 4.2.1.2.2 Kommunikationsform

Da es sich bei MQTT nicht um ein Request/Respone-Modell handelt, ist die Kommunikationsform nicht einfach festzulegen. Aus der Sicht eines Subscribers findet die Kommunikation zwischen ihm und dem Broker „asynchron“ statt. Der Subscriber abonniert ein Topic und erhält eine Nachricht/Antwort, sobald etwas an diesen Topic publiziert wurde. Der Publisher hingegen kommuniziert grundsätzlich weder synchron noch asynchron. Er sendet lediglich eine Nachricht an den Broker. Berücksichtigt man jedoch die Möglichkeiten des QoS-Levels (Quality of Service), bei dem Nachrichten auch vom Empfänger quittiert werden, so kommt eine **synchrone** Kommunikationskomponente zwischen Sender und Empfänger (Publisher und Broker/Broker und Subscriber) hinzu. [52]

#### 4.2.1.2.3 Kommunikationswege und Adressierung

Bei MQTT erfolgt die Adressierung der Nachrichten über sogenannte Topics. Ein Publisher sendet die Nachricht nur direkt an den Broker, der diese an alle Subscriber weiterleitet. Somit besteht das Senden einer Nachricht also aus einer oder mehreren **Unicast**-Nachrichten. Betrachtet man das Pub/Sub-Modell aber im Gesamten, ist auch **Multicast möglich**, da die Nachrichten an alle Clients gesendet werden, die dieses Topic abonniert haben. Per Broadcast an alle mit dem Broker verbundenen Clients zu senden, ist nicht möglich. Das Empfangen mehrerer oder aller Nachrichten ist durch sogenannte Wildcards möglich. Die Topics sind wie folgt durch verschiedene Level (hierarchischer Namensraum) aufgebaut und sind case-sensitive. [53]

Topic mit zwei übergeordneten Level: `meinZuhause/Wohnzimmer/temp`

Bei den Wildcards wird zwischen Single Level und Multi Level unterschieden.

Single Level Wildcard: `meinZuhause/+/temp`
Multi Level Wildcard: `meinZuhause/#`

Die Adressierung auf der Transportschicht übernimmt der Broker. Die Clients müssen lediglich die IP-Adresse oder den Hostnamen des Brokers beim Aufbau der Verbindung kennen.

#### 4.2.1.2.4 Kommunikationsrichtung

Da die Publisher und Subscriber nichts von der gegenseitigen Existenz wissen und die Kommunikation ausschließlich über den Broker stattfindet, handelt es sich um eine **unidirektionale** Kommunikation vom Publisher (über den Broker) zum Subscriber. Zwischen Client und Broker besteht jedoch eine bidirektionale Verbindung, wodurch der Empfang einer Nachricht bestätigt werden kann. [51]

#### 4.2.1.3 Funktionsumfang

##### 4.2.1.3.1 Nachrichtenarten

Bei MQTT kommen 14 verschiedene Arten von Nachrichten zum Einsatz, mit denen die komplette Kommunikation gesteuert wird. Im Folgenden werden alle kurz vorgestellt.

**CONNECT**

Jeder neue Client (Publisher/Subscriber) meldet sich damit beim Broker an. Dadurch wird die Verbindung hergestellt und eine Session gestartet. Sofern die Verbindung eine Authentifizierung erfordert, werden die Daten ebenfalls in dieser Nachricht übermittelt. [54]

**CONNACK**

Um die Verbindung zu bestätigen sendet der Broker ein CONNACK an den Client. Wenn der Client nach einer CONNECT-Nachricht nach einer gewissen Zeit noch keine Bestätigung erhalten hat, wird der Verbindungsaufbau neugestartet. [54]

**PUBLISH**

Diese Nachricht wird vom Publisher an den Broker gesendet, der sie dann an die Subscriber des entsprechenden Topics weitersendet. Der Topic muss explizit angegeben werden, die Verwendung von Wildcards ist nicht möglich. Abhängig von dem verwendeten QoS-Level (beschrieben unter 4.2.1.3.3) folgen weitere Nachrichten. [54]

**PUBACK**

Diese Nachricht bestätigt den Erhalt einer Publish-Nachricht (QoS Level 1) und wird entweder vom Broker an den Publisher gesendet oder vom Subscriber an den Broker. [54]

**PUBREC**

Als Antwort auf eine Publish-Nachricht mit QoS Level 2 wird eine PUBREC an den Publisher zurück geschickt (oder vom Subscriber an den Server), um den Erhalt zu bestätigen. Die Publish-Nachricht wird zu diesem Zeitpunkt noch nicht weitergeleitet. [54]

**PUBREL**

Der Publisher erteilt dem Broker mit dieser Nachricht die Freigabe der anfangs gesendeten Publish-Nachricht, welche dann an die Subscriber weitergeleitet und mit einer PUBCOMP-Nachricht an den Publisher abgeschlossen wird. Im Falle einer Kommunikation zwischen Broker und Subscriber wird die Publish-

Nachricht nach einem PUBREL an die verarbeitende Anwendung weitergegeben. [54]

**PUBCOMP**

Wurde eine PUBREL-Nachricht vom Broker an die Subscriber weitergeleitet, wird dies durch eine PUBCOMP-Nachricht an den Publisher bestätigt. Entsprechend auch vom Subscriber gegenüber dem Broker. [54]

**SUBSCRIBE**

Mit dieser Nachricht registriert sich ein Client für ein oder mehrere Topics. Hierbei kann können auch die vorgestellten Wildcards verwendet werden. Die Übertragung der Publish-Nachrichten vom Broker an den Subscriber erfolgt aufgrund des beim Abonnieren angegebenen QoS-Levels. Die SUBSCRIBE-Nachricht selbst wird durch ein SUBACK an den Subscriber bestätigt. [54]

**SUBACK**

Dies ist die Bestätigung eines Brokers gegenüber dem Subscriber für ein Abonnement eines angeforderten Topics. [54]

**UNSUBSCRIBE**

Jeder Subscriber kann sich auch wieder von einzelnen Topics ausschreiben. Dies erfolgt mit dieser Nachricht und wird mit einem UNSUBACK bestätigt. [54]

**UNSUBACK**

Dies ist die Bestätigung für ein erfolgreiches Abbestellen eines bestimmten Topics. [54]

**PINGREQ**

Mit dieser Nachricht prüft ein Client, ob der Broker noch verfügbar ist. [54]

**PINGRESP**

Der Broker bestätigt die Verfügbarkeitsanfrage mit dieser Nachricht. [54]

**DISCONNECT**

Ein Client kann damit seine Verbindung zum Broker beenden. Dies stellt eine sauberere Lösung dar, als einfach die TCP-Verbindung zu beenden und ermöglicht die „Clean Session“, bei der beim Verbindungsabbau alle Daten des Clients gelöscht werden. [54]

#### 4.2.1.3.2 Nachrichtenformate

MQTT überlässt es dem Entwickler, welches Format übertragen werden soll. Das Protokoll unterstützt sowohl die Übertragung von binären Daten als auch reinen Text und mit JSON oder XML strukturierte Texte. [55]

#### 4.2.1.3.3 Quality of Service

Quality of Service ist eine der Hauptfunktionalitäten von MQTT und unterscheidet zwischen drei verschiedenen Level bezüglich der garantierten Nachrichtenübertragung. Dabei bezieht sich die QoS immer auf den Teilabschnitt Publisher–Broker (abhängig vom QoS-Level in der Publish-Nachricht) oder Broker–Subscriber (abhängig vom QoS-Level, das der Client beim Subscriben angegeben hat). Der QoS-Level der kompletten Übertragung (Publisher–Subscriber) richtet sich also nach dem kleinsten Level auf den Teilabschnitten. [52]

**QoS-Level 0 (At most once)**

Der Publisher schickt seine Nachricht an den Broker, welcher diese an die interessierten Subscriber weiterleitet. Eine Bestätigung über den Empfang der Nachricht erfolgt an keiner Stelle. Die Nachricht wird vom Broker auch nicht zwischengespeichert, um sie aktuell nicht verfügbaren Subscribern später zu übermitteln. [54]

**QoS-Level 1 (At least once)**

Bei diesem Level wird der Erhalt der Nachricht gegenüber dem Sender bestätigt. Der Broker bestätigt den Empfang gegenüber dem Publisher und der Subscriber gegenüber dem Broker. Die Nachricht wird beim Broker zwischengespeichert bis der Empfang von allen Subscribern (mit QoS >0) bestätigt wurde. [54]

**QoS-Level 2 (Exactly once)**

Zusätzlich zum QoS-Level 1 wird hier sichergestellt, dass die Nachricht nur einmal beim Empfänger ankommt. Dies erfolgt in mehreren Schritten bei denen sowohl der Empfang als auch das Weiterleiten der Nachricht bestätigt wird. Das Weiterleiten erfolgt erst nach der Freigabe durch den Sender. [54]

#### 4.2.1.3.4 Sicherheit

MQTT erlaubt es, für die Verbindung mit einem Broker eine Authentifizierung durch einen Benutzername und Passwort festzulegen. Diese werden beim CONNECT-Aufruf übermittelt. Zudem bietet das darunterliegende TCP-Protokoll die Möglichkeit zur Verschlüsselung über TLS. [56]

#### 4.2.1.3.5 Sonstiges

Von MQTT besteht auch eine Version speziell für Sensornetzwerke: MQTT-SN. Hier sind die Nachrichten nochmal kürzer und speziell auf Sensornetzwerke, in denen auch „schlafende“ Clients enthalten sind, ausgelegt. Damit ist auch eine Kommunikation über nicht-TCP/IP-Netzwerke wie ZigBee einfacher umzusetzen. Über MQTT-SN Gateways sind diese dann mit dem MQTT-Broker verbunden. [57]

### 4.2.1.4 Verwendungsmöglichkeiten

#### 4.2.1.4.1 Hardware

Da die Hauptaufgabe vom Broker übernommen wird, kann MQTT auch auf MCUs mit sehr wenigen Ressourcen betrieben werden. Als Beispiel kann hier der ESP8266 genannt werden. Da die MQTT-Broker eine IP-basierten Adressierung verwenden, sollte die Hardware, die mit dem Broker verbunden ist, über einen WiFi-Chip verfügen.

#### 4.2.1.4.2 Programmiersprachen

MQTT wird von allen gängigen Programmiersprachen unterstützt. Eine detaillierte Auflistung kann im offiziellen MQTT-Wiki eingesehen werden. [58]

#### 4.2.1.4.3 Plattform-Dienste

Diese PaaS-Dienste unterstützen MQTT als Anwendungsschicht-Protokoll. [59]

- flowthings.io
- Carriots
- Beebotte
- IBM Watson IoT
- AWS IoT

### 4.2.1.5 Aufbau des Protokolls und Overhead

Das MQTT-Protokoll besteht aus einem festen und einem variablen Header. Der Overhead hat bei diesem leichtgewichtigen Protokoll nur 2 Byte (fester Header). Die Länge des variablen Headers ist abhängig von der Nachrichtenart [54]. Die durchschnittliche CONNECT-Nachricht hat einen Overhead von ungefähr 86 Byte, eine PUBLISH-Nachricht von ungefähr 15 Byte [60], was sehr **gering** ist.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| **byte 1** | Message type (3) | | | | DUP flag | QoS level | | RETAIN |
| | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| **byte 2** | Remaining Length | | | | | | | |

Tabelle 4-2: Fester MQTT-Header [54]

| | Beschreibung | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| **Topic Name** | | | | | | | | | |
| **byte 1** | Length MSB (0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **byte 2** | Length LSB (3) | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| **byte 3** | ‚a‘ (0x61) | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| **byte 4** | ‚/‘ (0x2F) | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| **byte 5** | ‚b‘ (0x62) | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| **Message Identifier** | | | | | | | | | |
| **byte 6** | Message ID MSB (0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **byte 7** | Message ID LSB (10) | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |

Tabelle 4-3: Variabler MQTT-Header einer PUBLISH-Nachricht an das Topic „a/b“ [54]

#### 4.2.1.6 Energiesparendes Protokolldesign

Aufgrund des geringen Overheads und der Möglichkeit zur Verwendung von sehr kleinen MCUs kann man bei MQTT von einem **hohen** energiesparenden Protokolldesign sprechen.

#### 4.2.1.7 Verbreitung/Support/Foren

Bei der vorangegangenen Recherche konnten viele Informationen zu diesem Protokoll gefunden werden. Eine Google-Suche „MQTT“ ergab 4.210.000 Ergebnisse und bei Stackoverflow.com 4.246 Treffer. MQTT wird zwar nicht von sehr vielen PaaS-Plattformen unterstützt, wird jedoch von anderweitigen Anwendungen vielseitig verwendet, weshalb man von einer **hohen** Verbreitung sprechen kann.

#### 4.2.1.8 Zukunftsfähigkeit

Obwohl MQTT schon älter ist, gewinnt es im Bereich IoT mehr und mehr an Bedeutung. Das Protokoll ist durch den geringen Overhead vor allem in Netzen mit geringer Bandbreite von Vorteil. MQTT wurde 2015 als OASIS [61] Standard und 2016 als ISO Standard akzeptiert. Weitere werden erwartet. Das macht MQTT zu einem der wichtigsten Anwendungsschichtprotokolle im IoT mit einer **hohen** Zukunftsfähigkeit. [62]

#### 4.2.1.9 Vor- und Nachteile des Protokolls

Die Vor- und Nachteile, die im Laufe der Recherche ersichtlich wurden, werden auch in anderen Berichten genannt [63] und sind hier aufgelistet.

**Vorteile:**

- Sehr geringer Overhead
- Skalierbar bis zu mehreren hunderttausenden Clients pro Server
- Viele Protokollfeatures, die speziell für IoT-Anwendungsfälle entwickelt wurden
- Für ressourcenbeschränke Geräte geeignet
- Clientimplementierungen sind für alle gängigen Programmiersprachen verfügbar

**Nachteile:**

- Reine Request/Response-Architekturen sind mit MQTT nur mit zusätzlichem Aufwand umsetzbar

#### 4.2.1.10 Zusammenfassung und Fazit

MQTT überzeugt vor allem durch den geringen Overhead sowie die große Unterstützung durch alle gängigen Programmiersprachen. Aktuelle und kommende Standardisierungen lassen vermuten, dass MQTT zukünftig noch mehr verwendet wird. Aus diesem Grund wird MQTT auch eines der Protokolle sein, die im praktischen Teil verwendet werden.

#### 4.2.1.11 Übersichtstabelle

| | |
|---|---|
| **Beschreibung** | Leichtgewichtiges M2M-Protokoll, offener Standard, geringe CPU-Ressourcen |
| **Kommunikation** | |
| **Kommunikationsarchitektur** | Publish/Subscribe |
| **Kommunikationsform** | Synchron |
| **Kommunikationswege und Adressierung** | Unicast/Multicast über Topics |
| **Kommunikationsrichtung** | Unidirektional |
| **Funktionsumfang** | |
| **Nachrichtenarten** | CONNECT, CONNACK, PUBACK, PUBREC, PUBREL, PUBCOMP, UBSCRIBE, SUBACK, UNSUBSCRIBE, UNSUBACK, PINGREQ, PINGRESP, DISCONNECT |
| **Nachrichtenformate** | Text und Binär |
| **Quality of Service** | QoS 0: At most once<br>QoS 1: At least once<br>QoS 2: Exactly once |
| **Sicherheit** | Authentifizierung durch Benutzername und Passwort, Verschlüsselung über TLS |
| **Sonstiges** | Eine MQTT-SN Version speziell für Sensor-netzwerke verfügbar |
| **Verwendungsmöglichkeiten** | |
| **Hardware** | Alle gängigen Computer und MCUs mit WiFI-Chip |
| **Programmiersprachen** | Alle gängigen Programmiersprachen |
| **Plattform-Dienste** | flowthings.io<br>Carriots<br>Beebotte<br>IBM Watson IoT<br>AWS IoT |
| **Aufbau des Protokolls und Overhead** | gering |
| **Energiesparendes Protokolldesign** | hoch |
| **Verbreitung/Support/Foren** | hoch |
| **Zukunftsfähigkeit** | hoch |
| **Zentraler Verwaltungsserver/Broker** | ja |
| **Vor- und Nachteile des Protokolls** | Vorteile:<br>• Sehr geringer Overhead<br>• Hoch skalierbar<br>• Viele IoT-Protokollfeatures<br>• Für ressourcenbeschränke Geräte geeignet<br>• Clientimplementierungen sind für alle gängigen Programmiersprachen verfügbar<br>Nachteile:<br>• Reine Request/Response-Architekturen sind mit MQTT nur mit zusätzlichem Aufwand umsetzbar |

Tabelle 4-4: Übersichtstabelle MQTT

### 4.2.2 HTTP/1.1

#### 4.2.2.1 Beschreibung

HTTP im Allgemeinen wird im WWW seit 1990 verwendet und war in der ersten verwendeten Version (http/0.9) ein einfaches Protokoll zur Übertragung von rudimentären Daten im Internet. Mit der Version 1.0 (RFC 1945) wurde das Protokoll erheblich erweitert. Vor allem die Möglichkeit der Nachrichtenübertragung im MIME-ähnlichen Stil. Durch HTTP/1.1 (RFC 2616) kamen strengere Vorgaben zur Benutzung hinzu, die eine zuverlässige Implementierung ihrer Funktionen sicherstellen sollen. http wird nicht nur für die Übertragung von HTML-Seiten verwendet, sondern kann fast überall genutzt werden, wo ein grundliegender Zugriff auf Hypermedia benötigt wird (z.B. FTP, SMTP). [64]

**Kurzbeschreibung: Weit verbreitetes und vielseitig unterstütztes Protokoll**

#### 4.2.2.2 Kommunikation

##### 4.2.2.2.1 Kommunikationsarchitektur

HTTP/1.1 ist eine klassische **Request/Response**-Architektur. Der Client stellt eine oder mehrere Anfragen an den Server und erhält dazu die Antwort. Die verschiedenen Nachrichtenarten, die dabei ausgetauscht werden können, sind unter 4.2.2.3.1 näher beschrieben. [65]

##### 4.2.2.2.2 Kommunikationsform

Bei der Kommunikation zwischen Client und Server findet die Kommunikation über HTTP/1.1 ausschließlich **synchron** statt. Im Unterschied zu HTTP/1.0, bei dem auf eine Anfrage eine Antwort erfolgt und dann die Verbindung abgebaut wird, kann bei HTTP/1.1 eine sogenannte persistente (keep-alive) Verbindung aufgebaut werden, bei der mehrere Anfragen über einen Verbindungsaufbau abgesetzt werden und dann gebündelt als Antwort gesendet werden. Dabei spricht man von **pipelining**. Dadurch wird die Anzahl der Pakete verringert und die Wartezeit zwischen der ersten und der letzten Anfrage reduziert [64] [65]. Nichtdestotrotz ist für die Dauer der Verbindung keine weitere Anfrage möglich. Dauert eine Anfrage also etwas länger, kommt die neue Anfrage nicht durch. Dabei spricht man vom Head-of-line Blocking. [66]

##### 4.2.2.2.3 Kommunikationswege und Adressierung

Da die Kommunikation nur zwischen Client und Server stattfindet, sind nur **Unicast**-Nachrichten möglich. Ein Versenden einer Nachricht an mehrere oder alle ist nicht möglich. [64]

Die Adressierung erfolgt über einen URI (Unified Ressource Identifier) genauer über einen **URL** (Unified Ressource Locator). Dabei kann die Adressierung der Ressource über den Hostnamen oder über die IP-Adresse erfolgen. [65]

#### 4.2.2.2.4 Kommunikationsrichtung

Durch die direkte Verbindung zwischen Nachrichtenanfrager und -sender (Request/Response-Architektur) besteht eine **bidirektionale** Kommunikation. [65]

### 4.2.2.3 Funktionsumfang

#### 4.2.2.3.1 Nachrichtenarten

Für die Kommunikation mit HTTP/1.1 stehen die im Folgenden beschriebenen Nachrichtenarten (hier Methoden genannt) zur Verfügung.

**OPTIONS**

Mit der Options-Methode kann der Client beim Server Informationen über die verfügbaren Kommunikationsoptionen abfragen, ohne eine Ressourcenabfrage durchzuführen. Weder die Anfrage noch die Antwort hat einen Inhalt (Body). Die notwendige Kommunikation findet in der normalen Ausführung nur über den Header statt. Weitere Informationen können jedoch auch über den Body erfolgen. [64]

**GET**

Über GET wird vom Client eine einfache Anfrage an den Server gestellt. Dabei bleibt der Body bei der Anfrage leer (es kann zwar etwas mitgeschickt werden, ohne ein Problem zu verursachen, jedoch besteht dafür die POST-Methode). Der Server antwortet mit allen Informationen, die für den angefragten URL verfügbar sind. [64]

**HEAD**

Die HEAD-Anfrage ist gleich aufgebaut wie bei GET, jedoch wird bei der Antwort nur der Header zurückgesendet. Diese Methode wird häufig dafür verwendet, URLs auf ihre Verfügbarkeit hin zu untersuchen. [64]

**POST**

Bei einer POST-Anfrage können im Gegensatz zu GET weitere Informationen im Body an den Server gesendet werden. Damit diese verwendet werden können, muss dieser Nachrichtenaustausch mit dem Server abgestimmt sein. Durch diese zusätzlichen Informationen ist die Antwort des Servers nicht mehr nur auf den URL zurückzuführen und kann auch keinen Inhalt haben. [64]

**PUT**

Über die PUT-Methode wird der Server dazu aufgefordert, die im Body enthaltene Entität unter dem angegeben URL zu speichern. Sofern an dieser Stelle schon eine Ressource besteht, soll diese überschrieben und andernfalls angelegt werden. Der Unterschied zur POST-Methode liegt darin, dass der bei der

POST-Methode angegebene URL den Nachrichteninhalt verarbeiten und bei PUT dieser unter der Ressource abgelegt werden soll. [64]

**DELETE**

Über die DELETE-Methode kann eine angegebene Ressource gelöscht werden. [64]

**TRACE**

Die TRACE-Methode dient der Verfolgung des Requests, der zwischen Client und Server über einen oder mehrere Proxys weitergeleitet wird. [64]

**CONNECT**

Über die CONNECT-Methode baut ein Proxy einen Tunnel zwischen Client und Server auf. [64]

#### 4.2.2.3.2 Nachrichtenformate

Bei HTTP ist der Header zwar in Textform, jedoch können alle Arten von Nachrichten (auch Binärdaten) übertragen werden. Damit der Empfänger weiß, wie er mit der Nachricht umgehen soll, muss der Datentyp (Content-Type) sowie die Art der Übertragung (Content-Transfer-Encoding) im Header angegeben werden. [64]

#### 4.2.2.3.3 Quality of Service

Bei HTTP/1.1 sind keine zusätzlichen Routinen zur Sicherstellung des Nachrichtenempfangs vorgesehen.

#### 4.2.2.3.4 Sicherheit

HTTP/1.1 unterstützt die Basisauthentifizierung über Benutzername und Passwort sowie die Digest Access Authentication. Bei der Basisauthentifizierung können die Daten direkt bei der Anfrage an den Host angefügt werden:

```
http://username:password@example.com/
```

Dabei werden beide aber im Klartext übertragen. Um dies zu umgehen besteht die Digest Access Authentication. Dabei wird nach der Anfrage beim Server eine zufällig erstellte Zeichenfolge an den Client gesendet. Dieser berechnet damit zusammen mit Benutzername und Passwort einen Hash-Wert, der an den Server als Authentifizierung zurückgesendet wird. Da HTTP/1.1 auch über TCP übertagen wird, kann zusätzlich eine Verschlüsselung über TLS erfolgen. [64]

#### 4.2.2.3.5 Sonstiges

Keine zusätzlichen Anmerkungen.

#### 4.2.2.4 Verwendungsmöglichkeiten

##### 4.2.2.4.1 Hardware

HTTP/1.1 wird von allen gängigen Computern und MCUs mit Wifi-Chip unterstützt. Das Protokoll hat zwar unter Umständen einen großen Overhead, welcher aber nur aus Text besteht und keine rechenintensiven Operationen erfordert. Zu Problemen kann es jedoch führen, wenn der Header zu groß für den verfügbaren RAM wird.

##### 4.2.2.4.2 Programmiersprachen

HTTP/1.1 wird von allen gängigen Programmiersprachen unterstützt. Eine explizite Liste war nirgendwo zu finden.

##### 4.2.2.4.3 Plattform-Dienste

Bei den Plattformdiensten ist es ähnlich. Jede der untersuchten Plattformdienste unterstützt HTTP/1.1. Es folgt die Liste mit den PaaS-Anbietern, die dahingehend untersucht wurden: [59]

- ThingSpeak
- data.sparkfun.com
- Runabove IoT Lab
- flowthings.io Carriots Ubidots
- GroveStreams
- Exosite
- Beebotte
- MODE
- Initial State
- Temboo
- Oracle Cloud
- IBM Watson IoT
- Azure IoT Hub
- AWS IoT
- Google Cloud Platform

#### 4.2.2.5 Aufbau des Protokolls und Overhead

Der HTTP/1.1-Header ist im Textformat und kann sehr lang werden. Im Bereich IoT werden viele Headerangaben, anders als bei der Anforderung einer Webseite, jedoch nicht benötigt [64]. Ein solcher kann schnell 400 Bytes erreichen. Im Folgenden ist ein Beispiel eines HTTP/1.1 Requests dargestellt. Dabei ist zu sehen, dass der Header einer Nachrichtenanfrage kurz gehalten werden kann, wobei der einer Nachrichtenübermittlung (Antwort) aufgrund der zusätzlich nötigen Angaben etwas größer werden kann. Je nach Anwendungsfall kann der Header sowohl beim Request als auch bei der Response um ein Vielfaches größer sein als im Beispiel.

```
REQUEST

GET /livingroom/temperature HTTP/1.1
Host: 192.168.13.6
Accept: application/x-www-form-urlencoded


RESPONSE

HTTP/1.1 200 OK
Server: 192.168.13.6
Content-Length: 21
Content-Type: application/x-www-form-urlencoded; charset=utf-8

temperature=28
```

Quellcode 4-2: HTTP/1.1 Request und Response Beispiel

Da der Header in Textform vorliegt und zudem einige Felder beinhalten kann, ist der Overhead im Vergleich zu einem fest definierten Header, der mit einfachen Flags auskommt, **hoch**.

#### 4.2.2.6 Energiesparendes Protokolldesign

Da HTTP/1.1 auch auf kleinen Geräten läuft, jedoch mit dem Text-Header einen erheblichen Overhead erzeugt, kann das energiesparende Protokolldesign nur als **mittel** bezeichnet werden.

#### 4.2.2.7 Verbreitung/Support/Foren

Bei der vorangegangenen Recherche konnten viele Informationen zu diesem Protokoll gefunden werden. Eine Google-Suche „HTTP/1.1“ ergab 50.600.000 Ergebnisse und bei Stackoverflow.com 65.788 Treffer. HTTP/1.1 wird von allen untersuchten PaaS-Plattformen unterstützt und auch von anderweitigen Anwendungen stark genutzt, weshalb man von einer sehr **hohen** Verbreitung sprechen kann (die Google-Suche nach HTTP ohne weiteren Zusatz ergab 13.600.000.000 Ergebnisse).

#### 4.2.2.8 Zukunftsfähigkeit

HTTP/1.1 wird derzeit wohl am meisten von allen IoT-Protokollen unterstützt. Das wird auch trotz HTTP/2 noch eine Weile so bleiben [67]. Dennoch ist das Protokoll nicht optimal für den Bereich IoT und wird hier zukünftig immer mehr an Bedeutung verlieren und von anderen Protokollen abgelöst werden. Deshalb, wenn auch aktuell noch gut vertreten, ist die Zukunftsfähigkeit eher **gering**.

#### 4.2.2.9 Vor- und Nachteile des Protokolls bezogen auf ihr Einsatzgebiet

**Vorteile**

- Nahezu uneingeschränkte Unterstützung
- Einfacher Protokollaufbau

**Nachteile**

- Großer Overhead bei jedem Request
- Head-of-line Blocking
- Keine Push-Nachrichten

#### 4.2.2.10 Zusammenfassung und Fazit

Derzeit ist HTTP/1.1 noch relevant, da es von so gut wie jeder Hard- und Software unterstützt wird. Es ist aufgrund mehrerer genannter Aspekte jedoch nicht optimal für das Internet der Dinge. Es jetzt schon abzuschreiben und zu vernachlässigen wäre etwas voreilig, denn viele der anderen Protokolle werden noch nicht flächendeckend unterstützt, wo dann auf HTTP/1.1 zurückgegriffen werden muss.

#### 4.2.2.11 Übersichtstabelle

| | |
|---|---|
| **Beschreibung** | Kurzbeschreibung: Weit verbreitetes und vielseitig unterstütztes Protokoll |
| **Kommunikation** | |
| **Kommunikationsarchitektur** | Request/Response |
| **Kommunikationsform** | Synchron, pipelining |
| **Kommunikationswege und Adressierung** | Unicast über URL |
| **Kommunikationsrichtung** | Bidirektional |
| **Funktionsumfang** | |
| **Nachrichtenarten** | OPTIONS<br>GET<br>HEAD<br>POST<br>PUT<br>DELETE<br>TRACE<br>CONNECT |
| **Nachrichtenformate** | Text und Binär |
| **Quality of Service** | – |
| **Sicherheit** | Authentifizierung über Benutzername und Passwort sowie eine verschlüsselte Übertragung der Benutzerdaten, Verschlüsselung über TLS |
| **Sonstiges** | – |
| **Verwendungsmöglichkeiten** | |
| **Hardware** | Alle gängigen Computer und MCUs mit WiFI-Chip |
| **Programmiersprachen** | Alle gängigen Programmiersprachen |
| **Plattform-Dienste** | ThingSpeak<br>data.sparkfun.com<br>Runabove IoT Lab<br>flowthings.io Carriots Ubidots<br>GroveStreams<br>Exosite<br>Beebotte<br>MODE<br>Initial State<br>Temboo<br>Oracle Cloud<br>IBM Watson IoT<br>Azure IoT Hub<br>AWS IoT<br>Google Cloud Platform |
| **Aufbau des Protokolls und Overhead** | hoch |
| **Energiesparendes Protokolldesign** | mittel |
| **Verbreitung/Support/Foren** | hoch |
| **Zukunftsfähigkeit** | gering |
| **Zentraler Verwaltungsserver/Broker** | nein |
| **Vor- und Nachteile des Protokolls** | Vorteile<br>• Nahezu uneingeschränkte Unterstützung<br>• Einfacher Protokollaufbau<br>Nachteile<br>• Gr. Overhead bei jedem Request<br>• Head-of-line Blocking<br>• Keine Push-Nachrichten |

Tabelle 4-5: Übersichtstabelle HTTP/1.1

### 4.2.3 HTTP/2

#### 4.2.3.1 Beschreibung

HTTP/2 (RFC 7540) greift einige gravierende Probleme von HTTP/1.1 auf und schafft Lösungen. Das Protokoll wurde unter Berücksichtigung der heutigen Anwendungen überarbeitet und weist vor allem Verbesserungen in der Flusskontrolle und im Overhead vor. Vorläufer des Protokolls ist SPDY. Dadurch konnte das Problem des Head-of-line Blocking behoben werden, was eine signifikante Steigerung der Geschwindigkeit beim Austausch von mehreren Nachrichten bewirkt. HTTP/2 ist abwärtskompatibel, was eine schnelle Verbreitung und Umstellung unterstützt. [68]

**Kurzbeschreibung: Erheblich verbessertes HTTP Protokoll mit besserem Datenaustausch**

#### 4.2.3.2 Kommunikation

##### 4.2.3.2.1 Kommunikationsarchitektur

Die bisher von HTTP/1.1 verwendete **Request/Response**-Architektur wurde um ein weiteres Element erweitert: HTTP/2 ermöglicht es dem Server per **Push**-Nachricht, zuvor angefragte Daten an den Client zu senden. Dies kann zwar noch nicht als Publish/Subscribe-Architektur bezeichnet werden, geht jedoch in die Richtung und bringt einen großen Mehrwert mit sich, vor allem auch für das Internet der Dinge. [68]

##### 4.2.3.2.2 Kommunikationsform

Neben der **synchronen** Kommunikation und dem **Pipelining** unterstützt HTTP/2 erstmals **asynchrone** Kommunikation und löst damit das Problem des Head-of-line Blocking. [68]

##### 4.2.3.2.3 Kommunikationswege und Adressierung

Beim Kommunikationsweg und der Adressierung hat sich gegenüber HTTP/1.1 nichts verändert. Nachrichten werden als **Unicast** versendet und anhand des **URL** adressiert. [68]

##### 4.2.3.2.4 Kommunikationsrichtung

Wie auch schon bei HTTP/1.1 handelt es sich um eine **bidirektionale** Verbindung. Der Unterschied ist hierbei, dass nicht für jede Anfrage eine neue TCP-Verbindung aufgebaut werden muss, sondern alle gebündelt über eine laufen können. Vor allem bei Webseiten mit vielen verschiedenen Elementen beschleunigt das die Kommunikation erheblich. [68]

#### 4.2.3.3 Funktionsumfang

##### 4.2.3.3.1 Nachrichtenarten

Bei HTTP/2 können alle bekannten Methoden verwendet werden. Bei der CONNECT-Methode ergab es jedoch eine kleine Änderung. Wo diese bei HTTP/1.1 zum Verbindungsaufbau über einen Proxy verwendet wurde, kommt sie hier zu einem ähnlichen Zweck zum Einsatz, um einen einzelnen HTTP/2-Stream mit einem entfernten Host zu verbinden. [68]

##### 4.2.3.3.2 Nachrichtenformate

Grundsätzlich unterstützt HTTP/2 alle Grundfunktionalitäten von HTTP/1.1 aus der Sicht der Anwendung. Tatsächlich besteht eine Nachricht aus mehreren binären Frames, die für die Übertragung verschiedener Teile der Nachricht zuständig sind. Während die Frames HEADERS und DATA die bisherige Funktionalität abdecken, ermöglichen die Frames SETTINGS, WINDOW_UPDATE oder PUSH_PROMISE neue HTTP/2-Features. Die Übertragung ist immer binär, wobei aber auch wie bisher sowohl **Text**- als auch **Binärdaten** übertragen werden können. Die binäre Übertragung erleichtert den Transport, da nicht zwischen verschiedenen Codierungen (z.B. base64) unterschieden werden muss. [68]

##### 4.2.3.3.3 Quality of Service

HTTP/2 hat zwar keine speziellen QoS-Optionen dazubekommen, jedoch fördert das Multiplexen die Zuverlässigkeit der Datenübertragung, da sich die Anfragen nicht mehr gegenseitig blockieren. [68]

##### 4.2.3.3.4 Sicherheit

Da die Grundfunktionalitäten von HTTP/1.1 unterstützt werden, ist zum einen auch die Authentifizierung per Benutzername und Passwort weiterhin möglich, zum anderen wird auch die Übertragung über TLS (Transport Layer Security) ab der Version TLS 1.2 vorgesehen. [68]

##### 4.2.3.3.5 Sonstiges

HTTP/2 beinhaltet auch **Header Compression**, was den Overhead vor allem bei Nachrichten nach dem Verbindungsaufbau deutlich verringert. Redundante Datenübertragung und damit einhergehender unnötiger Netztwerktraffic werden dadurch reduziert. Durch das Multiplexen und den Nachrichtenaustausch über **Streams** innerhalb einer Verbindung ist das Problem des gegenseitigen Blockierens der Nachrichten zwar beseitigt, jedoch ist dadurch die reihenfolgegetreue Anlieferung der angeforderten Daten nicht mehr gegeben. Dieses Problem wird bei HTTP/2 jedoch über die Verwendung von **Priorisierung** beseitigt. [68]

#### 4.2.3.4 Verwendungsmöglichkeiten

##### 4.2.3.4.1 Hardware

HTTP/2 stellt im Vergleich zu HTTP/1.1 keine zusätzlichen Anforderungen an die Hardware. Durch die Headerkomprimierung und die binäre Übertragung wird der RAM nicht ganz so beansprucht wie bei HTTP/1.1, was dem Protokoll im Bereich IoT zu Gute kommt.

##### 4.2.3.4.2 Programmiersprachen

HTTP/2 als Nachfolger von HTTP/1.1 wird von allen gängigen Programmiersprachen unterstützt. Eine genaue Liste kann im HTTP/2-Wiki auf Github eingesehen werden. [69]

##### 4.2.3.4.3 Plattform-Dienste

Durch die Abwärtskompatibilität kann HTTP/2 bei allen Plattformen verwendet werden, die auch HTTP/1.1 unterstützen, jedoch nur mit der Funktionalität von HTTP/1.1. Der Cloud Dienst Amazon CloudFront bietet seit September 2016 die direkte Unterstützung vom HTTP/2 und den neuen Funktionen an. [70]

#### 4.2.3.5 Aufbau des Protokolls und Overhead

Anders als bei HTTP/1.1 findet die Kommunikation nicht mehr über voneinander unabhängigen Paketen statt. Zudem werden die Requests in unterschiedliche Frames und Streams aufgeteilt. In Abbildung 4-2 ist die Aufteilung in HEADRES-Frame und DATA-Frame zu sehen. [71]

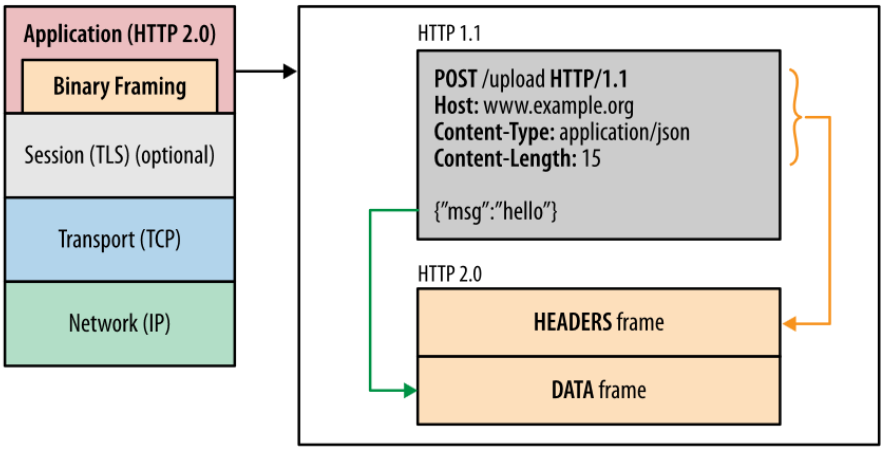


Abbildung 4-2: HTTP/2 Header im Vergleich zu HTTP/1.1 [71]

Durch die Aufteilung der Requests in Frames und Streams, welche über nur eine TPC-Verbindung transportiert werden, können die Header von folgenden Requests im Gegensatz zu HTTP/1.1 erheblich reduziert werden, da im Header nur noch das enthalten sein muss, was sich im Vergleich zum vorherigen Stream verändert hat. [71]

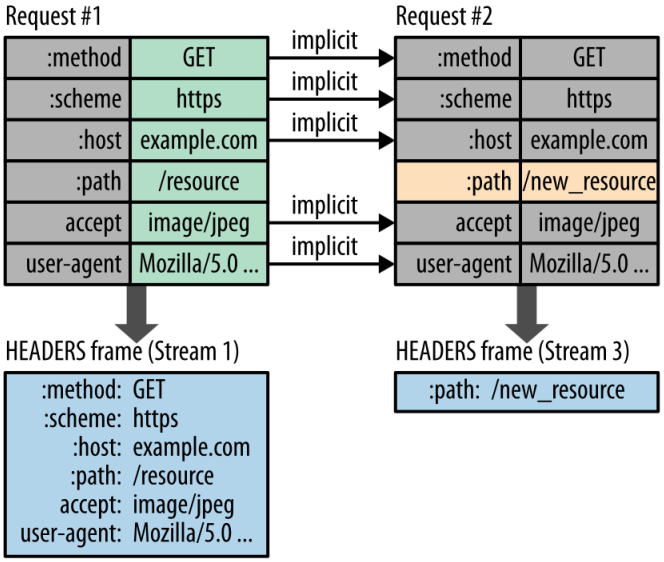


Abbildung 4-3: HTTP/2 Requests und Header Compression [71]

Diese Reduzierung des Overheads macht HTTP/2 damit vorteilhafter für die Verwendung im Bereich IoT. Da jedoch nur die Folgeanfragen innerhalb einer TCP-Verbindung von dieser Technik profitieren und gerade kleine Devices im IoT nur sehr wenig und selten senden, wird das Protokoll dadurch nicht grundsätzlich leichtgewichtiger. Dennoch ist der Overhead reduziert und kann nun als **mittel** eingestuft werden.

#### 4.2.3.6 Energiesparendes Protokolldesign

Mit den Änderungen in der Protokoll-Architektur verbessern sich auch die Eigenschaften beim Energieverbrauch. Dennoch ist es vor allem in Sensornetzen, die mit sehr wenig Energie auskommen müssen, nicht optimal. Deshalb kann das energiesparende Protokolldesign als **mittel** festgesetzt werden.

#### 4.2.3.7 Verbreitung/Support/Foren

Da das Protokoll noch sehr neu ist, ist es in der Anwendung noch nicht weit verbreitet. Eine Google-Suche „HTTP/2“ ergab 231.000.000 Ergebnisse und bei Stackoverflow.com 536.909 Treffer. Obwohl HTTP/2 gerade im Bereich Webhosting noch nicht sehr weit verbreitet ist, wird es bereits von den meisten Browsern unterstützt [72]. Für den Bereich IoT kann aktuell dennoch nur von einer **geringen** Verbreitung gesprochen werden. Dies kann sich jedoch in näherer Zukunft durchaus ändern.

#### 4.2.3.8 Zukunftsfähigkeit

Im Bereich des WWW bringt HTTP/2 lang ersehnte und notwendige Verbesserungen, die die Zukunft von HTTP sichern. Im Bereich IoT ist es stark von der Anwendung abhängig, ob HTTP/2 einem anderen Protokoll vorgezogen werden sollte. Vor allem in Sensornetzwerken sind hier andere Protokolle passender. Deshalb kann die Zukunftsfähigkeit in Bezug auf IoT als **mittel** eingestuft werden.

#### 4.2.3.9 Vor- und Nachteile des Protokolls bezogen auf ihr Einsatzgebiet

**Vorteile**

- Verbesserter Header
- Streams bei der Übertragung großer Datenmengen

**Nachteile**

- Derzeit noch geringe Verbreitung
- Nicht für IoT optimiert

#### 4.2.3.10 Zusammenfassung und Fazit

HTTP/2 bringt die notwendigen Verbesserungen für HTTP, die das Protokoll benötigt, um den Anforderungen der Zukunft gerecht zu werden. Aufgrund der bisher geringen Verbreitung und der Tatsache, dass HTTP nicht als M2M-Protokoll (Machine to Machine) entwickelt wurde, macht die Verwendung nur dann Sinn, wenn zur Anwendung Plattformen und andere Geräte mitverwendet werden, die ausschließlich eine Anbindung über HTTP unterstützen.

### 4.2.3.11 Übersichtstabelle

| | |
|---|---|
| **Beschreibung** | Erheblich verbessertes HTTP-Protokoll mit besserem Datenaustausch |
| **Kommunikation** | |
| **Kommunikationsarchitektur** | Request/Response + Push |
| **Kommunikationsform** | Synchron, asynchron, pipelining |
| **Kommunikationswege und Adressierung** | Unicast über URL |
| **Kommunikationsrichtung** | bidirektional |
| **Funktionsumfang** | |
| **Nachrichtenarten** | OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE, CONNECT,<br>PUSH |
| **Nachrichtenformate** | Text und Binär |
| **Quality of Service** | (Multiplexen) |
| **Sicherheit** | Authentifizierung über Benutzername und Passwort sowie eine verschlüsselte Übertragung der Benutzerdaten, Verschlüsselung über TLS |
| **Sonstiges** | Header Compression<br>Streams<br>Priorisierung |
| **Verwendungsmöglichkeiten** | |
| **Hardware** | Alle gängigen Computer und MCUs mit WiFI-Chip |
| **Programmiersprachen** | Alle gängigen Programmiersprachen |
| **Plattform-Dienste** | ThingSpeak<br>data.sparkfun.com<br>Runabove IoT Lab<br>flowthings.io Carriots Ubidots<br>GroveStreams<br>Exosite<br>Beebotte<br>MODE<br>Initial State<br>Temboo<br>Oracle Cloud<br>IBM Watson IoT<br>Azure IoT Hub<br>AWS IoT<br>Google Cloud Platform<br><br>*Grau, da nur per Downgrade auf HTTP/ 1.1 |
| **Aufbau des Protokolls und Overhead** | mittel |
| **Energiesparendes Protokolldesign** | mittel |
| **Verbreitung/Support/Foren** | gering |
| **Zukunftsfähigkeit** | mittel |
| **Zentraler Verwaltungsserver/Broker** | nein |
| **Vor- und Nachteile des Protokolls** | **Vorteile**<br>• Verbesserter Header<br>• Streams bei der Übertragung großer Datenmengen<br><br>**Nachteile**<br>• Derzeit noch geringe Verbreitung<br>• Nicht für IoT optimiert |

Tabelle 4-6: Übersichtstabelle HTTP/2

### 4.2.4 REST

#### 4.2.4.1 Beschreibung

Während der einführenden Recherche wurde REST (Representational State Transfer) oft gleichwertig im Zusammenhang mit anderen Anwendungsschicht-Protokollen genannt. Dadurch entstand der Eindruck, REST sei auch ein Protokoll. Bei REST handelt es sich jedoch um einen Architekturstil, der dazu dienen soll, die Schnittstellen in einem größtenteils durch Wiederverwendung und Erweiterung gewachsenen Web übersichtlich zu halten. Durch diese kompakten Schnittstellen und einer gegebenen Adressierbarkeit soll auch die Komposition einzelner Subsysteme erleichtert werden. [73] Obwohl REST bereits im Jahr 2000 von Roy Fielding in seiner Dissertation [74] vorgestellt wurde, sind diese Prinzipien mit dem Aufkommen von IoT aktueller denn je. Erfüllt ein Protokoll bzw. eine Anwendung bei ihrer Abfrage die Prinzipien von REST, so spricht man von RESTful.

Da sich die zu untersuchenden Kriterien jedoch großteils nur auf Protokolle anwenden lassen, wird im Folgenden die Struktur abgewandelt und die Prinzipien von REST kurz beschrieben.

#### 4.2.4.2 Client-Server

Als grundlegende Voraussetzung für eine RESTful-Architektur gilt die Client-Server-Architektur, bei der beide unabhängig voneinander agieren und der Client beim Server einen gewissen Dienst abrufen kann. [73]

#### 4.2.4.3 Zustandslosigkeit

Dabei geht es darum, dass jede Nachricht in sich abgeschlossen ist und weder der Server noch die Anwendung Zustandsinformationen zwischen zwei Nachrichten speichert. Dadurch müssen auch keine Sitzungsdaten gespeichert werden. Alle für die Verarbeitung notwendigen Informationen müssen in der Anfrage enthalten sein. Dadurch kann die Schnittstelle klar und deutlich gehalten werden, ohne dass aus dem Kontext ermittelt werden muss, in welcher Relation eine Anfrage zu einer vorherigen steht. Dies begünstigt zudem die Skalierbarkeit, da Anfragen problemlos auf beliebige Maschinen verteilt werden können. [73]

#### 4.2.4.4 Caching

Durch das Caching soll unnötiger Datenverkehr vermieden werden. Liegen bereits die aktuellen Daten vor, bringt eine erneute Übertragung keinen Mehrwert. Durch das Caching wird auch die Latenz verringert. [73]

#### 4.2.4.5 Einheitliche Schnittstellen

##### 4.2.4.5.1 Adressierbarkeit von Ressourcen

Ressourcen gelten als Grundbausteine einer RESTful API. Diese müssen eindeutig adressierbar sein. Dies erfolgt über den URL. Dabei sollen jedoch keine Parameter übergeben werden oder nur ein spezieller Zustand abgefragt werden.

| | | |
|---|---|---|
| Beispiele: | /livingroom/lights?action=getStatus&light=1 | nicht RESTful |
| | /livingroom/light/1/on | nicht RESTful |
| | /livingroom/light/1/status | RESTful |

Durch diese konsistente Adressierbarkeit kann ein Webservice auch einfach als Teil eines Mashups weiterverwendet werden. Über die Adressierung soll zudem nicht angegeben werden, in welchem Format die Daten angefordert oder übertragen werden. Die Ressourcen und die Repräsentationsform sollen voneinander getrennt angesteuert werden können. [73]

##### 4.2.4.5.2 Repräsentation der Ressourcen

Eine Ressource muss über mindestens eine Repräsentationsform verfügen. Anfragen an einen Webservice durch einen Browser ergeben in der Regel die Repräsentation der Daten als HTML-Dokument. Werden jedoch reine Daten abgefragt oder übertragen, können diese in einem von der Anwendung gut zu verarbeitenden Format (XML, JSON) sein. Da die Repräsentation unabhängig von der Adressierung sein muss, kann das Format nicht über einen Parameter im URL festgelegt werden. Die Auswahl kann zum einen anhand der Header-Information erfolgen, wobei vom Server ermittelt wird, welches Format für den entsprechenden Client in Frage kommt. Zum anderen bietet z.B. HTTP das Attribut „Accept“ im Header an, über das das gewünschte Format angegeben werden kann. Über das Attribut „Content-Type“ wird hingegen angegeben, in welchem Format der im Body enthaltene Content ist. Durch diese Entkopplung der Ressource von der Präsentation ist z.B. eine flexible Erweiterung der Repräsentationsformen möglich, ohne die Logik zu verändern. [73]

##### 4.2.4.5.3 Selbstbeschreibende Methoden

Zu einem einheitlichen Interface gehörten auch einheitliche Methoden, die optimalerweise wenig Anleitung bedürfen. Eine Applikation sollte in der Lage sein, alle Operationen mit diesen wenigen einheitlichen Basismethoden durchführen zu können. Dies erfordert einiges an Aufwand bei der Entwicklung der Applikation, führt aber zu einer einfachen übersichtlichen Schnittstelle, die nicht immer wieder an neue Anwendungen angepasst werden muss. REST wird meist mit der Verwendung von HTTP genannt. Dabei beschränken sich die Methoden auf GET, POST, PUT und DELETE. Mit diesen Methoden können Daten abgerufen, verändert, erstellt und gelöscht werden. [73]

#### 4.2.4.5.4 Hypermedia as the Engine of Application State (HATEOAS)

Roy Fielding formulierte es bereits in seiner Dissertation und hat es seitdem immer wieder angemerkt: „REST APIs must be hypertext-driven“ [74]. Damit ist HATEOAS eine essenzielle Bedingung für jede REST-Architektur und wird deshalb noch genauer beschrieben.

**Hypermedia**

Hypermedia ist eine Komposition aus Hypertext und Multimedia. Hyper ist griechisch und bedeutet „über“ oder „hinaus“, wodurch Hypertext bedeutet, dass es über den eigentlichen Text hinausgeht. Dies geschieht durch Verlinkung auf einen anderen Text. Hypermedia steht also für einen um eine Multimediakomponente erweiterten Hypertext. [74]

**Engine**

In diesem Zusammenhang steht Engine für einen Zustandsautomat. Ein Zustandsautomat kann in UML als Zustandsdiagramm modelliert werden, bei dem die Zustände mit Kanten verbunden sind, welche die Übergänge von einem Zustand zu einem oder mehreren anderen Zuständen darstellen. Der Zustand einer RESTful-Applikation ist also vollständig durch ihre Repräsentation abgebildet. Das Zustandsdiagramm zeigt somit alle möglichen Zustände (Repräsentationen) und Übergange einer Anwendung. Die Zustandsübergänge werden durch Hypermedia gesteuert. Deshalb „Repräsentativer Zustandsübergang“ (Representational State Transfer). [74]

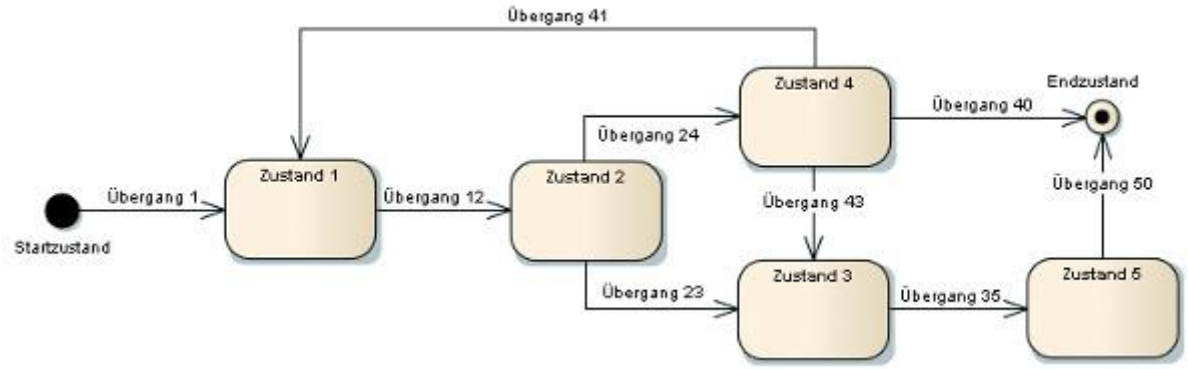


Abbildung 4-4: UML-Zustandsdiagramm [74]

**Application**

Application hat in diesem Fall zwei Bedeutungen: Zum einen bezieht es sich auf die Anwendung im Sinne von Software und zum anderen auf die Anwendung der Prinzipien der REST-Architektur. [74]

**State**

State, also Zustand, ergänzt den vorherigen Begriff des Engines und bringt die drei Worte Engine, Application und State in den Zusammenhang: Ein Automat besteht aus Zuständen und Übergängen und beschreibt das Verhalten einer Anwendung. [75]

#### 4.2.4.6 Mehrschichtige Systeme

Durch die Verwendung von mehrschichtigen Systemen wird die Wartung, Wiederverwendbarkeit und Erweiterbarkeit erleichtert und sorgt dafür, dass Anwender nur eine Schnittstelle bedienen müssen, unabhängig der darunterliegenden Ebenen. Dadurch wird die Architektur insgesamt vereinfacht. [75]

#### 4.2.4.7 Code on Demand

Unter Code on Demand wird die Möglichkeit definiert, bei der im Bedarfsfall auch ein Code zu lokalen Ausführung an den Client übertragen werden kann. Ein einfaches Beispiel dafür ist ein in einer HTML-Repräsentation enthaltener JavaScript-Code. Diese Forderung ist jedoch nur optional. [75]

#### 4.2.4.8 Zusammenfassung und Fazit

> *“I am getting frustrated by the number of people calling any HTTP-based interface a REST API.“ - Roy T. Fielding [76]*

Wie bereits zu Beginn dieses Kapitels zu REST erwähnt, wird der Begriff häufig falsch verwendet oder es werden viele HTTP-basierte APIs als RESTful bezeichnet, ohne dass sie es tatsächlich sind. Diese Vorgaben werden im Bereich IoT sehr oft, vor allem bei PaaS-Systemen [59], für ihre HTTP-APIs verwendet und sind deshalb eine wichtige Komponente für das Internet der Dinge.

### 4.2.5 CoAP

#### 4.2.5.1 Beschreibung

CoAP (RFC 7252) steht für Constrained Application Protocol und ist für die Verwendung in eingeschränkten Anwendungen konzipiert, die mit wenig Energie und einem geringen Datenverkehr auskommen müssen. Entwickelt wurde es von der IETF Contrained RESTful environments (CoRE) Working Group. Einer der Hauptziele im Protokolldesign ist ein geringer Overhead. Zwar können mit dem bereits vorgestellten 6LoWPAN durch Zerlegung auch größere Pakete über ein eingeschränktes Netz versendet werden, jedoch verringert dies nicht den Datenverkehr, sondern schleift z.B. auch einen für Sensornetze viel zu großen HTTP-Header mit. Mit CoAP sollte ein Protokoll geschaffen werden, das nach den Prinzipien der REST-Architektur und ähnlich dem HTTP-Protokoll für M2M-Netzwerke optimiert ist. CoAP versteht sich jedoch nicht als ein komprimiertes RESTful HTTP-Protokoll, sondern als ein speziell für eingeschränkte M2M-Netze konzipiertes Protokoll, welches nicht nur eigene Funktionalitäten mitbringt, sondern sich auch einfach in HTTP umwandeln lässt. Dadurch können Daten von Geräten aus Sensornetzwerken auch sehr einfach in bestehende Web-Netze integriert werden. [77]

**Kurzbeschreibung: Leichtgewichtiges M2M-Protokoll mit der einfachen Möglichkeit der Überführung in HTML**

#### 4.2.5.2 Kommunikation

##### 4.2.5.2.1 Kommunikationsarchitektur

Durch die Anlehnung an HTTP beruht die CoAP-Architektur auch auf dem **Request/Response**-Modell. Die Kommunikation findet mit den unter 4.2.5.3.1 vorgestellten Methoden statt [77]. Im Unterschied zu HTTP bietet CoAP jedoch die Möglichkeit, eine Ressource mit einer GET Anfrage zu abonnieren. Anders als bei MQTT ist das Abonnement jedoch zeitlich begrenzt. Der Client muss dann vor Ablauf der sogenannten „Subscription-lifetime“ erneut eine Anfrage stellen. Sobald sich der Zustand der Ressource ändert, wird der Client vom Server über die Änderung durch eine „notification response“ informiert. Ebenfalls in der Benachrichtigung enthalten ist die restliche Laufzeit der Subscription [78]. Dieses Verfahren entspricht zwar nicht zu 100% dem **Publish/Subscribe**-Modell, kann aber unter Vorbehalt als solches bezeichnet werden.

##### 4.2.5.2.2 Kommunikationsform

Während die Kommunikation über Request/Response immer **synchron** abläuft [77] kommt mit der Möglichkeit, zeitverzögert und mehrfach eine Antwort auf einen Request zu erhalten, eine **asynchrone** Kommunikationsform hinzu [78].

#### 4.2.5.2.3 Kommunikationswege und Adressierung

Durch die Verwendung von UDP kann CoAP nicht nur **Unicast**-Nachrichten an einen Empfänger versenden, sondern ebenfalls **Muiticast**-Nachrichten an mehrere Empfänger gleichzeitig. Die Adressierung erfolgt über den CoAP-URI, welcher dem von HTTP ähnelt.

CoAP-URI = "coap:" "//" host [ ":" port ] path-abempty [ "?" query ]

Als Host kann entweder ein Name oder eine IP-Adresse verwendet werden. [77]

#### 4.2.5.2.4 Kommunikationsrichtung

Da Endgeräte bei CoAP auch direkt miteinander kommunizieren können, besteht eine **bidirektionale** Kommunikation.

### 4.2.5.3 Funktionsumfang

#### 4.2.5.3.1 Nachrichtenarten

CoAp verwendet vier bereits von HTTP bekannte Methoden: GET, POST, PUT und DELETE. Deren Funktion gleicht der von HTTP und kann im Kapitel 4.2.2.3.1 nachgelesen werden. [77]

#### 4.2.5.3.2 Nachrichtenformate

CoAP unterstützt wie auch HTTP die Übertragung von **Text-** und **Binärdaten**. Wird kein Content-Typ angegeben, wird von reinem Text ausgegangen. [77]

#### 4.2.5.3.3 Quality of Service

Bei CoAP besteht die Option, den Empfang von Nachrichten bestätigen zu lassen, um deren Zustellung sicherzustellen. Die in diesem Zusammenhang stehenden Nachrichten werden im Folgenden kurz beschrieben. [77]

**Non-confirmable Message**

Dies ist die Standardnachricht, bei der der Empfang nicht bestätigt werden muss. Diese wird bei regulären Anwendungen, wie dem einfachen Auslesen eines Sensors verwendet. [77]

**Confirmable Message**

Bei dieser Nachricht muss der Empfang gegenüber dem Sender bestätigt werden. Dies erfolgt über eine Acknowledgement Message. [77]

**Acknowledgement Message**

Beim erfolgreichen Empfang einer Confonfirmable Message wird dieser durch eine Acknowledgement Message bestätigt. Mit dieser Nachricht wird lediglich der Empfang bestätigt, jedoch nicht, ob der Inhalt korrekt und wie erwartet übertragen wurde. [77]

**Reset Message**

Mit dem Senden einer Reset Message gibt der Empfänger an, dass er die Nachricht nicht verarbeiten kann. Dies kann zudem auch verwendet werden, um die Erreichbarkeit eines Endgerätes zu ermitteln. Dazu wird einfach eine leere Confirmable Message gesendet. [77]

#### 4.2.5.3.4 Sicherheit

Bei CoAP besteht wie bei HTTP auch die Möglichkeit zur Authentifizierung über **Benutzername und Passwort**. Zusätzlich kann zur Verschlüsselung der Daten DTLS (Datagramm Transport Layer Security) verwendet werden. Dies ist auch am URI zu erkennen, da dieser dann mit „coaps“ beginnt. Vier Verschlüsselungsmodi stehen zur Verfügung. [77]

**NoSec**

Bei diesem Modus wird keine Verschlüsselung verwendet. Die Nachricht wird einfach als coap gekennzeichnet und über UDP versendet. [77]

**PreSharedKey**

Hierbei ist DTLS aktiviert und es liegt eine Liste vor, welcher Schlüssel mit welchen Endgeräten verwendet werden kann. Im Extremfall besteht für jedes Endgerät ein eigener Schlüssel. Andererseits ist ein Schlüssel, der für mehr als zwei Endgeräte verwendet wird, nur noch dafür verwendbar, um die Zugehörigkeit des Geräts zu einer Gruppe zu bestätigen, nicht mehr als einzelner Knotenpunkt. [77]

**RawPublicKey**

Hierbei ist DTLS aktiviert und die Verschlüsselung erfolgt durch ein asymmetrisches Verfahren mit einem Schlüsselpaar. Ein Zertifikat für das Schlüsselpaar liegt nicht vor. [77]

**Certificate**

Hierbei ist DTLS aktiviert und die Verschlüsselung erfolgt durch ein asymmetrisches Verfahren mit einem Schlüsselpaar. Im Unterschied zum RawPublicKey-Modus ist das Schlüsselpaar jedoch von einem „Trust Root“ zertifiziert. [77]

#### 4.2.5.3.5 Sonstiges

**Discovery**

Sofern ein Endgerät unbekannt ist, kann es durch eine Multicast-Anfrage an alle Knoten mittels „All CoAP Nodes“-Adresse gefunden werden. Zusätzlich unterstützt CoAP ein Well-Known-Interface für Ressource Discovery. Dabei wird dann eine Liste der verfügbaren Ressourcen im CoRE Link Format zurückgeliefert. [77]

z.B. REQ: GET /.well-known/core

RES: 2.05 Content
</sensors/temp>;if="sensor",
</sensors/light>;if="sensor"

**Piggybacked Response**

Um den Datenstrom möglichst gering zu halten besteht die Möglichkeit, bei einer Anfrage über eine Confirmable Message die Response (sofern direkt verfügbar) an die Acknowledgement Message anzufügen. Dadurch entfällt das separate Senden einer Response Nachricht. [77]

### 4.2.5.4 Verwendungsmöglichkeiten

#### 4.2.5.4.1 Hardware

Grundsätzlich kann CoAP von allen gängigen Computern und MCUs verwendet werden. Im Gegensatz zu HTTP ist das Protokoll jedoch speziell für die Verwendung in einer eingeschränkten Umgebung konzipiert. Aus diesem Grund eignet sich CoAP besonders für Sensornetzwerke, die nur geringe Ressourcen verfügbar haben. [77]

#### 4.2.5.4.2 Programmiersprachen

Durch seinen einfachen Protokollaufbau, kann es sehr einfach von allen gängigen Programmiersprachen verwendet werden. Für die, bei denen dies nicht ohne weiteres möglich ist, sind Implementierungen verfügbar. [79]

#### 4.2.5.4.3 Plattform-Dienste

Unter den untersuchten Plattformdiensten unterstützt nur Exosite die Verwendung von CoAP. [59]

#### 4.2.5.5 Aufbau des Protokolls und Overhead

Bei CoAP ist der Header im Gegensatz zu HTTP binär. Dies ermöglicht eine deutliche Verringerung der Overheads. Ein Paket ist mindestens 4 Byte groß. Die tatsächliche Größe richtet sich nach dem Inhalt. Dieser geringe Overhead resultiert daraus, dass die Nachrichtenmethode und weitere Optionen nicht wie bei HTTP im Klartext übergeben werden, sondern anhand von Codes. Die Bestandteile des Headers werden kurz vorgestellt. [77]

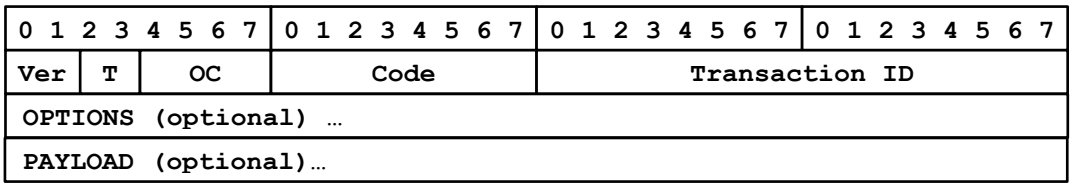


Abbildung 4-5: CoAP Header

**Ver**: CoAP-Version, 2-bit unsigned Integer

**T**: Nachrichtentyp (QoS), 2-bit unsigned Integer
00 (0) = Confirmable
01 (1) = Non-Confirmable
10 (2) = Acknowledgement
11 (3) = Reset

**OC**: Option Count, 4-bit unsigned integer
Gibt an, ob ein OPTION-Header enthalten ist und wenn ja, wie viele Optionen. Bei 0 folgt dem Header direkt der Payload (falls vorhanden).

**Code**: 8-bit unsigned Integer
Über dieses Feld wird bei einem Request die Methode gewählt (1–10). Bei einem Response wird hier Response-Codeübertragen (40–255).
00000001 (1) = GET
00000010 (2) = POST
00000011 (3) = PUT
00000100 (4) = DELETE
01010000 (80) = OK (HTTP 200)

#### 4.2.5.6 Energiesparendes Protokolldesign

Da CoAP vor allem als Protokoll für eingeschränkte Netzwerke entwickelt wurde, kann das energiesparende Protokolldesign als **hoch** eingestuft werden.

#### 4.2.5.7 Verbreitung/Support/Foren

Eine Google-Suche „CoAP“ ergab 885.000 Ergebnisse und bei Stackoverflow.com 237 Treffer. Obwohl CoAP speziell für die Verwendung in eingeschränkten Netzen entwickelt wurde, ist die allgemeine Verbreitung sehr **gering**. Das könnte unter Umständen daran liegen, dass viele Sensornetzwerke nicht auf IP basieren und zudem auch eigene Anwendungsschicht-Protokolle verwenden.

#### 4.2.5.8 Zukunftsfähigkeit

Das Protokolldesign und dessen geringe Anforderungen an die Hardware schaffen eine gute Basis, damit das Protokoll auch in Zukunft verwendet werden kann. Ob sich an der Verbreitung noch etwas ändert, wird sich zeigen. Die Zukunftsfähigkeit des Protokolls ist **hoch**.

#### 4.2.5.9 Vor- und Nachteile des Protokolls bezogen auf ihr Einsatzgebiet

**Vorteile**

- Offener Standard
- Web-Interoperabilität (leicht in HTTP umzuwandeln)
- Geringer Overhead

**Nachteile**

- Primär für die direkte Kommunikation zwischen zwei Geräten konzipiert.
- Kein richtiges Publish/Subscribe

#### 4.2.5.10 Zusammenfassung und Fazit

Durch seinen geringen Overhead und die einfache Umwandlung in HTTP schafft CoAP gute Voraussetzungen, um eine einfache Kommunikation zwischen Sensornetzwerken und dem Internet zu ermöglichen. Leider ist das Protokoll nicht allzu häufig vertreten und wird gerade von PaaS-Systemen so gut wie nicht unterstützt. Sinnvoll oder nicht, das hängt bei CoAP stark vom Einsatzgebiet ab.

#### 4.2.5.11 Übersichtstabelle

| | |
|---|---|
| **Beschreibung** | Leichtgewichtiges M2M-Protokoll mit der einfachen Möglichkeit der Überführung in HTML |
| **Kommunikation** | |
| **Kommunikationsarchitektur** | Request/Response<br>(Publish/Subscribe) |
| **Kommunikationsform** | Synchron und Asynchron |
| **Kommunikationswege und Adressierung** | Unicast und Multicast über URI |
| **Kommunikationsrichtung** | Bidirektional |
| **Funktionsumfang** | |
| **Nachrichtenarten** | POST, GET, PUT, DELETE |
| **Nachrichtenformate** | Text und Binär |
| **Quality of Service** | Confirmable Message<br>NonConfirmable Message |
| **Sicherheit** | Authentifizierung über Benutzername und Passwort und DTSL<br>(sharedKey, PublicKey, Certificate) |
| **Sonstiges** | Discovery<br>Piggybacked Response |
| **Verwendungsmöglichkeiten** | |
| **Hardware** | Alle gängigen Computer und MCUs mit WiFI-Chip |
| **Programmiersprachen** | Alle gängigen Programmiersprachen |
| **Plattform-Dienste** | Exosite |
| **Aufbau des Protokolls und Overhead** | gering |
| **Energiesparendes Protokolldesign** | hoch |
| **Verbreitung/Support/Foren** | gering |
| **Zukunftsfähigkeit** | hoch |
| **Zentraler Verwaltungsserver/Broker** | nein |
| **Vor- und Nachteile des Protokolls** | **Vorteile**<br>• Offener Standard<br>• Web-Interoperabilität (leicht in HTTP umzuwandeln)<br>• Geringer Overhead<br>**Nachteile**<br>• Primär für die direkte Kommunikation zwischen zwei Geräten konzipiert<br>• Kein richtiges Publish/Subscribe |

Tabelle 4-7: Übersichtstabelle CoAP

### 4.2.6 XMPP

#### 4.2.6.1 Beschreibung

Das Extensible Messaging and Presence Protocol (RFC 6120 und RFC6122), also das Erweiterbare Nachrichten- und Anwesenheitsprotokoll (früher Jabber) kommt aus dem Bereich des Instant Messaging. Es ermöglicht den Austausch von XML (Extensible Markup Language)-Daten zwischen zwei oder mehreren Netzwerkeinheiten nahezu in Echtzeit. Wie der Name schon sagt, ist es erweiterbar, was die Einsatzmöglichkeiten ausweitet. [80]

**Kurzbeschreibung: Vielfältig einsetzbares und erweiterbares Protokoll**

#### 4.2.6.2 Kommunikation

##### 4.2.6.2.1 Kommunikationsarchitektur

Die Kommunikation bei XMPP findet über sogenannte Streams statt, welche als Container für den Nachrichtenaustausch verwendet werden. Standardmäßig wird ein Stream zwischen zwei Netzwerkentitäten aufgebaut. Die Nachricht selbst wird in einem „XML Stanza“ übertragen. Diese kann entweder eine einfache Nachricht (message), eine Anwesenheits-/Aktivitätsanfrage (presens) oder eine Abfrage (iq = info/query) sein. Über iq kommt die **Request/Response**-Architektur zum Einsatz. [80]

Aufgrund der Erweiterbarkeit des Protokolls lohnt sich die Recherche nach Erweiterungen. Für XMPP besteht eine sogenannte Extension, mit der auch eine **Puplish/Subscribe**-Architektur umgesetzt werden kann. [81]

##### 4.2.6.2.2 Kommunikationsform

XMPP unterstützt sowohl **synchrone** als auch **asynchrone** Kommunikation [80]. Zusätzlich gibt es auch hier eine Erweiterung, mit der auch **Pipelining** möglich ist. Es besteht somit die freie Wahl der Kommunikationsform [82].

##### 4.2.6.2.3 Kommunikationswege und Adressierung

Durch seinen Ursprung im Instant Messaging handelt es sich bei XMPP zunächst um eine Ende-zu-Ende-Kommunikation über **Unicast** [80]. Über eine Erweiterung können jedoch auch **Multicast**-Nachrichten versendet werden. Voraussetzung dafür ist jedoch, dass der Server über diese Erweiterung verfügt. Der Client kann dies über eine spezielle Info-Abfrage an den Server vorab feststellen [83]. Vergleichbar wie bei MQTT läuft die Verbindung über einen zentralen Verwaltungsserver. Zusätzlich können Nachrichten auch per **Broadcast** an alle mit dem Server verbundenen Endgeräte gesendet werden. [80]

Ähnlich wie bei einer Email werden bei XMPP **global einzigartige Adressen** verwendet (basierend auf DNS), um die Nachrichten über das Netzwerk zu führen. Eine Serveradresse besteht aus dem vollständigen Domainnamen

(im.example.com), Accounts bestehen aus Benutzername und Domainname und werden „bare JID“ genannt (erica@im.example.com) und ein direkt verbundenes Gerät, welches in Verwendung ist, enthält zudem den Ressourcenabschnitt und wird „full JID“ genannt (erica@im.examlpe.com/livingroom). Die Bezeichnung JID oder Jabber ID erinnern an die Ursprünge des Formats. Die Adressierungsformate sind in einem gesonderten RFC dokumentiert. [84]

#### 4.2.6.2.4 Kommunikationsrichtung

Da die Kommunikation in beide Richtungen funktioniert, handelt es sich um eine **bidirektionale** Kommunikation. Bei der Verbindungsaufnahme mit einem Empfänger sind die Nachrichten jedoch zunächst **unidirektional** bis die Art der Verbindung geklärt ist (Stream Negotiation) [80].

### 4.2.6.3 Funktionsumfang

#### 4.2.6.3.1 Nachrichtenarten

Sobald ein XMPP-Stream aufgebaut wurde, können mehrere Nachrichten bis zum Beenden der Verbindung gesendet werden. Dabei wird zwischen drei Nachrichtenarten unterschieden. [80]

**<message/>**

Eine message-Nachricht ist eine Art Push-Nachricht, bei der ein Endgerät eine Nachricht an ein anderes sendet. Das einzige Attribut, das diese Nachricht enthalten muss, ist der Empfänger („to“). [80]

**<presence>**
Die presence-Nachricht ist als Broadcast-Nachricht gedacht, mit der zum einen eine Abfrage durchgeführt werden kann, welche Geräte verfügbar sind und zum anderen auch, um eine Art Publish-Nachricht zu versenden. Dies ist aber nicht mit dem Pulish/Subscribe-Verfahren aus der Erweiterung zu verwechseln. Eine presence-Nachricht kann also an einen expliziten Empfänger geschickt werden (über „to“) oder an alle (ohne „to“). [80]

**<iq>**

Eine iq-Nachricht (iq=Info/Query) ist für ein Request/Response-Verfahren, ähnlich dem bereits bei HTTP vorgestellten Verfahren. Es ermöglicht einem Gerät eine Anfrage an ein anderes zu stellen und diesem darauf zu antworten. Der Inhalt der Nachricht folgt dem vorgegebenen XML-Schema. XMPP unterschiedet bei den Anfragen zwischen den zwei Methoden get und set sowie zwischen den zwei möglichen Antworten result und error die im „type“-Attribut angegeben werden. [80]

get: Fragt eine bestimmte im Body definierte Information an
set: Übergibt Daten, die entweder aktualisiert oder angefügt werden sollen
result: Übergibt die angefragten Daten oder bestätigt ein erfolgreiches „set“
error: Meldet einen Fehler bei der Anfrage oder Ausführung (mit Angabe)

```
Requesting                  Responding
  Entity                      Entity
----------                  ----------
    |                           |
    | <iq id='1' type='get'>    |
    |   [ ... payload ... ]     |
    | </iq>                     |
    | ------------------------> |
    |                           |
    | <iq id='1' type='result'> |
    |   [ ... payload ... ]     |
    | </iq>                     |
    | <------------------------ |
    |                           |
    | <iq id='2' type='set'>    |
    |   [ ... payload ... ]     |
    | </iq>                     |
    | ------------------------> |
    |                           |
    | <iq id='2' type='error'>  |
    |   [ ... condition ... ]   |
    | </iq>                     |
    | <------------------------ |
    |                           |
```

Quellcode 4-3: XMPP Request/Response Kommunikation [80]

Über eine Erweiterung kann neben dem Request/Response- auch ein Publish/Subscribe-Verfahren verwendet werden. Das Publishen und Subscriben erfolgt ebenfalls über get, set, error und result. Der Subscriber kann sich dabei über zusätzliche Angaben und Attribute für verschiedene Informationen einschreiben. Über das „id“-Attribut im iq-Tag wird angegeben, welche Operation (z.B. Publishen, Subscriben) durchgeführt werden soll.

```
<iq type='set'
    from='francisco@denmark.lit/barracks'
    to='pubsub.shakespeare.lit'
    id='sub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <subscribe
        node='princely_musings'
        jid='francisco@denmark.lit'/>
  </pubsub>
</iq>
```

Quellcode 4-4: XMPP Subscribe-Anfrage [81]

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='sub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <subscription
        node='princely_musings'
        jid='francisco@denmark.lit'
        subid=
        'ba49252aaa4f5d320c24d3766f0bdcade78c78d3'
        subscription='subscribed'/>
    </pubsub>
  </iq>
```

Quellcode 4-5: XMPP Subscribe-Antwort [81]

#### 4.2.6.3.2 Nachrichtenformate

XMPP wurde entwickelt, um verhältnismäßig kleine Einheiten von XML-Daten (**Textdaten**) zwischen zwei Netzwerkteilnehmern auszutauschen. Aber über Erweiterungen können auch **Binärdaten** übertragen werden. Dadurch wird auch die Übertragung von Dateien möglich. Ziel von XMPP war es, ein Protokoll zu entwickeln, das in der Basis so allgemein wie möglich gehalten wird, anstelle speziell nur für eine Anwendung passend zu sein. [80]

#### 4.2.6.3.3 Quality of Service

Durch die Verwendung von TCP besteht die Möglichkeit, dass Nachrichten nicht ankommen. Die Basisversion von XMPP beinhaltet kein Verfahren eines Stream-Managements. Jedoch besteht eine Erweiterung, die dies übernimmt. Dadurch kann eine Bestätigung (Acknowledgement) über den Erhalt einer Nachricht angefordert werden, sofern die Erweiterung aktiviert ist. [80]

#### 4.2.6.3.4 Sicherheit

XMPP unterstützt eine einfache Authentifizierung über **SASL** (Simple Authentication and Security Layer) [85] und die **Verschlüsselung über TSL** [80].

#### 4.2.6.3.5 Sonstiges

Was XMPP maßgeblich von anderen Protokollen unterscheidet, ist die Möglichkeit zur Erweiterung des Protokolls ja nach Anwendungsfall.

#### 4.2.6.4 Verwendungsmöglichkeiten

##### 4.2.6.4.1 Hardware

Es werden keine besonderen Anforderungen an die Hardware gestellt, weshalb XMPP von allen gängigen Computern unterstützt wird. MCUs mit kleinem Speicher und wenig Ressourcen könnten unter Umständen Probleme machen.

##### 4.2.6.4.2 Programmiersprachen

Für XMPP gibt es Implementierungen für alle gängigen Programmiersprachen. Eine detaillierte Übersicht findet man auf der offiziellen Seite von XMPP [86].

##### 4.2.6.4.3 Plattform-Dienste

Unter den untersuchten IoT-Plattformdiensten unterstützt nur Exosite die Kommunikation durch XMPP [59].

#### 4.2.6.5 Aufbau des Protokolls und Overhead

Da bei XMPP der Header in Textform ist und sowohl beim Aufbau einer Verbindung (Stream) als auch bei den Nachrichten selbst diverse Attribute übertragen werden müssen, hat dieses Protokoll einen **hohen** Overhead. Der Aufbau einer iq-Nachricht kann unter 4.2.6.3.1 eingesehen werden. Hier ein Beispiel für ein XMPP-Stream.

```
<stream:stream
    from='juliet@im.example.com'
    to='im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'>
  <message>
    <body>foo</body>
  </message>
</stream:stream>
```

Quellcode 4-6: XMPP Stream [80]

#### 4.2.6.6 Energiesparendes Protokolldesign

Da XMPP wie auch HTTP für die Übertragung von Textdaten optimiert ist, hat es auch nicht den Anspruch, besonders energiesparend zu sein. Das energiesparende Protokolldesign kann hier nur mit **gering** festgesetzt werden.

#### 4.2.6.7 Verbreitung/Support/Foren

Eine Google-Suche „XMPP“ ergab 4.150.000 Ergebnisse und bei Stackoverflow.com 14.883 Treffer. Trotz dieser verhältnismäßig großen Anzahl an Resultaten bei der Suche erkennt man bei näherer Betrachtung, dass sich die wenigsten auf das Internet der Dinge beziehen. Aus diesem Grund kann hier nur von einer **geringen** Verbreitung gesprochen werden.

#### 4.2.6.8 Zukunftsfähigkeit

Obwohl XMPP vielseitig unterstützt wird und vor allem über sehr viele Features verfügt, kann diesem Protokoll für IoT nur eine **geringe** Zukunftsfähigkeit zugesprochen werden. Immer mehr Daten sollen immer schneller von immer kleineren Einheiten übertragen werden. Da ist XMPP dann nicht die beste Wahl.

#### 4.2.6.9 Vor- und Nachteile des Protokolls bezogen auf ihr Einsatzgebiet

**Vorteile**

- Implementierungen für Programmiersprachen
- Sehr viele Features durch Erweiterungen

**Nachteile**

- Hoher Protokolloverhead
- Optimiert für Instant Messenger-Anwendungsfälle

#### 4.2.6.10 Zusammenfassung und Fazit

Da XMPP sowohl sehr einfache als auch unbeschränkt komplexe Nachrichten unterstützt und über viele Erweiterungen verfügt, ist das Protokoll vielfältig einsetzbar. Die globale Adressierung und Abhängigkeit von einem Server zur Kommunikation können jedoch je nach Anwendung hinderlich sein. Im Bereich IoT findet es nur sehr wenig Anwendung.

#### 4.2.6.11 Übersichtstabelle

| | |
|---|---|
| **Beschreibung** | Vielfältig einsetzbares und erweiterbares Protokoll |
| **Kommunikation** | |
| **Kommunikationsarchitektur** | Request/Response<br>Publish/Subscribe |
| **Kommunikationsform** | Synchron, asynchron, Pipelining |
| **Kommunikationswege und Adressierung** | Unicast, Multicast, Broadcast<br>Über globale Adressen |
| **Kommunikationsrichtung** | Unidirektional und bidirektional |
| **Funktionsumfang** | |
| **Nachrichtenarten** | message<br>presence<br>iq<br>get, set, result, error |
| **Nachrichtenformate** | Text und Binär |
| **Quality of Service** | Acknowledgement |
| **Sicherheit** | Benutzerauthentifizierung über SASL<br>TLS |
| **Sonstiges** | Erweiterung des Protokolls |
| **Verwendungsmöglichkeiten** | |
| **Hardware** | Alle gängigen Computer und MCUs mit WiFI-Chip |
| **Programmiersprachen** | Alle gängigen Programmiersprachen |
| **Plattform-Dienste** | Exosite |
| **Aufbau des Protokolls und Overhead** | hoch |
| **Energiesparendes Protokolldesign** | gering |
| **Verbreitung/Support/Foren** | gering |
| **Zukunftsfähigkeit** | gering |
| **Zentraler Verwaltungsserver/Broker** | ja |
| **Vor- und Nachteile des Protokolls** | Vorteile<br>• Implementierungen für Programmier-sprachen<br>• Sehr viele Features durch Erweiterun-gen<br>Nachteile<br>• Hoher Protokolloverhead<br>• Optimiert für Instant Messenger-Anwendungsfälle |

Tabelle 4-8: Übersichtstabelle XMPP

### 4.2.7 DDS

#### 4.2.7.1 Beschreibung

DDS (Data Distribution System) wurde ursprünglich für den Bereich Luftfahrt und Verteidigung entwickelt. Es ist seit vielen Jahren und bei weiträumigen und dezentralen Echtzeit-Systemen im Einsatz. Mit dem Aufkommen von IoT und dem Industrial IoT (IIoT) findet es immer häufiger Verwendung. Vor allem die Möglichkeit zur asynchronen Echtzeitübertragung großer Datenmengen in einem dezentralen Netz macht das DDS aus. [87] Aufgrund der Komplexität und der dezentralen Datenverwaltung ist DDS jedoch nicht einfach nur ein Protokoll sondern vielmehr auch Middleware. Da diese vor allem auch im industriellen Umfeld (explizit im amerikanischen Raum) häufig verwendet wird, erfolgt eine bestmögliche Untersuchung anhand der festgelegten Kriterien.

**Kurzbeschreibung: Dezentrale M2M Middleware zur Echtzeitübertragung**

#### 4.2.7.2 Kommunikation

##### 4.2.7.2.1 Kommunikationsarchitektur

Bei DDS handelt es sich um ein datenzentriertes **Publish/Subscribe**-Modell, welches also im Gegensatz zu MQTT ohne Message Broker auskommt. Die Daten werden direkt zwischen den einzelnen Geräten ausgetauscht [87]. Dies ermöglicht den Austausch von mehreren Tausend Nachrichten pro Sekunde. Dies macht das Protokoll vor allem für Anwendungen im medizinischen Bereich sowie bei der Entwicklung von selbstfahrenden Fahrzeugen [88] ??? (Wort fehlt).

##### 4.2.7.2.2 Kommunikationsform

Im Publish/Subscribe-Modell von DDS handelt es sich um eine **asynchrone** Kommunikationsform beim Nachrichtenaustausch. [87]

##### 4.2.7.2.3 Kommunikationswege und Adressierung

Über DDS werden die Nachrichten nicht an einen zentralen Server gesendet sondern einem Topic zugehörig in das Netzwerk publiziert. Interessierte Subscriber können dann auf diese Daten zugreifen. Die Adressierung erfolgt damit nicht direkt an oder von einem anderen Gerät und auch die Kommunikation ist eher als **Multicast** zu betrachten. Die Kommunikation findet in einem Global Data Space (GDS) über verschiedene **Topics** statt. Diese GDS wird von allen Teilnehmern, den sogenannten Domain Participants, gleichzeitig verwendet. Diese Domain Participants teilen sich in Publisher und Subscriber auf. Die GDS eines bestimmten Netzwerks wird auch als DDS-Domäne bezeichnet und über eine eindeutige Zahl (unique ID) identifiziert. In der Abbildung 6-4 ist eine beispielhafte GDS mit Participants aufgezeigt. [89]

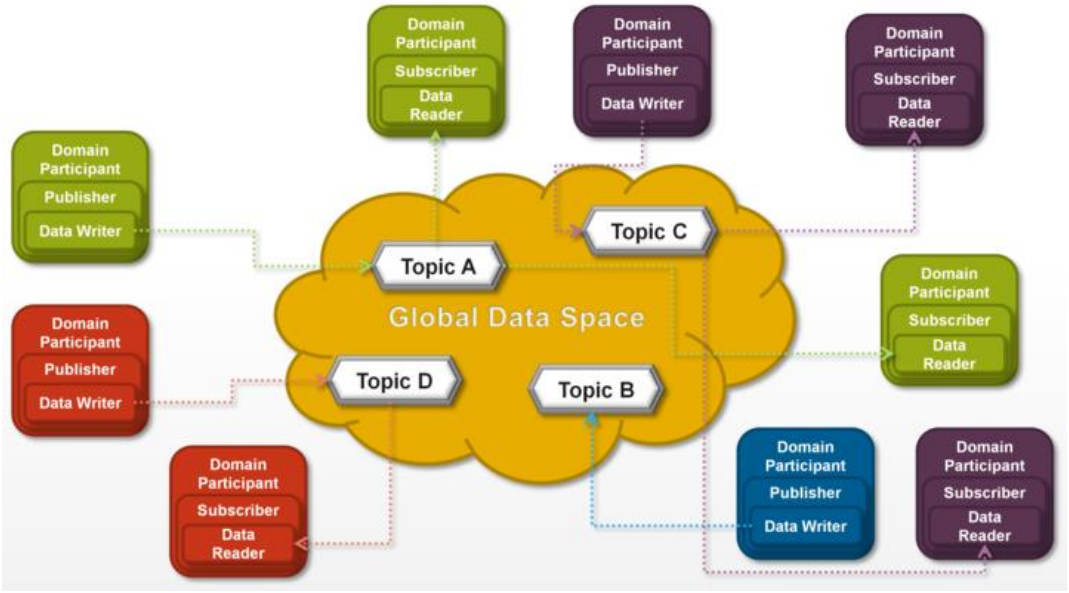


Abbildung 4-6: Beispielhafte Systemkonfiguration mit allen relevanten DDS-Entitäten [89]

#### 4.2.7.2.4 Kommunikationsrichtung

Da DDS auch mit der eigenen Selbstorganisation beschäftigt ist, ist der Datenaustausch nicht nur auf die publizierten Nachrichten beschränkt und dadurch bidirektional. Versucht man DDS auf die Protokolleigenschaften der Nachrichten zu reduzieren, wäre hier die Kommunikationsrichtung unidirektional. [87]

### 4.2.7.3 Funktionsumfang

#### 4.2.7.3.1 Nachrichtenarten

Wie DDS die Kommunikation aufbaut und wartet, übersteigt das Kriterium der Nachrichtenarten für die Untersuchung von Protokollen. Die beiden grundliegenden Nachrichtenarten Publish und Subscribe ergeben sich aus der Architektur.

#### 4.2.7.3.2 Nachrichtenformate

Bei DDS sind die Topics nicht nur Identifikationsnamen, sondern geben gleichzeitig auch das Datenformat für die über dieses Topic zu versendenden Nachrichten. Diese Struktur entspricht einem Objekt und kann verschiedene Eigenschaften primitiver Datentypen enthalten. Dateiübertragung ist mit dem DDS Core nicht möglich. Über sogenannte Topic Keys können z.B. innerhalb des Topics „Temperature“ die verschiedenen Sensoren identifiziert werden. [90]

```
struct Temperature {
    float data;
    unsigned long sensorId; //Topic key
}
```

Quellcode 4-7: DDS Nachrichtenformat-Struktur [90]

#### 4.2.7.3.3 Quality of Service

DDS verfügt über eine Vielzahl von QoS-Mechanismen. Die wichtigsten werden im Folgenden kurz vorgestellt.

**Durability**

Hier wird festgelegt, ob und inwieweit bereits veröffentlichte Daten durch die Publisher gespeichert werden, um sie auch für eventuell später dem Netzwerk beigetretene Subscriber zur Verfügung zu stellen. [89]

**Lifespan**

Über Lifespan wird festgelegt, wie lange Daten nach dem Veröffentlichen noch gültig sind. Dadurch kann verhindert werden, dass veraltete Daten an die Subscriber ausgeliefert werden. [89]

**Time-Based Filter**

Da Daten auch häufiger publiziert werden können, als ein Subscriber diese verarbeiten kann, besteht unter dieser Einstellung die Möglichkeit, festzulegen, in welchen zeitlichen Abständen der Data Reader Nachrichten zu einem bestimmten Topic erhalten möchte. [89]

**Reliability**

Mit dieser Einstellung kann festgelegt werden, ob der geschriebene Datensatz bei allen Readern angekommen sein muss. Falls Datensätze eines Topics verloren gehen, werden diese von der Middleware-Schicht wiederhergestellt. [89]

**Resource Limits**

Über Resource Limits kann u. a. festgelegt werden, wie viele Dateninstanzen die Middleware für jeden im Netzwerk enthaltenen Data Writer bzw. Data Reader speichern kann. [89]

#### 4.2.7.3.4 Sicherheit

DDS verfügt über diverse Plugins (z.B. RTI Connext DDS Secure) mit einer Vielzahl an einstellbaren Sicherheitsmaßnahmen. Darunter Authentifizierung, Verschlüsselung und Zugriffskontrolle. [91]

#### 4.2.7.3.5 Sonstiges

Ähnlich wie bei XMPP gibt es auch für DDS Erweiterungen, mit denen unter anderem auch ein Request/Response-Verfahren möglich ist. [92]

#### 4.2.7.4 Verwendungsmöglichkeiten

##### 4.2.7.4.1 Hardware

Grundsätzlich stellt DDS keine besonderen Anforderungen an die Hardware. Für ressourcenbeschränkte MCUs ist DDS in der Core-Version nicht geeignet. Jedoch gibt es von der Firma RTI (Real Time Innovations) das Produkt Connext DDS Micro, welches speziell für diesen Einsatz entwickelt wurde und weitere Plugins für die Verwendung über eine eingeschränkte Bandbreite. [93]

##### 4.2.7.4.2 Programmiersprachen

Für DDS existiert für alle gängigen Programmiersprachen eine Library. Eine genaue Auflistung kann auf der Spezifikationsseite der Object Management Group (OMG) eingesehen werden. [94]

##### 4.2.7.4.3 Plattform-Dienste

Keine der untersuchten IoT-Plattformdienste unterstützt DDS als Schnittstelle zur Kommunikation. [59]

#### 4.2.7.5 Aufbau des Protokolls und Overhead

Der genaue Aufbau einer DDS-Nachricht konnte trotz weitreichender Recherche nicht gefunden werden. Jedoch wird an mehreren Stellen von einem geringen Overhead gesprochen. Aus diesem Grund wird der Overhead für DDS als **gering** eingestuft. [93]

#### 4.2.7.6 Energiesparendes Protokolldesign

Damit bei DDS trotz eines fehlenden Message-Brokers ein Publish/Subscribe-Verfahren möglich ist, ist jede Einheit an der Verwaltung der DDS-Domäne beteiligt. Dadurch sind die Endgeräte nicht nur reine Sender und Empfänger, sondern auch am Betrieb des Netzwerks beteiligt. DDS hat den Anspruch, Daten mit hohem Durchsatz und geringer Latenz möglichst in Echtzeit zur Verfügung zu stellen, nicht ressourcensparend. Somit kann die Eigenschaft als energiesparendes Protokolldesign nur als **gering** festgehalten werden. [89]

#### 4.2.7.7 Verbreitung/Support/Foren

Eine Google-Suche „data distribution system“ (für „DDS“ gab es zu viele irrelevante Treffer) ergab 39.100.000 Ergebnisse und bei Stackoverflow.com mit dem Suchbegriff „DDS“ 1.527 Treffer. DDS wird vor allem in Amerika im Bereich IIoT (Industrial Internet of Things) verwendet. Es gibt viele Dokumentationen zur Verwendung. Mit Blick auf das gesamte IoT können die Verbreitung und der Support als **mittel** eingestuft werden.

#### 4.2.7.8 Zukunftsfähigkeit

Bei der Zukunftsfähigkeit muss man bei DDS genauer unterscheiden. Für den Consumer Bereich ist es aufgrund der hohen Komplexität eher weniger geeignet, wobei hingegen industrielle Anwendungen gerade auf die vielfältigen QoS-Möglichkeiten setzen können. Aus diesem Grund besitzt DDS eine **hohe** Zukunftsfähigkeit, die jedoch auch durch die Entwicklung von Konkurrenzprodukten beeinflusst wird.

#### 4.2.7.9 Vor- und Nachteile des Protokolls bezogen auf ihr Einsatzgebiet

**Vorteile**

- geringer Overhead
- hoher Datendurchsatz
- geringe Latenz
- viele QoS-Möglichkeiten

**Nachteile**

- nicht optimal für eine Request/Response-Anwendung
- keine Dateiübertragung

#### 4.2.7.10 Zusammenfassung und Fazit

DDS überzeugt durch seine Möglichkeit, eine Publish/Subscribe-Architektur ohne einen zentralen Nachrichtenverwalter zu ermöglichen. Dies führt zu einem hohen Datendurchsatz und ist damit vor allem für den industriellen Bereich interessant, bei dem viele Daten in Echtzeit ausgetauscht werden sollen. Ebenso die vielfältigen Möglichkeiten im QoS sprechen vor allem für die Verwendung mit sicherheitssensiblen Anwendungen.

#### 4.2.7.11 Übersichtstabelle

| | |
|---|---|
| **Beschreibung** | Dezentrale M2M Middleware zur Echtzeitübertragung |
| **Kommunikation** | |
| **Kommunikationsarchitektur** | Publish/Subscribe |
| **Kommunikationsform** | Asynchron |
| **Kommunikationswege und Adressierung** | Multicast über Topics (und Topic Keys) |
| **Kommunikationsrichtung** | Unidirektional |
| **Funktionsumfang** | |
| **Nachrichtenarten** | Publish, Subscribe |
| **Nachrichtenformate** | Objekte mit primitiven Datentypen |
| **Quality of Service** | (Auszug)<br>Durability, Lifespan, Time-Based Filter, Reliability, Resource Limits |
| **Sicherheit** | Authentifizierung, Verschlüsselung und Zugriffskontrolle |
| **Sonstiges** | Erweiterung auf ein Request/Response-Verfahren |
| **Verwendungsmöglichkeiten** | |
| **Hardware** | Alle gängigen Computer und MCUs (mit RTI Connext DDS Micro) |
| **Programmiersprachen** | Alle gängigen Programmiersprachen |
| **Plattform-Dienste** | Keine |
| **Aufbau des Protokolls und Overhead** | gering |
| **Energiesparendes Protokolldesign** | gering |
| **Verbreitung/Support/Foren** | mittel |
| **Zukunftsfähigkeit** | hoch |
| **Zentraler Verwaltungsserver/Broker** | nein |
| **Vor- und Nachteile des Protokolls** | Vorteile<br>• geringer Overhead<br>• hoher Datendurchsatz<br>• geringe Latenz<br>• viele QoS Möglichkeiten<br>Nachteile<br>• nicht optimal für eine Request/Response-Anwendung<br>• keine Dateiübertragung |

Tabelle 4-9: Übersichtstabelle DDS

## 4.3 Vergleichende Übersicht

| Protokoll | MQTT | HTTP/1.1 | HTTP/2 |
|---|---|---|---|
| **Beschreibung** | **Leichtgewichtiges M2M-Protokoll, offener Standard, geringe CPU-Ressourcen** | Kurzbeschreibung: Weit verbreitetes und vielseitig unterstütztes Protokoll | Erheblich verbessertes HTTP-Protokoll mit besserem Datenaustausch |
| **Kommunikation** | | | |
| **Kommunikationsarchitektur** | Publish/Subscribe | Request/Response | Request/Response + Push |
| **Kommunikationsform** | Synchron | Synchron, pipelining | Synchron, asynchron, pipelining |
| **Kommunikationswege und Adressierung** | Unicast/Multicast über Topics | Unicast über URL | Unicast über URL |
| **Kommunikationsrichtung** | Unidirektional | Bidirektional | bidirektional |
| **Funktionsumfang** | | | |
| **Nachrichtenarten** | CONNECT, CONNACK, PUBACK, PUBREC, PUBREL, PUBCOMP, UBSCRIBE, SUBACK, UNSUBSCRIBE, UNSUBACK, PINGREQ, PINGRESP,<br>DISCONNECT | OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE, CONNECT | OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE, CONNECT, PUSH |
| **Nachrichtenformate** | Text und Binär | Text und Binär | Text und Binär |
| **Quality of Service** | QoS 0: At most once<br>QoS 1: At least once<br>QoS 2: Exactly once | – | (Multiplexen) |
| **Sicherheit** | Authentifizierung durch Benutzername und Passwort, Verschlüsselung über TLS | Authentifizierung über Benutzername und Passwort sowie eine verschlüsselte Übertragung der Benutzerdaten, Verschlüsselung über TLS | Authentifizierung über Benutzername und Passwort sowie eine verschlüsselte Übertragung der Benutzerdaten, Verschlüsselung über TLS |
| **Sonstiges** | Eine MQTT-SN Version speziell für Sensornetzwerke verfügbar | – | Header Compression<br>Streams<br>Priorisierung |
| **Verwendungsmöglichkeiten** | | | |
| **Hardware** | Alle gängigen Computer und MCUs mit WiFI-Chip | Alle gängigen Computer und MCUs mit WiFI-Chip | Alle gängigen Computer und MCUs mit WiFI-Chip |
| **Programmiersprachen** | Alle gängigen Programmiersprachen | Alle gängigen Programmiersprachen | Alle gängigen Programmiersprachen |
| **Plattform-Dienste** | flowthings.io<br>Carriots<br>Beebotte<br>IBM Watson IoT<br>AWS IoT | ThingSpeak<br>data.sparkfun.com<br>Runabove IoT Lab<br>flowthings.io Carriots Ubidots<br>GroveStreams<br>Exosite<br>Beebotte<br>MODE<br>Initial State<br>Temboo<br>Oracle Cloud<br>IBM Watson IoT<br>Azure IoT Hub<br>AWS IoT<br>Google Cloud Platform | ThingSpeak<br>data.sparkfun.com<br>Runabove IoT Lab<br>flowthings.io Carriots Ubidots<br>GroveStreams<br>Exosite<br>Beebotte<br>MODE<br>Initial State<br>Temboo<br>Oracle Cloud<br>IBM Watson IoT<br>Azure IoT Hub<br>AWS IoT<br>Google Cloud Platform<br>*Grau, da nur per Downgrade auf HTTP/ 1.1 |
| **Aufbau des Protokolls und Overhead** | gering | hoch | mittel |
| **Energiesparendes Protokolldesign** | hoch | mittel | mittel |
| **Verbreitung/Support/Foren** | hoch | hoch | gering |
| **Zukunftsfähigkeit** | hoch | gering | mittel |
| **Zentraler Verwaltungsserver/Broker** | ja | nein | nein |
| **Vor- und Nachteile des Protokolls** | Vorteile:<br>• Sehr geringer Overhead<br>• Hoch skalierbar<br>• Viele IoT-Protokollfeatures<br>• Für ressourcenbeschränke Geräte geeignet<br>• Clientimplementierungen sind für alle gängigen Programmiersprachen verfügbar<br>Nachteile:<br>• Reine Request/Response-Architekturen sind mit MQTT nur mit zusätzlichem Aufwand umsetzbar | Vorteile<br>• Nahezu uneingeschränkte Unterstützung<br>• Einfacher Protokollaufbau<br>Nachteile<br>• Gr. Overhead bei jedem Request<br>• Head-of-line Blocking<br>• Keine Push-Nachrichten | **Vorteile**<br>• Verbesserter Header<br>• Streams bei der Übertragung großer Datenmengen<br>**Nachteile**<br>• Derzeit noch geringe Verbreitung<br>• Nicht für IoT optimiert |

Tabelle 4-10: Gesamtübersicht – MQTT – HTTP/1.1 – HTTP/2

| Protokoll | CoAP | XMPP | DDS |
|---|---|---|---|
| **Beschreibung** | Leichtgewichtiges M2M-Protokoll mit der einfachen Möglichkeit der Überführung in HTML | Vielfältig einsetzbares und erweiterbares Protokoll | Dezentrale M2M Middleware zur Echtzeitübertragung |
| **Kommunikation** | | | |
| **Kommunikationsarchitektur** | Request/Response<br>(Publish/Subscribe) | Request/Response<br>Publish/Subscribe | Publish/Subscribe |
| **Kommunikationsform** | Synchron und Asynchron | Synchron, asynchron, Pipelining | Asynchron |
| **Kommunikationswege und Adressierung** | Unicast und Multicast über URI | Unicast, Multicast, Broadcast<br>Über globale Adressen | Multicast über Topics (und Topic Keys) |
| **Kommunikationsrichtung** | Bidirektional | Unidirektional und bidirektional | Unidirektional |
| **Funktionsumfang** | | | |
| **Nachrichtenarten** | POST, GET, PUT, DELETE | message<br>presence<br>iq<br>get, set, result, error | Publish, Subscribe |
| **Nachrichtenformate** | Text und Binär | Text und Binär | Objekte mit primitiven Datentypen |
| **Quality of Service** | Confirmable Message<br>NonConfirmable Message | Acknowledgement | (Auszug)<br>Durability, Lifespan, Time-Based Filter, Reliability, Resource Limits |
| **Sicherheit** | Authentifizierung über Benutzername und Passwort und DTSL (sharedKey, PublicKey, Certificate) | Benutzerauthentifizierung über SASL<br>TLS | Authentifizierung, Verschlüsselung und Zugriffskontrolle |
| **Sonstiges** | Discovery<br>Piggybacked Response | Erweiterung des Protokolls | Erweiterung auf ein Request/Response Verfahren |
| **Verwendungsmöglichkeiten** | | | |
| **Hardware** | Alle gängigen Computer und MCUs mit WiFI-Chip | Alle gängigen Computer und MCUs mit WiFI-Chip | Alle gängigen Computer und MCUs (mit RTI Connext DDS Micro) |
| **Programmiersprachen** | Alle gängigen Programmiersprachen | Alle gängigen Programmiersprachen | Alle gängigen Programmiersprachen |
| **Plattform-Dienste** | Exosite | Exosite | Keine |
| **Aufbau des Protokolls und Overhead** | gering | hoch | gering |
| **Energiesparendes Protokolldesign** | hoch | gering | gering |
| **Verbreitung/Support/Foren** | gering | gering | mittel |
| **Zukunftsfähigkeit** | hoch | gering | hoch |
| **Zentraler Verwaltungsserver/Broker** | nein | ja | nein |
| **Vor- und Nachteile des Protokolls** | **Vorteile**<br>• Offener Standard<br>• Web-Interoperabilität (leicht in HTTP umzuwandeln)<br>• Geringer Overhead<br>**Nachteile**<br>• Primär für die direkte Kommunikation zwischen zwei Geräten konzipiert<br>• Kein richtiges Publish/Subscribe | Vorteile<br>• Implementierungen für Programmiersprachen<br>• Sehr viele Features durch Erweiterungen<br>Nachteile<br>• Hoher Protokolloverhead<br>• Optimiert für Instant Messenger-Anwendungsfälle | Vorteile<br>• geringer Overhead<br>• hoher Datendurchsatz<br>• geringe Latenz<br>• viele QoS-Möglichkeiten<br>Nachteile<br>• nicht optimal für eine Request/Response-Anwendung<br>• keine Dateiübertragung |

Tabelle 4-11: Gesamtübersicht – CoAP – XMPP – DDS

## 4.4 Fazit

Von Beginn an war klar, dass es nicht ein bestimmtes Protokoll geben wird, das dann als das IoT-Anwendungsschicht-Protokoll schlechthin definiert werden kann. Hinzu kommt, dass auch keines der untersuchten Protokolle ein reines IoT-Protokoll ist, welches nur dafür entwickelt wurde. Alle bestehen schon seit Jahren für die unterschiedlichsten Anwendungen. Zuvor gab es die Sensornetzwerke mit ihren Protokollen und das Internet mit ihren Protokollen. Nun wird aus beiden Richtungen versucht, auch den anderen Bereich abzudecken. Teilweise gelingt das direkt, in anderen Fällen besteht eine Interoperabilität zwischen den Protokollen. Ein ganz wichtiges Element in Bezug auf die Protokolle des IoT sind derzeit Gateways in jeglicher Form, über die die unterschiedlichen Netze und auch Protokolle verbunden werden können. Zudem bestehen ganze Geräte, die sowohl mehrere Protokolle verwenden, als auch die zentrale Steuerung übernehmen. Als Beispiel für ein solches Zusammenwirken wird im kommenden Kapitel mit der Verwendung eines Raspberry Pi und einer speziellen Software ein Versuchsaufbau entwickelt.

Bei den verschiedenen Protokollen ist aufgefallen, dass sie teilweise unterschiedliche Bereiche des OSI-Schichtenmodells abdecken. Manche Protokolle sind mehr Middleware als Protokoll. Aus dem Grund können viele Protokolle nicht wirklich vergleichend betrachtet werden, sondern sind vielmehr erweiternd. Im Folgenden werden die untersuchten Protokolle noch einmal kurz zusammengefasst beschrieben.

MQTT wird als leichtgewichtiges Protokoll, das vielfältig unterstützt wird, zukünftig eine wichtige Rolle im Bereich der Publish/Subscribe-Architektur spielen. Durch seinen geringen Overhead und seine einfache Verwendung ist es optimal für Netzwerke, die über eingeschränkte Ressourcen verfügen.

HTTP/1.1 genießt aufgrund seiner bisher schon vielfältigen Verwendung eine weitreichende Unterstützung sämtlicher Geräte. Mit der Anwendung des REST-Architekturstils auf HTTP/1.1 wird es auch weiterhin ein wichtiges Protokoll im Bereich Request/Response-Architektur sein.

Der große Nachteil von HTTP/1.1, nämlich sein sehr großen Overhead, wurde bei HTTP/2 verbessert, was vor allem beim Austausch von mehreren Nachrichten über eine Verbindung durch das mögliche Weglassen von Head-Attributen erkennbar ist. HTTP/2 wird nativ aber noch von keiner IoT-Plattform unterstützt.

Am Beispiel von REST ist schnell zu erkennen, wie sehr Protokolle verschiedener Sichten, Middleware und Architekturstile missverständlich verwendet werden. REST ist kein Protokoll sondern nur ein Architekturstil zur Verwendung eines Protokolls und kann auf verschiedene Protokolle angewendet werden. Sofern von einer REST API oder einfach nur von REST gesprochen wird, ist in der Regel die Unterstützung von HTTP/1.1 im RESTful-Design gemeint.

Mit CoAP kann eine einfache Verbindung eines Sensornetzwerks mit dem Internet vorgenommen werden. Dies liegt vor allem an der einfachen Möglichkeit der Umwandlung in HTTP. Damit wird der geringe Overhead im Sensornetzwerk genutzt und über die Umwandlung in HTTP steht eine breite Unterstützung durch andere Geräte zur Verfügung. Für diese Verbindung kommt aber auch wieder ein Gateway ins Spiel, welches in diesem Fall aber nur geringer Ressourcen bedarf.

Die große Flexibilität von XMPP und die unzähligen Erweiterungen machen XMPP sehr interessant für vielfältige Anwendungen. Jedoch kann hier die Abhängigkeit von einem Namensserver zum einen hinderlich sein, zum anderen durch global eindeutige Adressen auch vorteilhat. Im Bereich IoT wird XMPP nur sehr wenig verwendet.

DDS kann nicht als einfaches Protokoll gesehen werden, da es viele administrative Funktionalitäten abdeckt. Die Möglichkeit der Verwendung einer Publish/Subscribe-Architektur ohne zentrale Einheit beinhaltet zwar sehr einfach austauschbare Nachrichten, ist jedoch mit einem zusätzlichen Verwaltungsaufwand verbunden. Als datenzentrische Lösung mit den ausgefeilten QoS-Einstellungen wird es zwar im Consumerbereich wohl keine Verwendung finden, jedoch im Bereich der Industrie, wo der Datendurchsatz und eine geringe Latenz wichtiger sind als ressourcensparendes Verhalten. Hier steht DDS, was größtenteils in den USA verwendet wird, in Konkurrenz zu OPC UA, was sich im europäischen Raum als das IIoT-Protokoll/Middleware durchzusetzen scheint.

Das anfangs auch erwähnte Google Cloud Pub/Sub ist ebenfalls eine Middleware bzw. PaaS und SaaS (Software as a Service), was auf die Bedürfnisse des IoT eingeht und neben einem Publish/Subscribe-Verfahren auch eine Request/Response-Anbindung unterstützt.

Letztendlich sollte das zu verwendende Protokoll anhand der Anwendung ausgewählt werden. Wichtig dabei ist: welche Geräte werden verwendet, welche Übertragungstechniken werden genutzt, benötige ich eine zentrale Verwaltung?

Bei dieser Arbeit geht es um die Anwendungsschicht-Protokolle des Internets der Dinge und um das Verständnis der verschiedenen Architekturen. Aus diesem Grund wird im kommenden Kapitel die praktische Verwendung von einem Publish/Subscribe und einem Request/Response-Protokoll veranschaulicht. Als Publish/Subscribe-Protokoll kommt das immer mehr verwendete MQTT-Protokoll zum Einsatz, als Request/Response-Protokoll HTTP/1.1, sofern möglich, im RESTful Design, da dies derzeit die Standardschnittstelle zu sämtlichen PaaS-Angeboten ist.

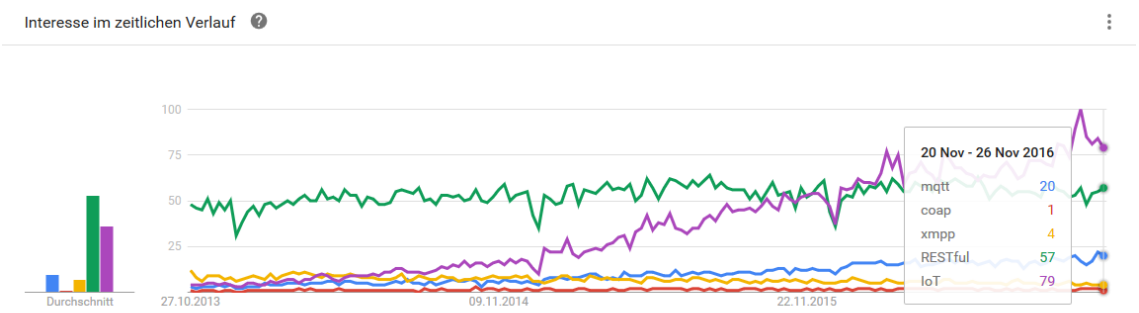


Abbildung 4-7: Häufigkeit der Google-Anfragen nach Protokollen [95]

Das aufkommende Interesse an IoT und dessen Protokollen kann in Abbildung 4-7 anhand der Analyse mit Google Trends eingesehen werden. Diese Abbildung unterstreicht auch nochmal die Wahl zur Verwendung der Protokolle MQTT und HTTP/1.1 im RESTful-Design für das praktische Beispiel im Vergleich zu CoAP und XMPP.

# 5 Beispielprojekt

Um die Funktionalität und das Zusammenspiel der Protokolle etwas zu veranschaulichen, werden in diesem Kapitel zum einen verschiedene Hard- und Softwarekomponenten vorgestellt und zum anderen an praktischen Beispielen erläutert. Nicht alle vorgestellten Geräte kommen zum Einsatz, da sie entweder an ein eigenständiges System angebunden oder noch zu neu sind.

## 5.1 Grundlagen

### 5.1.1 Hardware

#### 5.1.1.1 ESPRUINO Pico

Der Espruino Pico ist der leistungsstärkere und kleinere Nachfolger des Espruino. Er ist ein Microcontroller, der ohne spezielle Software programmiert werden kann. Nach dem Anschließen per USB hat der Benutzer die Möglichkeit, diesen über ein Webinterface in JavaScript zu programmieren. Zusätzlich steht eine Möglichkeit der Programmierung über einen grafischen Editor zur Verfügung. Somit ist der Espruino Pico auch optimal für Einsteiger geeignet, die direkt nach dem Anschließen ihre ersten Schritte gehen können.

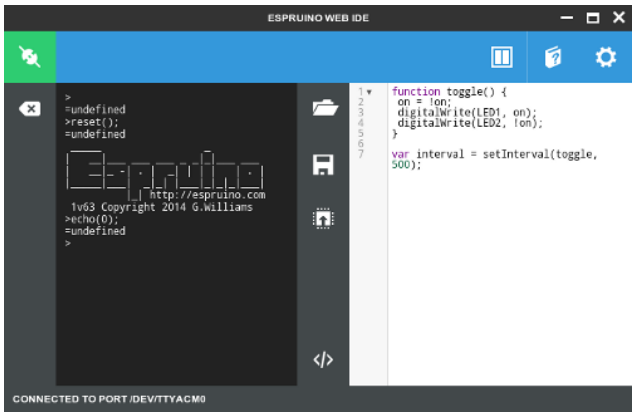


Abbildung 5-1: Espruino Web IDE [96]

Der Espruino Pico verfügt über keinen WiFi-Adapter, weshalb er im Originalzustand nicht für das Internet der Dinge geeignet ist. Wie in Abbildung 5-2 zu sehen, besteht jedoch die Möglichkeit, ihn um einen WiFi-Adapter zu erweitern.

Abbildung 5-2: Espruino Pico mit aufgelötetem ESP8266 (ESP-01) [96]

#### 5.1.1.2 ESP8266/ESP-01

Das in Abbildung 5-2 gezeigte Modul ist ein ESP8266 nach der Bauart ESP-01. Dieser dient jedoch nicht nur als WiFi-Adapter für den Espruino, sondern kann auch unabhängig von weiterer Hardware als MCU verwendet werden. In der Ausführung als ESP-01 kann jedoch nicht das gesamte Potential des ESP8266 genutzt werden. So hat der ESP-01 nur zwei GPIO-Pins zur Verfügung und es fehlen viele andere Anschlüsse, die der ESP8266 normalerweise bereitstellt. Aufgrund seiner kleinen Bauart und der bereits angelöteten Pins wurde dieser Chip jedoch für den Versuchsaufbau ausgewählt. Zur einfachen Programmierung bestehen Erweiterungen für die Arduino IDE sowie unzählige Tutorials und Forendiskussionen als Support. Zudem konnten vielfältige Modifikationen des Chips gefunden werden, mit denen mehr von den ursprünglichen Funktionen genutzt werden können. Was jedoch berücksichtigt werden muss: Im Gegensatz zum Espruino wird nicht nur eine Software zur Entwicklung benötigt, sondern auch eine speziell dafür ausgelegte Platine zum Aufspielen des Programms. Eine solche Platine wird im praktischen Teil entwickelt und vorgestellt.

#### 5.1.1.3 ESP8266/ESP-12E Development Board

Neben der Ausführung des ESP8266 als ESP-01 bestehen noch weitere Modelle, die die volle Funktionalität des Chips unterstützen. Diese Module sind jedoch nicht mehr mit Steckverbindungen versehen, sondern zum direkten Auflöten auf eine Platine gedacht. Um Entwicklern jedoch die Möglichkeit zu geben, Programme für den Chip zu entwickeln, bestehen sogenannte Development Boards. Bei diesem Projekt wurde ein ESP-12E Development Board verwendet. Dieses sorgt nicht nur für ein einfaches Anschließen der Pins, sondern übernimmt auch die korrekte Ansteuerung des Moduls beim Aufspielen von einem Programm. Neben den vielen Anschlüssen hat man durch einen Analog/Digital-Wandler die Möglichkeit zum Verwenden eines Drehreglers, z.B. zum Steuern der Helligkeit einer Glühbirne.

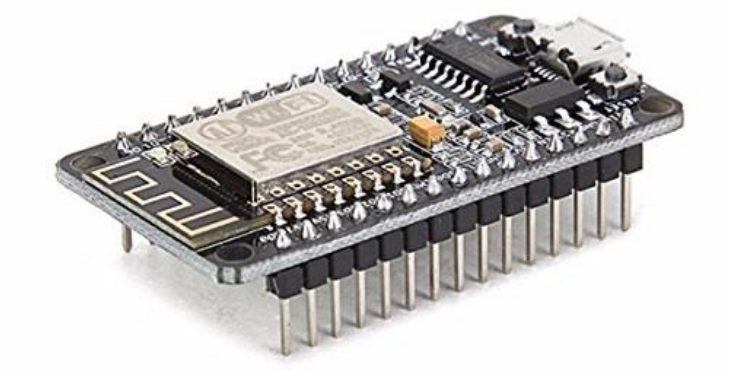

Abbildung 5-3: ESP8266/ESP-12E Development Board [97]

#### 5.1.1.4 ESP32

Bereits während der Untersuchung der Übertragungstechniken in Kapitel 3.1.8 war zu erkennen, dass die Zukunft des Internets der Dinge in der Verwendung von wenigen unterschiedlichen Techniken und deren Zusammenführung über Gateways liegen wird. Gegen Ende dieser Arbeit (Oktober 2016) wurde ein neues Modell des ESP veröffentlicht: Das ESP32-Modul. Die Besonderheiten dieses Moduls sind vor allem die beiden Übertragungschips (WiFi und Bluetooth), die beliebig ein- und ausgeschalten werden können, sowie ein Digital/Analog-Wandler. Dadurch sind unzählige Anwendungen möglich, die zuvor mehrere, miteinander verbundene Module vorausgesetzt haben. Der ESP32 ist bereits verfügbar und lag gegen Ende der Thesis auch vor.

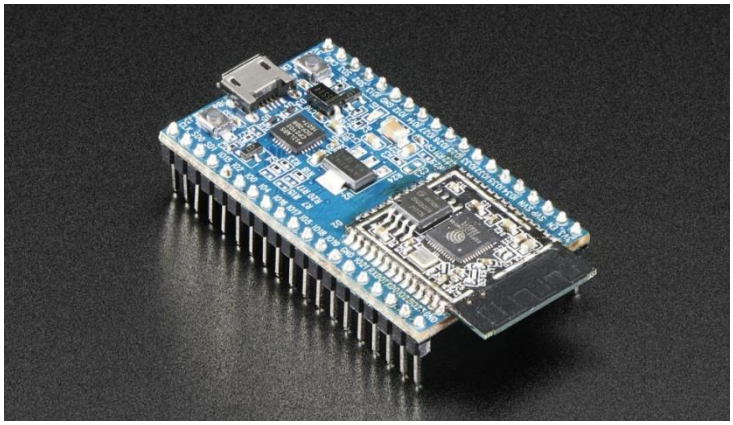

Abbildung 5-4: ESP32 Development Board (schwarzes Teil = ESP32 Modul) [98]

#### 5.1.1.5 Raspberry Pi

Der Raspberry Pi ist ein sogenannter Einplatinencomputer mit einem ARM-Mikroprozessor. Er kam Anfang 2012 auf den Markt. Nicht nur seit dem Aufkommen des Internets der Dinge erfreut er sich wachsender Beliebtheit. Es gibt unzählige Zusatzelemente und Betriebssystem-Images mit vorinstallierter Software für die verschiedensten Anwendungen. Auch für das IoT bietet der Raspberry Pi optimale Voraussetzungen. Im Beispielprojekt ist er ein zentrales Element der Kommunikation. Das verwendete Modell ist der Raspberry Pi 3 Model B.

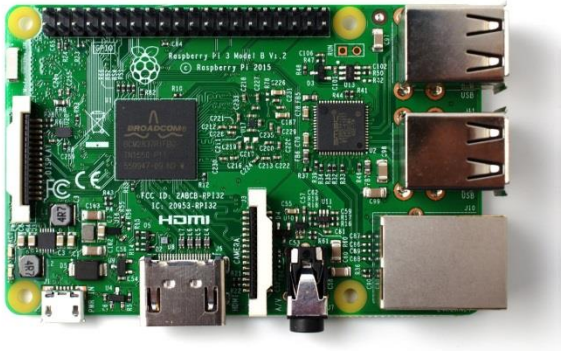

Abbildung 5-5: Raspberry Pi 3 Model B [99]

#### 5.1.1.6 Philips Hue

Das Philips Hue System besteht aus einer Bridge und diversen Lichterzeugern (z.B. Glühbirnen oder Light Strips) und Geräten, die damit verbunden werden können. Die Lichterzeuger können in verschiedenen Farben leuchten und sind in Farbe und Helligkeit stufenlos einstellbar. Das Philips Hue System nutzt dabei ZigBee als Übertragungstechnik. Über die Bridge, die an einen Router angeschlossen wird, können die Birnen über eine Vielzahl verfügbarer Smartphone-Apps direkt aus dem eigenen WLAN heraus oder über eine Cloud-Verbindung von überall gesteuert werden. Für die Ansteuerung der Lampen verfügt die Bridge zudem über ein API, weshalb das System auch im Beispielprojekt zum Einsatz kommt.

Abbildung 5-6: Philips Hue Starter Set [100]

Während der Thesis wurden dem Autor im Rahmen eines Produkttests drei Philips Hue Light & Color Birnen der 2nd Generation zur Verfügung gestellt. Dies ermöglichte erste Tests in der Kommunikation zwischen ESP8266 und anderen Geräten (in diesem Fall der Hue Bridge).

#### 5.1.1.7 Beacons

Beacons oder auch iBeacons sind kleine Bluetooth-Module, die über Bluetooth LE immer wieder Signale absenden. Diese können bei erweiterten Beacons verschiedene Informationen enthalten. So kann z.B. ein am Fenster eingesetzter Beacon den Neigungswinkel des Fensters sowie den Öffnungszustand enthalten. Andere Beacons, die z.B. nur zur Identifizierung verwendet werden, senden lediglich ihre ID und damit auch über ihre Signalstärke die Entfernung zum Empfänger [101]. Durch die Verwendung von BTLE können diese Daten auch von mehreren Geräten empfangen werden, ohne sich mit dem Beacon zu paaren [21]. Ein weiter Vorteil ist der sehr geringe Stromverbraucht eines solchen Beacons, da er keinerlei Verbindung aufrechterhalten muss. Sendet er zudem in großen Abständen, können die Batterien mehrere Jahre halten.

### 5.1.1.8 iPhone als Beacon

Da die Beacons noch verhältnismäßig teuer sind, besteht auch die Möglichkeit, sein iPhone (mit Bluetooth 4.0) als Beacon zu verwenden. Mit der Open Source App GemTotSDK, die mithilfe von Xcode installiert werden kann, kann man einen sendenden Beacon simulieren. Somit hat man eine kostengünstige Möglichkeit, seinen Beacon-Empfänger zu testen. [102]

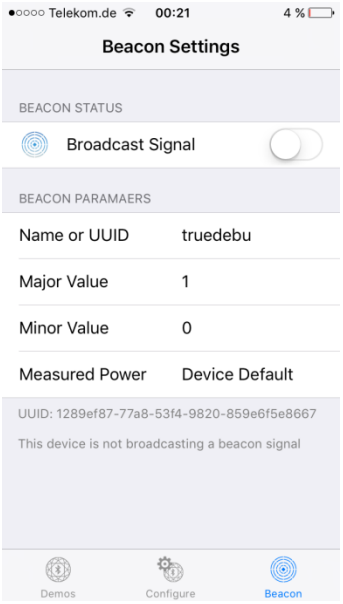


Abbildung 5-7: GemTotSDK App zur Beacon Simulation

### 5.1.1.9 Happy Bubbles

Anhand der übertragenen Signalstärke kann man mithilfe von mehreren Empfängern, die optimal platziert sind, die Position des Beacons im Raum bestimmen. Der Heavy Bubble Presence Detector ermöglicht diese Funktionalität. Interessant bei diesem Detektor ist vor allem für diese Arbeit, dass er die empfangenen Daten der Beacons an einen MQTT-Broker weiterleitet. Diese Daten können dann von jedem beliebigen Gerät oder Software abonniert werden. Heavy Bubbles ist ein Open Source Projekt und wurde mithilfe eines NodeMCU (auch ein ESP8266 Development Board) in Verbindung mit einem Blietoot LE Modul umgesetzt. Mit der Einführung des ESP32, welcher einen integrierten BTLE-Chip besitzt, kann dieselbe Funktionalität auch mit einem Modul und ohne zusätzliche Hardware umgesetzt werden. [103]

### 5.1.1.10 AWS IoT Button

Seit kurzem sind auch in Deutschland die sogenannten „Dash Buttons“ von Amazon erhältlich, mit denen auf Knopfdruck ein bestimmtes Produkt bestellt und meist schon am folgenden Tag geliefert werden. Die Einführung der Dash Buttons war in Deutschland zunächst verzögert, da zum Kaufvertrag die Zustimmung des Kunden nötig ist. Da der Kunde nach der Bestellbenachrichtigung die Möglichkeit zur Stornierung hat, konnten die Buttons im August 2016 eingeführt werden. [104]

Bei der Recherche nach IoT-Geräten und ihrer Verwendung mit Plattformen wurde auf Amazon auch ein AWS IoT (Dash) Button angeboten, der selbst „programmiert“ werden kann. Dabei handelt es sich jedoch nicht um einen programmierbaren IoT-Button, sondern einen Dash Button, der lediglich seine Seriennummer, Batteriespannung und die Art des Klicks (einfach, doppelt, lange) an die AWS IoT sendet (Quellcode 5-1). Dort kann dann online festgelegt werden, welche Aktion ausgelöst werden soll. Damit ist man mit dem AWS IoT Button an einen Server gebunden. [105]

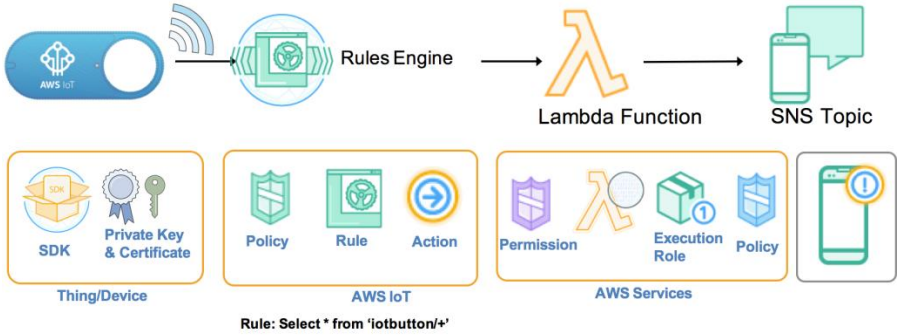


Abbildung 5-8: AWS IoT Button Ablauf [105]

```
{
   "serialNumber": "GXXXXXXXXXXXXXXXXX",
   "batteryVoltage": "mV",
   "clickType": "SINGLE | DOUBLE | LONG"
}
```

Quellcode 5-1: JSON Payload des AWS IoT Buttons [105]

Der AWS IoT Button ist derzeit nur in den USA erhältlich. Zur Untersuchung wurde er aber bestellt und über einen Vermittler nach Deutschland gesendet, wo er laut Amazon Angaben dennoch funktioniert. [105]

> *„Der Taster funktioniert überall, wo ein WLAN (2,4 GHz) vorhanden ist.“ [105]*

Da der Button, wie auch die normalen Dash Buttons, keine zusätzliche Hardware benötigt, war festzustellen, wie er sich mit dem Server verbindet. Wie in der Anleitung beschrieben, befand sich der Button nach fünfsekündigem Gedrückthalten in einem AP (Access Point) Modus. Nach dem Verbinden mit dem neuen WLAN konnte über die Adresse „192.168.0.1/index.html“ die Konfigurationsseite abgerufen werden. Neben den Zugangsdaten des zu verwendenden WLANs muss hier der Verbindungsserver angegeben werden und sowohl ein Zertifikat als auch eine Private-Key Datei hochgeladen werden (Abbildung 5-9).
Somit verbindet sich der Button nach der Aktivierung erst mit dem WLAN und setzt dann seine Meldung an den Server ab, bevor er sich selbst wieder abschaltet.

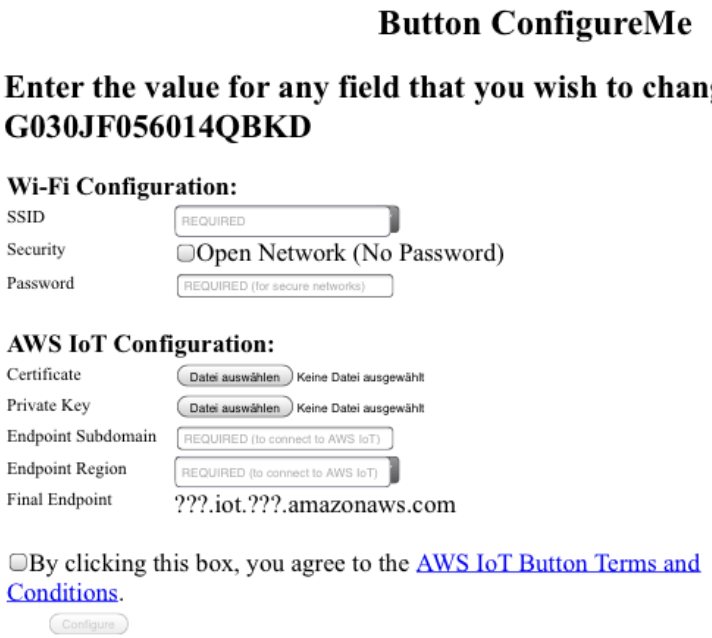


Abbildung 5-9: AWS IoT Button Konfiguration

Dieses Verhalten ermöglicht es, dass die Batterie des Buttons laut Amazon-Angaben für 1.000 Klicks ausreichend ist. [105]

### 5.1.2 Software

In diesem Kapitel werden die verwendete Software und die Einrichtung aller benötigten Komponenten beschrieben. Grundsätzlich gilt in diesem Fall, dass die Software immer auf dem neusten Stand sein sollte oder dass man nichts verändert, wenn alles läuft. Während der Erstellung dieser Arbeit kam es durch ein Update des Betriebssystems am Entwicklungs-Laptop zu einer erheblichen Verzögerung, da die Ursache nicht sofort ersichtlich war.

#### 5.1.2.1 Arduino DIE

Die große Beliebtheit des ESP8266 hat auch dazu geführt, dass die notwendigen Erweiterungen entwickelt wurden, damit er auch über die Arduino DIE programmiert werden kann.

##### 5.1.2.1.1 Basis

Die Arduino IDE kann direkt über die Herstellerseite bezogen werden (https://www.arduino.cc/en/Main/Software). In dieser Version ist es jedoch noch nicht möglich, den ESP8266 zu verwenden. Auf GitHub besteht die Möglichkeit, eine bereits für den ESP8266 eingerichtete Version zu beziehen [106]. Aufgrund von während dieser Arbeit aufgetretenen Problemen empfiehlt es sich jedoch, die Entwicklungsumgebung direkt vom Hersteller zu beziehen und die Unterstützung des ESP8266 wie folgt selbst zu installieren.

#### 5.1.2.1.2 Boardverwalter

Nach der erfolgreichen Installation der Arduino IDE kann die Unterstützung zusätzlicher Boards über den Boardverwalter ermöglicht werden. Für den ESP8266 erfolgt dies über folgende Schritte.

„Arduino->Einstellungen“

Dort trägt man unter „Zusätzliche Boardverwalter URLs“ folgenden Pfad ein:

```
http://arduino.esp8266.com/stable/package_esp8266com_index.json
```


Abbildung 5-10: Arduino IDE – zusätzliche Boardverwalter URLs

Anschließend kann über „Werkzeuge->Board->Boardverwalter“ der Boardverwalter aufgerufen werden. Der Eintrag „ESP82“ in das Suchfeld sorgt für ein schnelles Finden des richtigen Moduls. Hier besteht (anders als in der Abbildung) die Möglichkeit, einen „Installieren“-Button zu betätigen.

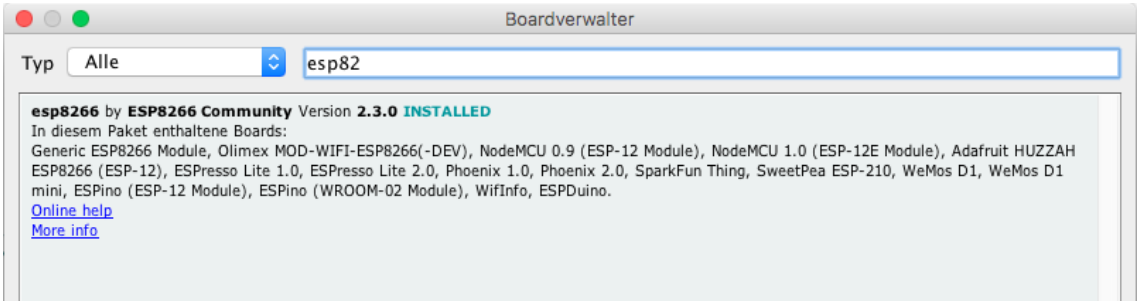


Abbildung 5-11: Arduino IDE – Boardverwalter

Damit ist die IDE bereit für die Entwicklung für und das Aufspielen auf den ESP8266.

#### 5.1.2.1.3 Bibliotheken

Im weiteren Verlauf werden zusätzliche Bibliotheken benötigt. Während für das Request/Response-Verfahren HTTP nativ unterstützt wird, muss für Publish/Subscribe über MQTT eine Bibliothek eingebunden werden. Dies ist möglich über den Bibliotheksverwalter.

Aufzurufen über: „Sketch->Bibliotheken einbinden->Bibliotheken verwalten“

Hier sollte ebenfalls das Suchfeld verwendet werden. Nach der Eingabe „PubSub“ erscheinen zwei verwendbare Bibliotheken. In dieser Arbeit wurde die Bibliothek von Nick O’Leary verwendet (Abbildung 5-12).

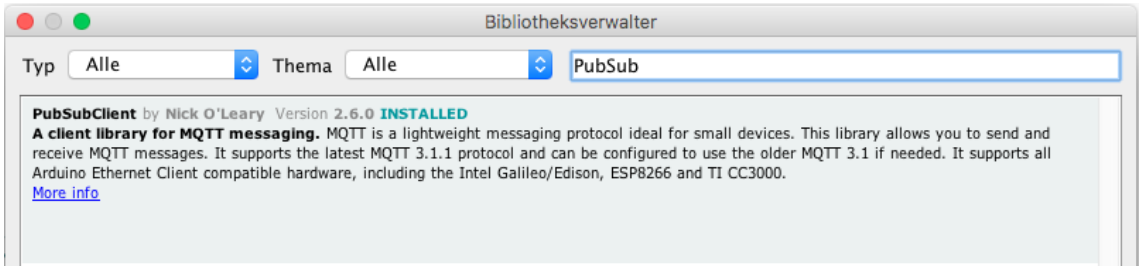


Abbildung 5-12: Arduino IDE – Bibliotheksverwalter PubSub

Wichtig sei dabei zu erwähnen, dass mit der Bibliothek Publishen nur in der QoS-Stufe 0 und Subscriben in den Stufen 0 und 1 möglich ist.

Für den Temperaturfühler DHT11 wird ebenfalls eine Bibliothek benötigt. In dieser Arbeit wurde die von Adafruit angebotene gewählt. Zu finden ist diese ebenfalls im Bibliotheksverwalter und mit der Suchanfrage „DHT11“.

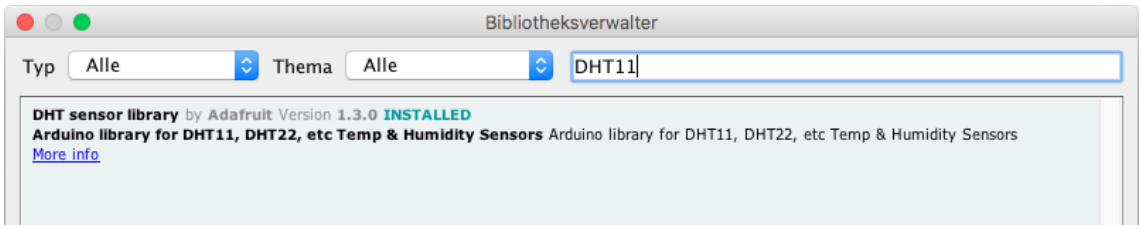


Abbildung 5-13: Arduino IDE – Bibliotheksverwalter DHT11

#### 5.1.2.2 Raspbian

Für den Raspberry Pi wurde das Betriebssystem Raspbian in der Version Jessie vom Juni 2016 verwendet. Dazu muss das Image vom Server geladen und auf eine SD-Karte aufgespielt werden (z.B. über DD). Die aktuelle Version ist immer zu finden unter: https://www.raspberrypi.org/downloads/raspbian/

#### 5.1.2.3 Node.js

Node.js ist eine Runtime, die JavaScript-Code serverseitig ausführt. Vor allem bei Einplatinencomputern kommt Node.js vielfältig zum Einsatz. Auch bei dieser Arbeit basieren einige Module darauf. Bei Raspbian Jessie ist Node.js bereits installiert, jedoch nicht in der aktuellen und benötigten Version. Deshalb muss die neuste Version installiert werden. Dies erfolgt über folgende Befehle auf dem Raspberry Pi über eine SSH Verbindung.

```
sudo npm cache clean -f
sudo npm install -g n
sudo n stable
```

#### 5.1.2.4 Mosquitto MQTT-Broker

Damit der Raspberry Pi als MQTT-Broker verwendet werden kann, muss ein solcher Broker darauf installiert werden. Dies ist z.B. der Mosquitto MQTT-Broker, der auch in dieser Arbeit verwendet wurde. Die Installation erfolgt über folgende Schritte. [107]

```
wget http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key
sudo apt-key add mosquitto-repo.gpg.key
cd /etc/apt/sources.list.d/
sudo wget http://repo.mosquitto.org/debian/mosquitto-wheezy.list
sudo apt-get update
sudo apt-get install mosquitto
```

#### 5.1.2.5 Philips Hue API

Phillips bietet für ihre Bridge eine HTTP API im RESTful Design an, über die der komplette Funktionsumfang der Bridge genutzt werden kann. Vor der Verwendung muss man über ein Web-Interface eine Benutzerkennung generieren. Die genaue Anleitung ist auf den Seiten von Philips Hue zu finden. [108]

Über ein JSON-Objekt im Body können die einzelnen Attribute von Lampen und Gruppen gesetzt werden. REST konform findet über die Adresse nur die Adressierung statt und Informationen werden im Body per PUT übertragen.

| | |
|---|---|
| Address | `http://<bridge ip address>/api/<username>/lights/1/state` |
| Body | `{"bri":42}` |
| Method | `PUT` |

Tabelle 5-1: PUT Request an die Hue Bridge [108]

Über GET können entweder die Attribute einer einzelnen Birne oder ein Komplettstatus angefragt werden. Bei einem Komplettstatus wird die Übersicht im JSON-Format zurückgeliefert. [108]

| | |
|---|---|
| Address | `http://<bridge ip address>/api/<username>/lights/` |
| Body | |
| Method | `GET` |

Tabelle 5-2: GET Request an die Hue Bridge [108]

### 5.1.2.6 MQTT hue Bridge

Für Node.js gibt es eine MQTT Hue Bridge, die alle Topics, die Lichter betreffen, abonniert hat und diese dann übersetzt an die Bridge weitergibt. Im weiteren Verlauf wird diese Bridge zwar nicht mehr benötigt, ist aber an dieser Stelle dennoch erwähnenswert. Die Installationsanweisungen können unter https://github.com/dennisdegreef/mqtt-hue-bridge gefunden werden.

### 5.1.2.7 Crontab

Mit Crontab lassen sich Prozesse beim Starten von Raspberry Pi initiieren. Im Fall der MQTT Hue Bridge für Node.js hat dies jedoch nicht allein mit Crontab funktioniert. Aus diesem Grund wurde eine Lösung über „Forever“ gefunden, mit der dann alles wie gewünscht lief.

### 5.1.2.8 Forever

Durch die Verwendung von Forever konnte das Node.js-Script erfolgreich zum Start mit Raspbian eingerichtet werden. Über folgende Schritte wurde dies erreicht. [109]

```
sudo -i npm install forever -g
crontab -u pi -e
@reboot /usr/bin/sudo -u pi -H /usr/local/bin/forever start
/var/www/simple-server.js
```

Diese Einrichtung kann entfallen, sofern die MQTT Hue Bridge nicht verwendet werden soll. Da Node-RED über eine Erweiterung verfügt, mit dem die Hue-Birnen angesteuert werden können, ist die MQTT Hue Bridge obsolet.

### 5.1.2.9 Node-RED

Node-RED ist eine serverseitige Software, die es ermöglicht, über eine grafische Oberfläche Hardwaregeräte, APIs und andere Onlinedienste miteinander zu verknüpfen. Da sowohl HTTP und MQTT unterstützt werden, ist dies die optimale Software für die Verbindung der verschiedenen Geräte und Protokolle.

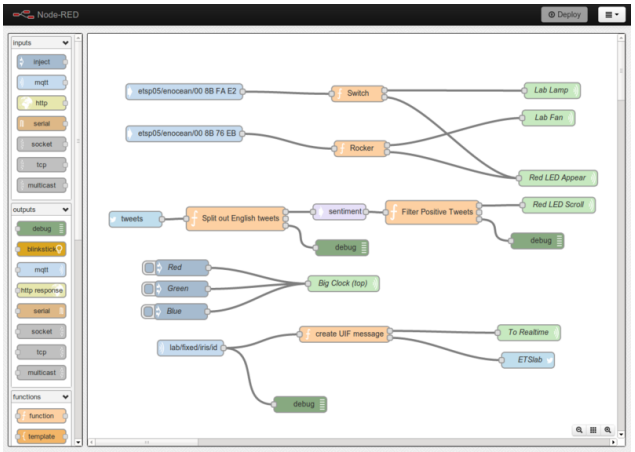


Abbildung 5-14: Node-RED – Flow Entwicklungsansicht [110]

Node-RED ist bei Raspbian Jessie bereits vorinstalliert, unterstützt in der installierten Version (Jessie Juni 2016) jedoch noch nicht alle Erweiterungen. Das Update kann über folgende Befehle durchgeführt werden.

```
sudo npm cache clean
sudo npm update -g --unsafe-perm node-red
```

Um Node-RED auch nach einem Reboot sofort verfügbar zu haben, kann es mit folgendem Befehl zum Autostart hinzugefügt werden. [111]

```
sudo systemctl enable nodered.service
```

Node-RED verfügt durch ein zusätzliches Modul auch über ein einfach zu erstellendes und verwaltendes User Interface.

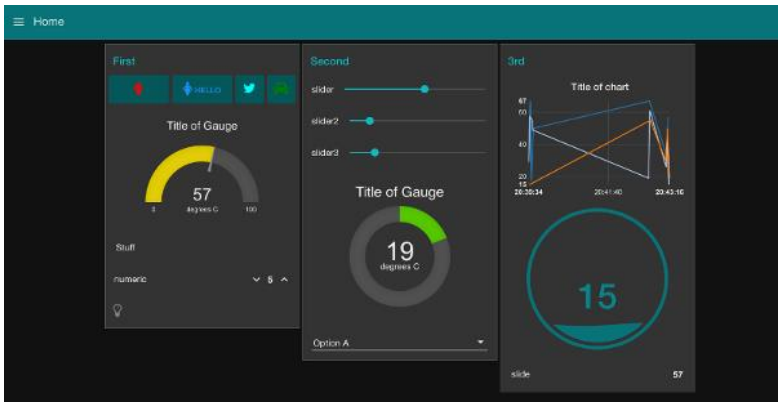


Abbildung 5-15: Node-RED – Dashboard [112]

Um die zusätzlichen Nodes für das User Interface (Dashboard) verwenden zu können, muss diese Erweiterung wie folgend beschrieben zu Node-RED hinzugefügt werden. [112]

```
cd ~/.node-red
npm install node-red-dashboard
```

Durch die Installation der MQTT hue Bridge können die Lampen einfach über eine MQTT-Message angesprochen werden. Für etwas mehr Komfort und Übersicht sorgt eine weitere Erweiterung für Node-RED, die wie folgt installiert werden kann.

```
cd ~/.node-red
npm install node-red-contrib-hueplus
```

Nach der Installation stehen in der Entwicklungsumgebung zwei neue Nodes zur Verfügung: „Hue-state“ und „Hue-set“. Einmal mit der Bridge verbunden, kann auf alle daran angemeldeten Lampen und erstellten Gruppen zugegriffen werden. [113]

#### 5.1.2.10 Fritzing

Aufgrund der sehr wenigen Bauteile, die anzuordnen waren, konnte auf eine spezielle Software und die Anfertigung von individuellen Platinen verzichtet werden. Für größere Platinen mit mehr Bauteilen oder zur Beschriftung bietet sich jedoch eine Software an, mit der dies einfach umzusetzen ist. Mit Fritzing können Verkabelungen auf einem Steckbrett vorbereitet und daraus komplexe Platinen hergestellt werden. Dabei besteht auch die Möglichkeit, sich aufgrund eines Schaltplans und der Positionierung der Bauteile automatisch eine Leiterplatine berechnen zu lassen.

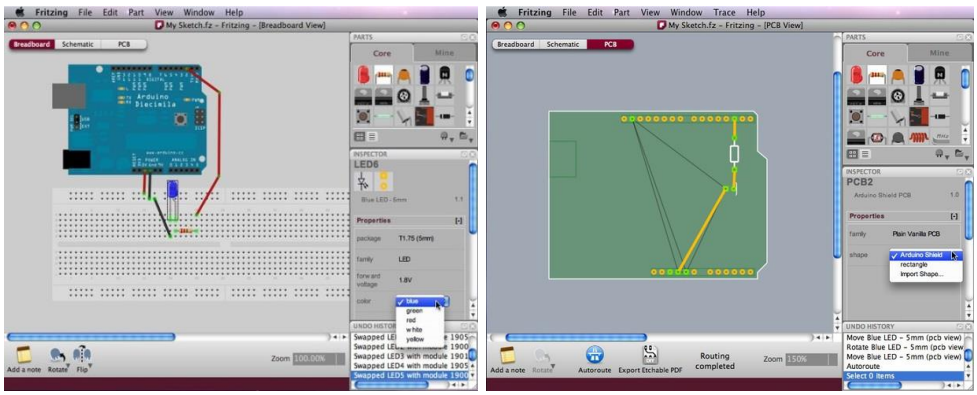

Abbildung 5-16: Fritzing Platinendesigner [114]

### 5.1.3 Einrichtung

Für einen weitreichenden Einblick in die Welt des IoT war es für diese Arbeit wichtig, nicht nur die gängigen Protokolle zu untersuchen, sondern sich auch mit der Hardware zu beschäftigen. Dies verlangte den Einsatz von Lötkolben, Lochrasterplatine, Messgerät und dem Auffrischen des Wissensstands im Bereich der Elektrotechnik. Die Arbeit mit dem ESP8266/ESP-01 war dabei besonders herausfordernd. Dies lag zum einen an einigen im Internet verbreiteten unvollständigen Plänen zur Pinbelegung und zum anderen an der Sensibilität der verwendeten Hard- und Software. Sobald alles jedoch korrekt eingerichtet ist und nicht mehr verändert wird (Betriebssystemupdate etc.), läuft alles problemlos. Aus diesem Grund ist darauf zu achten, dass alle Treiber installiert sind (für jede Art von verwendetem USB-Seriell-Adapter) und die Arduino IDE mit allen Libraries und Boards auf dem aktuellen Stand sind. Dies war im Rahmen der Untersuchung nicht immer der Fall und führte zu nicht nachvollziehbaren, weil nicht determinierten Problemen.

#### 5.1.3.1 ESP8266/ESP-01 Modul

Für die Einarbeitung in die Verwendung des ESP-01 Moduls musste sehr viel Zeit investiert werden. Erst wenn die wichtigsten Details klar sind, ist ein reibungsloses Arbeiten möglich. Dabei sei zu erwähnen, dass das ESP-01 Modul ursprünglich nur als WiFi-Modul für den Espruino Pico eingesetzt werden sollte. Nachdem sich aber herausgestellt hatte, dass das Modul auch eigenständig verwendet werden kann, gingen die Nachforschungen direkt in die Richtung nach der Umsetzung. Dabei stellten unvollständige Pläne zur Verkabelung eine kleine Hürde dar. Ein weiterer Punkt, der dazu führte, dass es ein paar

Tage dauerte, bis das erste Programm erfolgreich übertragen wurde, war der Umstand, dass selbst ein korrekt angeschlossenes Modul bei der Verkabelung über ein Steckbrett nicht funktionierte. Erst durch das Löten einer Platine konnte die Übertragung des Sketches auf das Modul erfolgen. Um hier eine zuverlässige Quelle zu bieten, werden die verschiedenen Anschlussmöglichkeiten aufgeführt.

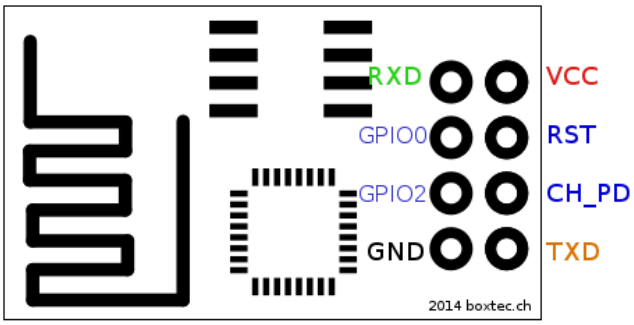


Abbildung 5-17: Pin-Belegung ESP-01 [115]

| **ESP-01** | **USB-Seriell-Adapter** |
|---|---|
| RX | TX |
| TX | RX |
| | VCC (3,3V) |
| CH_PD | VCC (3,3V) |
| GND | GND |
| GPIO0 | GND (um den Flashmodus zu aktivieren) |
| GPIO2 | |
| RST | DTR (Adapter setzt automatisch ein Reset) |

Tabelle 5-3: Anschluss über einen USB-Seriell-Adapter mit DTR Ausgang

| **ESP-01** | **USB-Seriell-Adapter** |
|---|---|
| RX | TX |
| TX | RX |
| VCC | VCC (3,3V) |
| CH_PD | VCC (3,3V) |
| GND | GND |
| GPIO0 | Über Taster an GND (um den Flashmodus zu aktivieren gedrückt halten und loslassen, sobald die Übertragung beginnt) |
| GPIO2 | |
| RST | DTR (Adapter setzt automatisch ein Reset) |

Tabelle 5-4: Anschluss über einen USB-Seriell-Adapter mit DTR Ausgang

| ESP-01 | USB-Seriell-Adapter |
|---|---|
| RX | TX |
| TX | RX |
| VCC | VCC (3,3V) |
| CH_PD | VCC (3,3V) |
| GND | GND |
| GPIO0 | GND (um den Flashmodus zu aktivieren) |
| GPIO2 | |
| RST | GND (nach dem Einstecken ist der Flashmodus sofort aktiv und das Modul erwartet Daten) |

Tabelle 5-5: Anschluss über einen USB-Seriell-Adapter

| ESP-01 | USB-Seriell-Adapter |
|---|---|
| RX | TX |
| TX | RX |
| VCC | VCC (3,3V) |
| CH_PD | VCC (3,3V) |
| GND | GND |
| GPIO0 | GND (um den Flashmodus zu aktivieren) |
| GPIO2 | |
| RST | Über Taster an GND<br>(vor der Übertragung kurz drücken) |

Tabelle 5-6: Anschluss über einen USB-Seriell-Adapter mit Taster für Reset

| ESP-01 | USB-Seriell-Adapter |
|---|---|
| RX | TX |
| TX | RX |
| VCC | VCC (3,3V) |
| CH_PD | VCC (3,3V) |
| GND | GND |
| GPIO0 | Über Taster an GND<br>(um den Flashmodus zu aktivieren gedrückt halten, dann den Taster an RST betätigen und loslassen, sobald die Übertragung beginnt) |
| GPIO2 | |
| RST | Über Taster an GND<br>(vor der Übertragung kurz drücken) |

Tabelle 5-7: Anschluss über einen USB-Seriell-Adapter mit Taster für Reset und Flashmodus

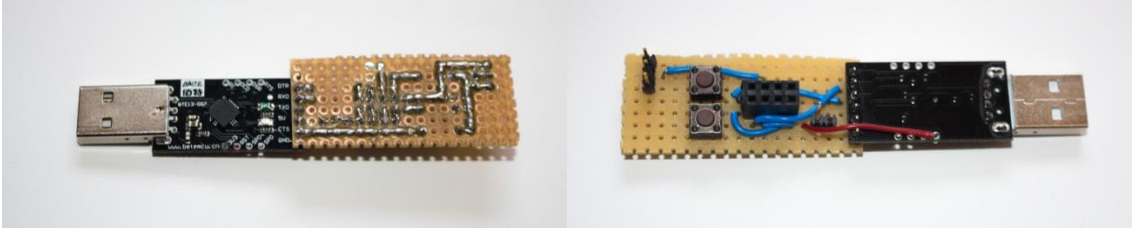

Abbildung 5-18: Erstes funktionsfähige Board zum Flashen des ESP-01

In Abbildung 5-18 ist das Board zu sehen, mit dem es das erste Mal funktioniert hat, ein Programm auf das Modul zu übertragen. Die Verkabelung entspricht Tabelle 5-5 (der DTR-Pin war zu dieser Zeit noch von keiner Anleitung her bekannt). Der obere Knopf im Bild ist der Knopf zur Einleitung des Flashmodus, der untere der Reset-Knopf. Der Adapter legt am Pin 5V an, weshalb er über das rote Kabel an 3,3V angeschlossen werden musste.

Im Rahmen der Arbeit wurden noch weitere USB-Seriell-Adapter besorgt. Dabei handelt es sich um eine Ausführung ohne DTR-Pin jedoch mit dem direkten Anschluss von 3,3V über einen Pin. Dies ermöglicht es, das Entwicklerboard auch unabhängig vom Adapter betreiben zu können.

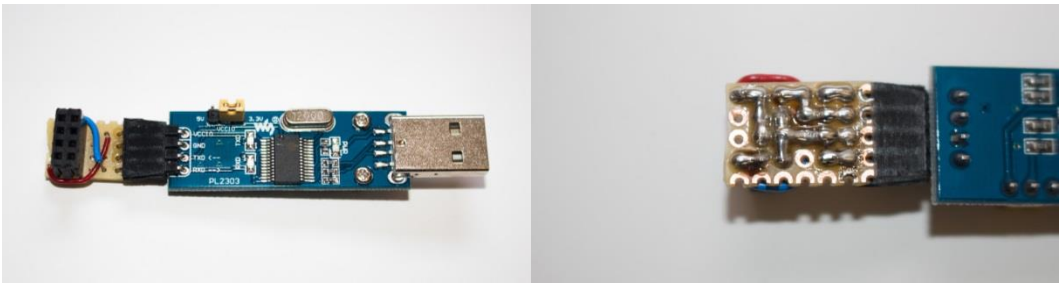

Abbildung 5-19: Adapter mit austauschbarem Board (Anschluss wie in Tabelle 5-3)

Beim Überlöten der Pins RST und GPIO2 mit einer durchgehenden Leiterbahn, muss darauf geachtet werden, dass diese Pins im Verbindungselement entfernt wurden. Bei diesem Adapter ist bei der ersten Benutzung auch zu prüfen, dass der Jumper den VCC-Pin auf 3,3V setzt. Zum Benutzen dieses Adapters, muss dieser lediglich zusammen mit dem ESP-01 Modul am USB-Port angeschlossen werden.

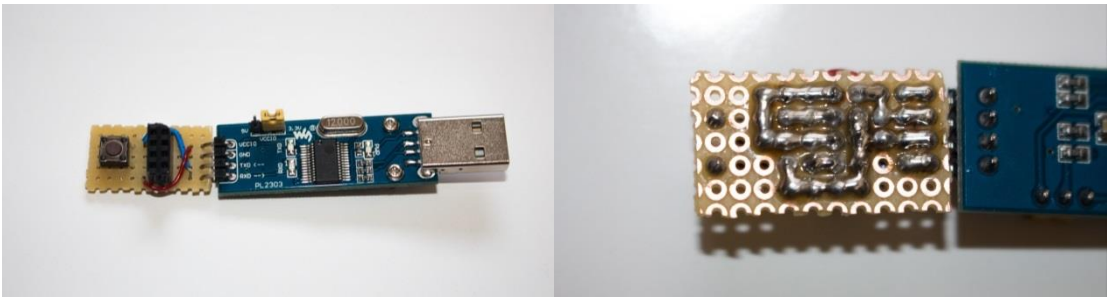

Abbildung 5-20: Adapter mit austauschbarem Board (Anschluss wie in Tabelle 5-4)

Dieser Adapter ist entstanden, bevor die Arduino IDE auf den neusten Stand gebracht wurde. Durch die alte Version wurde der Chipsatz des USB-Seriell-Adapters nicht korrekt verwendet. Teilweise begann der Upload, brach dann jedoch ab. Aus diesem Grund wurde er, mit der Vermutung auf Wackelkontakt (was zuvor beim Steckbrett der Fall war), am Board festgelötet und zudem ein Reset-Knopf angebracht, um sicher zu gehen, dass beim Verbinden des Adapters mit dem USB-Port nicht doch kurzzeitig ein falsches Signal anliegt. Dieser Adapter hat den Vorteil, dass ein erneutes Aufspielen ohne vorhergehendes Trennen vom Port durch Betätigen des Reset-Knopfes ermöglicht wird. Mit diesen drei Ausführungen ist es nun problemlos möglich, seinen Sketch auf den Microcontroller aufzuspielen.

#### 5.1.3.2 ESP8266/ESP-12E Development Board

Beim ESP8266/ESP-12E Development Board der Firma DOIT ist nur darauf zu achten, dass der Treiber für den integrierten USB-Seriell-Adapter (Chipsatz CH341) installiert wird. Dieser ist auf der auf dem Modul angegebenen Internetadresse für MAC und PC verfügbar. [116] Vor dem Aufspielen eines Sketches ist es wichtig, dass das richtige Board in der Arduino IDE gewühlt ist. Für dieses Modul sollte „NodeMCU 1.0 (ESP-12E Module)“ gewählt werden.

#### 5.1.3.3 Relais für 230V

Trotz der angegeben 5V, die das Relais verlangt, funktioniert es auch mit 3V als Steuerungssignal. Jedoch nur, solange die Spannung hoch genug ist. Danach schaltet das Relais nicht mehr. Das Relais bietet die Möglichkeit die 230V-Leitung so anzuschließen, dass sie entweder erst durch ein Signal geschlossen wird (NO=Normal Open) oder dass sie durch das Signal unterbrochen wird (NC=Normal Closed).

Abbildung 5-21: 5V/230V Relais

#### 5.1.3.4 Entfernungssensor

Der IR-Entfernungsmesser kann als Tür- oder Fenstersensor verwendet werden. Über den auf dem Modul befindlichen Drehwiderstand kann man die Empfindlichkeit einstellen.

Abbildung 5-22: IR-Entfernungsmesser

#### 5.1.3.5 Lichtsensor

Da der ESP-01 keinen Anschluss an einen Analog/Digital-Wandler besitzt, kann die Helligkeit nicht in verschiedenen Stufen gemessen werden. An diesem Lichtsensor-Modul kann über einen Drehwiderstand die Empfindlichkeitsstufe eingestellt werden, ab wann HIGH oder LOW am digitalen Ausgangspin anliegt.

Abbildung 5-23: Lichtsensor

### 5.1.3.6 DHT11 Temperatur- und Feuchtigkeitssensor

Im Gegensatz zu den anderen Sensoren überträgt dieser nicht ein einfaches HIGH oder LOW, sondern muss vom Microcontroller ausgelesen werden. Man kann die aktuelle Temperatur und Luftfeuchtigkeit auslesen. Bei diesem Sensor ist zudem zu beachten, dass seine typische Betriebsspannung 5V beträgt. Er funktioniert zwar auch mit 3V, jedoch häufen sich die Fehlerwerte mit dem Abnehmen der Batteriespannung. Deshalb ist bei der längeren Verwendung eine 5V-Spannungsquelle zu benutzen und einen Spannungsregler (z.B. LD33V) vor das WiFi-Modul zu schalten, da dieses nicht 5V kompatibel ist. Die ersten Versuche mit dem Sensor gelangen und auch die Messwerte waren zuverlässig. Sogar ein Senden aus dem Kühlschrank heraus war kein Problem. Jedoch als die Batteriespannung durch die Kälte sank, wurden keine fehlerfreien Werte mehr übertragen. Bei der Verwendung von neuen Bauteilen ist es deshalb immer ratsam, sich das Datenblatt anzusehen.


Abbildung 5-24: DHT11 Temperatur- und Feuchtigkeitssensor

### 5.1.3.7 ESP32 Development Board

Aufgrund der Spezifikationen und Entwicklung im Bereich IoT lässt sich vermuten, dass der ESP32 mindestens genauso beliebt werden wird, wie sein Vorgänger der ESP8266. Er schafft durch die Verbindung von WiFi und Bluetooth auf einer MCU viele neue Möglichkeiten. Die beiden Chips sind zudem einzeln aktivierbar. Das bereits vorgestellte Projekt „Heavy Bubbles“ könnte hier mit nur einem Modul umgesetzt werden. Erste Schritte in Richtung Bluetooth/WiFi-Gateway sollten an dieser Stelle erfolgen. Zum aktuellen Zeitpunkt (November 2016) besteht jedoch noch keine Unterstützung der Arduino IDE. Auf Anfrage bei Adafruit, die bereits die Arduino IDE Unterstützung für den ESP8266 realisiert haben, wurde mitgeteilt, dass bereits daran gearbeitet wird. Das vorliegende Development Board kann derzeit zwar noch nicht verwendet werden, verspricht aber eine Vielzahl an Anwendungsmöglichkeiten in der Zukunft.

## 5.2 Elemente des Beispielprojekts

### 5.2.1 Hue Color Matrix

Wie im vorangegangenen Teil bereits erwähnt, wurden mir im Rahmen eines Produkttests drei Philips Hue Birnen zur Verfügung gestellt. Diese konnten bei ersten Versuchen bereits mit dem ESP8266/ESP-12E Development Board angesteuert und die Helligkeit über einen Drehwiderstand eingestellt werden. Die einfache Steuerung bot sich auch vor der Einarbeitung in Node-RED an. Aus diesem Grund wurde eine Anfrage an Philips gestellt, ob sie weitere Birnen für ein Projekt innerhalb dieser Thesis zur Verfügung stellen können. Die Anfrage wurde kurz vor Ende dieser Arbeit positiv beantwortet. Für den Bau einer Farbmatrix unter Verwendung eines 4x4 IKEA-Regals lieferte Philips die dafür notwendigen 16 Hue Birnen der 3rd Generation. Die Idee war es, jede Birne einzeln ansprechen zu können, um dadurch Farbmuster zu erzeugen und die Lampe zudem über Sensoren mit der Umwelt zu interagieren.

#### 5.2.1.1 Bau

Zum Bau wurden unter anderem folgende Teile verwendet:

- 16 x Philips Hue White & Color 3rd Generation
- 1x Hue Bridge
- 16 x E27 Kunststofffassung zum Festschrauben
- 1x TP-Link TL-WR802N N300 WLAN Nano Router
- 1x Mehrfachsteckdosenleiste schmal mit USB
- 1x 147cm x 147 cm Acrylglas OPAL mit 35% Lichtdurchlässigkeit
- IKEA KALLAX Regal 4x4 in hochglanzweiß

Abbildung 5-25: Zusammenbau der Color Matrix

Abbildung 5-26: Zusammengebautes Regal ohne Acrylglas und Wandmontage

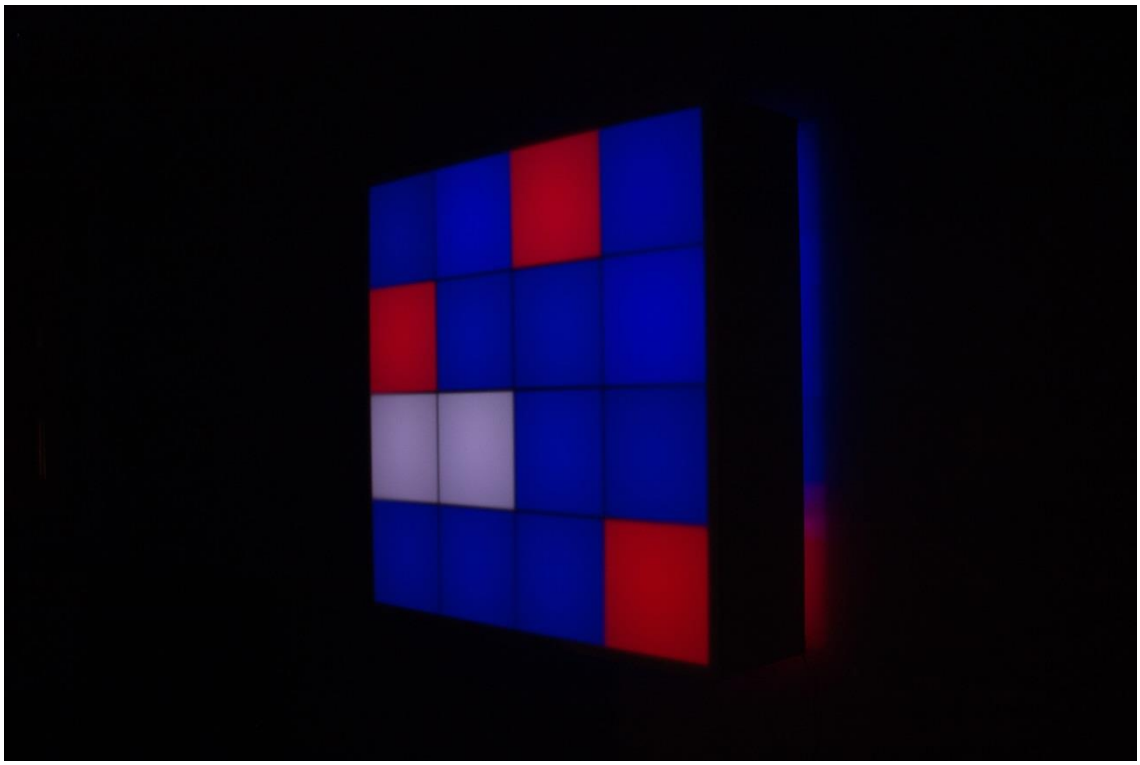

Abbildung 5-27: Fertige Color Matrix mit Abstand zur Wand montiert

### 5.2.1.2 Ansteuerung über Node-RED

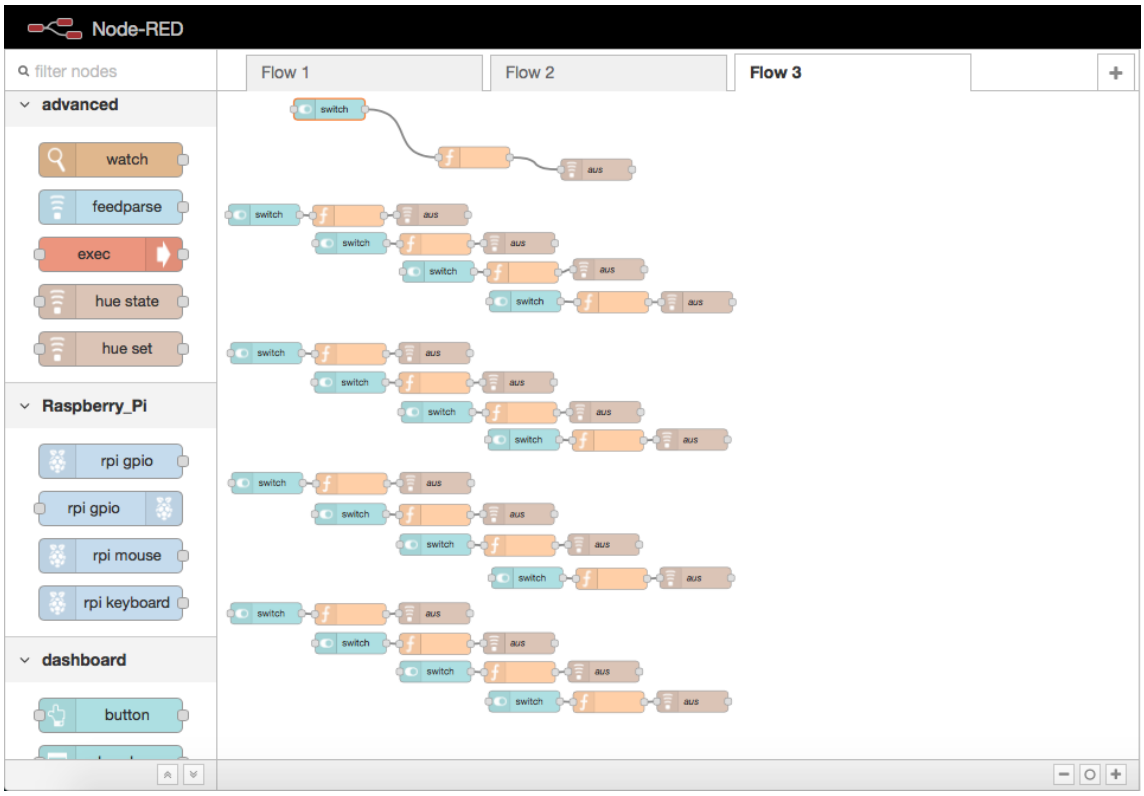


Abbildung 5-28: Noder-RED Flow für die Ansteuerung der einzelnen Birnen über die Web-Oberfläche

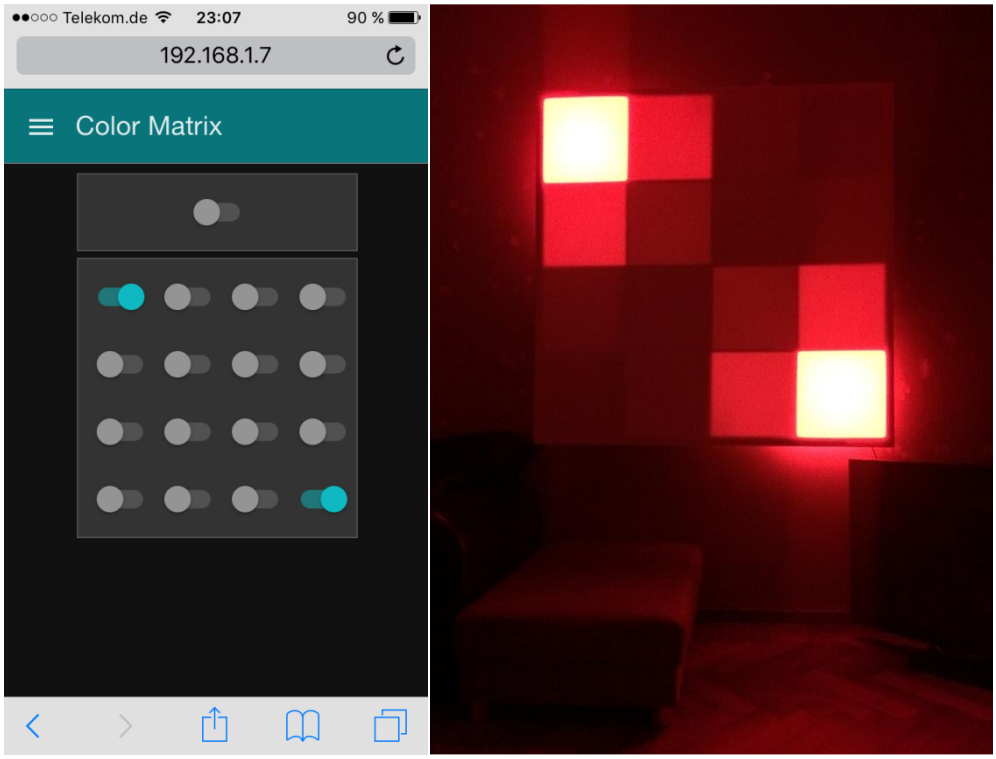


Abbildung 5-29: Vergleich – Ansicht in der Web-Oberfläche und Color Matrix

#### 5.2.1.3 Time Matrix

Ein speziell für die Hue Color Matrix entwickeltes Programm ist die Time Matrix. Dazu wird kein zentraler Verwaltungsserver benötigt. Es wurde ein ESP-01 Modul in einen einfachen Batteriehalter mit Schalter eingebaut. Im Inneren des Gehäuses war genug Platz vorhanden um ihn zu verstauen. Dazu musste jedoch ein kleinerer Schalter eingebaut werden und ein Loch für die Pins in das Gehäuse geschnitten werden. Dies ist wichtig, um den „Remote Kasten“ auch weiterhin mit anderem Programmcode bespielen zu können. Das Modul verbindet sich im 10-sekündigen Abstand mit einem NTP (Network Time Protocol) Zeitserver und stellt die aktuelle Uhrzeit dann in der Matrix senkrecht von unten nach oben zu lesen im Binärcode dar. Dabei werden die einzelnen Zellen nur ein- und ausgeschaltet. Dadurch kann ein beliebiges Farbmuster vor der Aktivierung gewählt werden, auf das dann die Uhrzeitanzeige angewendet wird. Bei der Entwicklung wurde schnell klar, dass das HTTP-Paket mühsam zusammengestellt werden muss, da die Nachricht nicht mit kleinem Overhead an ein Topic gesendet werden kann.


Abbildung 5-30: Kemote Kasten mit verbautem ESP-01 Modul

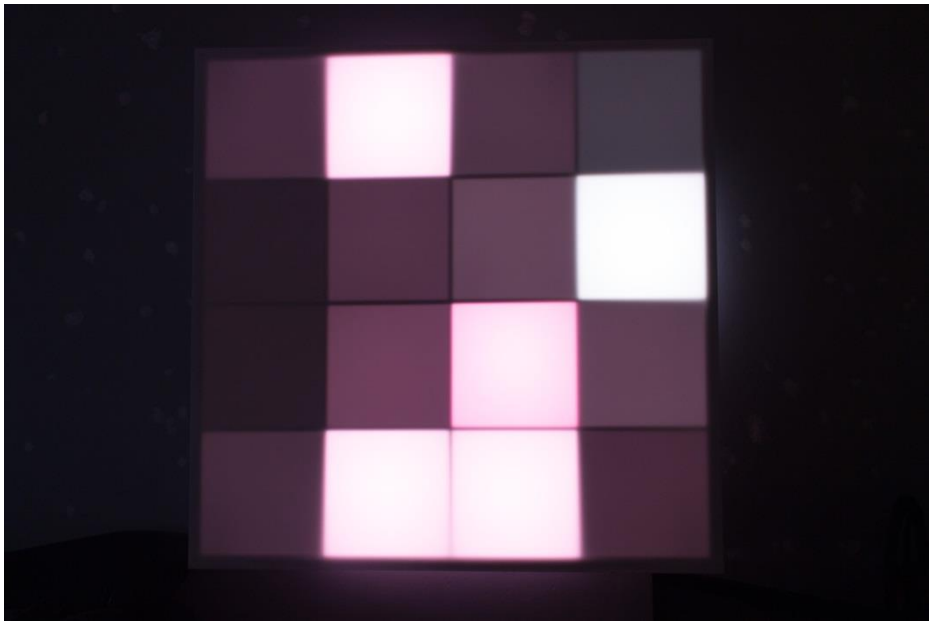

Abbildung 5-31: Hue Color Matrix mit Uhrzeit in binärer Anzeige (09:34 Uhr)

```
#include <ESP8266WiFi.h>
#include <WiFiUdp.h>
#include <ESP8266HTTPClient.h>
#define USE_SERIAL Serial

char ssid[] = "PINET";  //  your network SSID (name)
char pass[] = "SchneeOfen987";       // your network password

String content;
unsigned int localPort = 2390;       // local port to listen for
UDP packets

/* Don't hardwire the IP address or we won't get the benefits of
the pool.
 *  Lookup the IP address for the host name instead */
//IPAddress  timeServer(129,  6,  15,  28);  //  time.nist.gov  NTP
server
IPAddress timeServerIP; // time.nist.gov NTP server address
const char* ntpServerName = "0.de.pool.ntp.org";
int hour1, hour2, minute1, minute2;

const  int  NTP_PACKET_SIZE  =  48;  //  NTP  time  stamp  is  in  the
first 48 bytes of the message

byte  packetBuffer[  NTP_PACKET_SIZE];  //buffer  to  hold  incoming
and outgoing packets

// A UDP instance to let us send and receive packets over UDP
WiFiUDP udp;
WiFiClient client;

void setup()
{
  Serial.begin(115200);
  Serial.println();
  Serial.println();

  // We start by connecting to a WiFi network
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, pass);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");

  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());

  Serial.println("Starting UDP");
  udp.begin(localPort);
  Serial.print("Local port: ");
  Serial.println(udp.localPort());
}
```

```
void loop()
{
  //get a random server from the pool
  WiFi.hostByName(ntpServerName, timeServerIP);

  sendNTPpacket(timeServerIP); // send an NTP packet to a time
server
  // wait to see if a reply is available
  delay(1000);

  int cb = udp.parsePacket();
  if (!cb) {
    Serial.println("no packet yet");
  }
  else {
    Serial.print("packet received, length=");
    Serial.println(cb);
    // We've received a packet, read the data from it
    udp.read(packetBuffer, NTP_PACKET_SIZE); // read the packet
into the buffer

    //the timestamp starts at byte 40 of the received packet and
is four bytes,
    // or two words, long. First, esxtract the two words:

    unsigned long highWord = word(packetBuffer[40], packetBuff-
er[41]);
    unsigned long lowWord = word(packetBuffer[42], packetBuff-
er[43]);
    // combine the four bytes (two words) into a long integer
    // this is NTP time (seconds since Jan 1 1900):
    unsigned long secsSince1900 = highWord << 16 | lowWord;
    Serial.print("Seconds since Jan 1 1900 = " );
    Serial.println(secsSince1900);

    // now convert NTP time into everyday time:
    Serial.print("Unix time = ");
    // Unix time starts on Jan 1 1970. In seconds, that's
2208988800:
    const unsigned long seventyYears = 2208988800UL;
    // subtract seventy years:
    unsigned long epoch = secsSince1900 - seventyYears;
    // print Unix time:
    Serial.println(epoch);


    // print the hour, minute and second:
    Serial.print("The UTC time is ");       // UTC is the time
at Greenwich Meridian (GMT)
    hour1 = ((epoch  % 86400L) / 3600) + 1;  //for Germany
    hour2 = hour1 % 10; //last number
    hour1 /=10; //first number
    Serial.print((epoch  % 86400L) / 3600); // print the hour
(86400 equals secs per day)
    Serial.print(':');
    minute1 = (epoch  % 3600) /60;
```

```
    minute2 = minute1 % 10; //last number
    minute1 /= 10;
    if ( ((epoch % 3600) / 60) < 10 ) {
      // In the first 10 minutes of each hour, we'll want a
leading '0'
      Serial.print('0');
    }
    Serial.print((epoch  % 3600) / 60); // print the minute
(3600 equals secs per minute)
    Serial.print(':');
    if ( (epoch % 60) < 10 ) {
      // In the first 10 seconds of each minute, we'll want a
leading '0'
      Serial.print('0');
    }
    Serial.println(epoch % 60); // print the second
    Serial.println("");
    Serial.println(hour1);
    Serial.println("");
    Serial.println(hour2);
    Serial.println("");
    Serial.println(minute1);
    Serial.println("");
    Serial.println(minute2);

    hue_time_matrix();
    delay(1000);
    hue_time_matrix(); //sometimes is doesnt change on light
  }
  // wait ten seconds before asking for the time again
  delay(10000);
}

// send an NTP request to the time server at the given address
unsigned long sendNTPpacket(IPAddress& address)
{
  Serial.println("sending NTP packet...");
  // set all bytes in the buffer to 0
  memset(packetBuffer, 0, NTP_PACKET_SIZE);
  // Initialize values needed to form NTP request
  // (see URL above for details on the packets)
  packetBuffer[0] = 0b11100011;   // LI, Version, Mode
  packetBuffer[1] = 0;     // Stratum, or type of clock
  packetBuffer[2] = 6;     // Polling Interval
  packetBuffer[3] = 0xEC;  // Peer Clock Precision
  // 8 bytes of zero for Root Delay & Root Dispersion
  packetBuffer[12]  = 49;
  packetBuffer[13]  = 0x4E;
  packetBuffer[14]  = 49;
  packetBuffer[15]  = 52;

  // all NTP fields have been given values, now
  // you can send a packet requesting a timestamp:
  udp.beginPacket(address, 123); //NTP requests are to port 123
  udp.write(packetBuffer, NTP_PACKET_SIZE);
  udp.endPacket();
```

```
}

void hue_time_matrix(){
    hue_bulb(1,hour1/8);
    hour1%=8;
    hue_bulb(8,hour1/4);
    hour1%=4;
    hue_bulb(5,hour1/2);
    hour1%=2;
    hue_bulb(7,hour1);

    hue_bulb(2,hour2/8);
    hour2%=8;
    hue_bulb(9,hour2/4);
    hour2%=4;
    hue_bulb(6,hour2/2);
    hour2%=2;
    hue_bulb(11,hour2);

    hue_bulb(3,minute1/8);
    minute1%=8;
    hue_bulb(15,minute1/4);
    minute1%=4;
    hue_bulb(12,minute1/2);
    minute1%=2;
    hue_bulb(10,minute1);

    hue_bulb(16,minute2/8);
    minute2%=8;
    hue_bulb(13,minute2/4);
    minute2%=4;
    hue_bulb(4,minute2/2);
    minute2%=2;
    hue_bulb(14,minute2);
}

void hue_bulb(int id, int stat){
    if (client.connect("192.168.1.9", 80)) {
      Serial.println("Connected to server");
      // Make a HTTP request
      if(stat ==1)
        content = "{\"on\":true}";
      else
        content = "{\"on\":false}";
      client.print("PUT
/api/e1gHtzadVQCMsjJK9BmoV5f1wgro7zdyiJEOu4eo/lights/");
      client.print(id);
      client.println("/state HTTP/1.1");
      client.println("keep-alive");
      client.println("Host: 192.168.1.9");
      client.print("Content-Length: ");
      client.println(content.length());
      client.println("Content-Type: text/plain;charset=UTF-8");
      client.println();
      client.println(content);
    }
    else
```

```
    {
      Serial.println("connection failed http put");
      return;
    }
    client.stop();

}

void hue_group(int id, int stat){
    if (client.connect("192.168.1.9", 80)) {
      Serial.println("Connected to server");
      // Make a HTTP request
      if(stat==1)
        content= "{\"on\":true}";
      else
        content = "{\"on\":false}";
      client.print("PUT
/api/e1gHtzadVQCMsjJK9BmoV5f1wgro7zdyiJEOu4eo/groups/");
      client.print(id);
      client.println("/action HTTP/1.1");
      client.println("keep-alive");
      client.println("Host: 192.168.1.9");
      client.print("Content-Length: ");
      client.println(content.length());
      client.println("Content-Type: text/plain;charset=UTF-8");
      client.println();
      client.println(content);
    }
    else
    {
      Serial.println("connection failed http put");
      return;
    }
    client.stop();
}
```

Quellcode 5-2: Zeitabfrage und binäre Zeitdarstellung in der Hue Color Matrix

#### 5.2.1.4 Weitere Verwendungsmöglichkeiten

Die Color Matrix lässt sich mit jeder herkömmlichen Hue App ansteuern und auch damit sind schöne Muster und Farbwechsel möglich, die das Objekt als Kunstobjekt in den Raum integrieren. Auf ein 4x4-Farbquadrat ist jedoch keine der Apps ausgelegt. Node-RED bietet die Möglichkeit zur vielfältigen Verwendung. Zum einen kann die Color Matrix als Monitor für verschiedene Ereignisse eingesetzt werden und zum anderen lassen sich damit auch einfache Spiele entwickeln. Eine kleine Version von Schiffe-Versenken könnte ohne weiteres auf Basis der Ansicht in Abbildung 5-29 (links) entwickelt werden.

Entgegen der Erwartungen, kann mit 4x4 Pixeln sogar geschrieben und gemalt werden. Einige Figuren sind tatsächlich auf Anhieb zu erkennen.

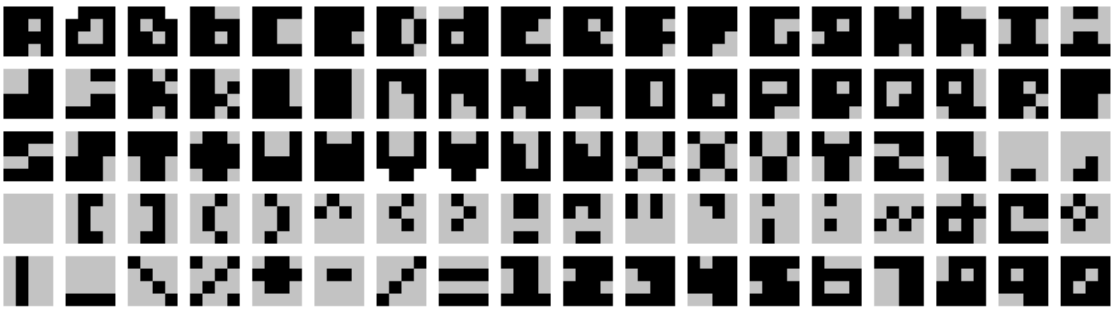

Abbildung 5-32: 4x4 Pixel Schrift [117]

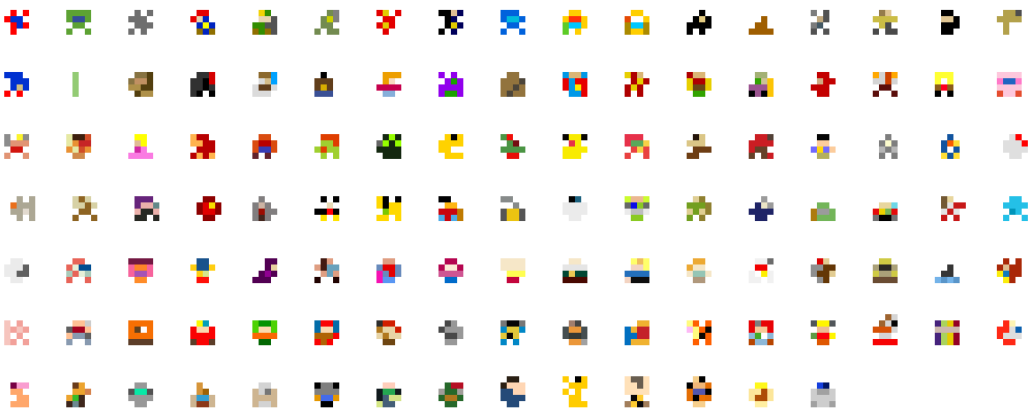

Abbildung 5-33: 4x4 Pixel Art [118]

Durch die einfache Hue API wäre auch eine Anbindung an einen Audio-Player denkbar. Dieser könnte die vier Spalten des Quadrats für die Anzeige des Ausschlags in vier Frequenzbereichen nutzen. Aufgrund des Delays sollte dabei der Titel optimalerweise zweimal vom Player eingelesen werden. Einmal um die Frequenzen zu analysieren und die Felder anzusteuern und ein weiteres Mal mit etwas Verzögerung, um den Titel synchron zur Anzeige abzuspielen.

### 5.2.2 ESP-01 Development Board

Die Grundlage der Microcontroller-Programmierung ist ein Adapter, mit dem die geschriebenen Programme auf die MCU übertragen werden können. Solche Programmieradapter wurden bereits vorgestellt. Während der Entwicklung ist es jedoch von Vorteil, wenn das aufgespielte Programm direkt getestet werden kann. Aus diesem Grund wurde ein ESP-01 Development Board entwickelt, welches an einen USB-Seriell-Adapter angeschlossen werden kann, über sämtliche Ein- und Ausgänge verfügt und zudem über eine Spannungsquelle auch ohne den Adapter verwendet werden kann.

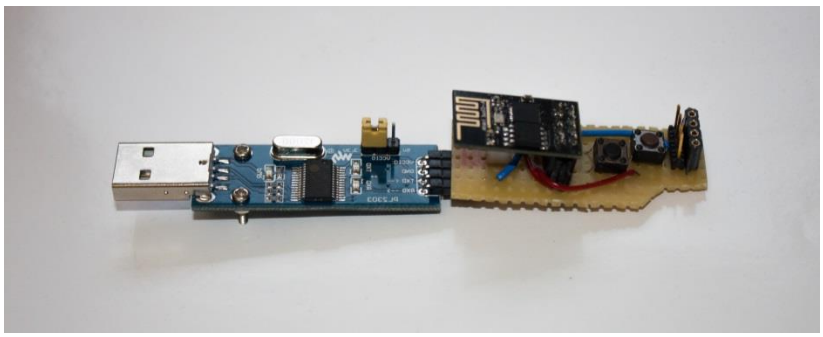

Abbildung 5-34: ESP-01 Development Board im Betrieb über USB (Knöpfe: RST/Flashmodus)

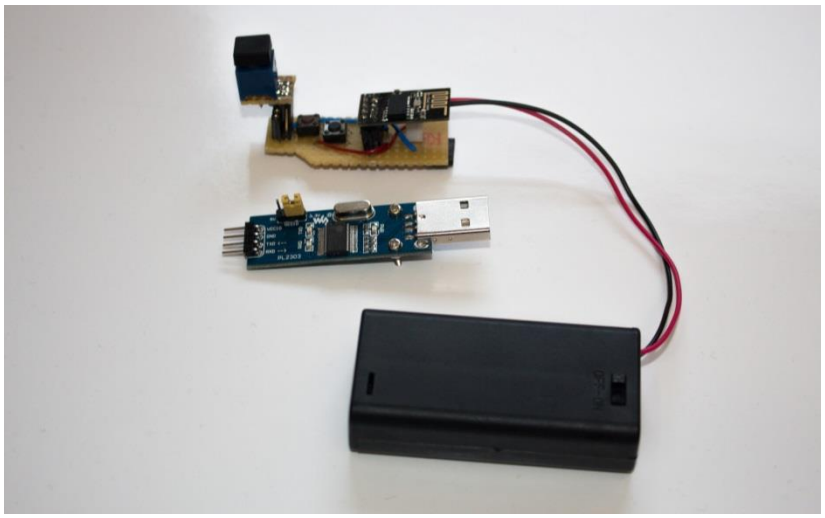

Abbildung 5-35: ESP-01 Development Board mit externer Spannungsquelle

Die männlichen und weiblichen Steckverbindungen ermöglichen den Anschluss eines Sensors über Kabel oder einem direkten Aufstecken. Um im gesamten Projekt ein problemloses Wechseln von Sensoren zu ermöglichen, ist diese Art der Verbindung bei allen zusammen entwickelten Modulen gleich. Natürlich sind auch die Pins immer in der gleichen Reihenfolge belegt: VCC, GND, GPIO2, GPOI0 (nur beim ESP-8266 Development Board verkabelt).

### 5.2.3 Basis-Board für Sensoren und Aktuatoren

Das Basis-Board für Sensoren sollte nur eine geringe Größe beanspruchen. Aus diesem Grund wurden nur die Anschlüsse gelegt, die von allen verwendeten Sensoren und Aktuatoren benötigt werden.

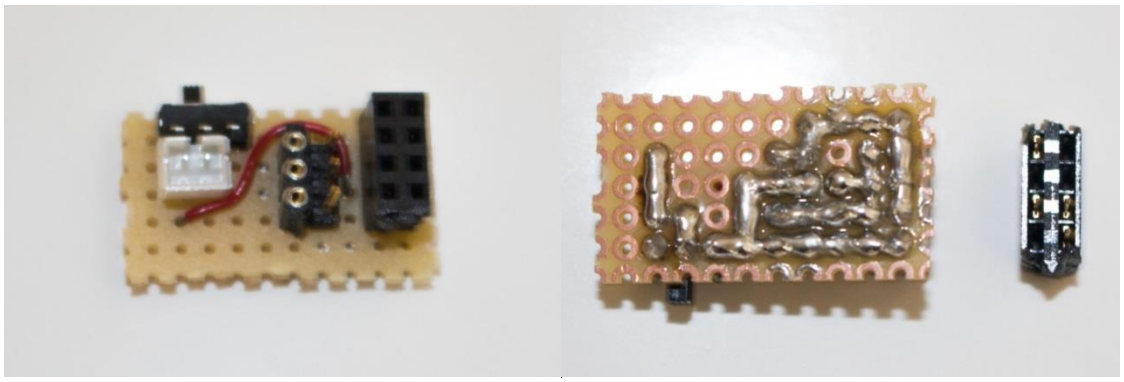

Abbildung 5-36: Basis Board zum Aufstecken und Anschließen von Sensoren und Aktuatoren

Um die Leiterbahnen so nah aneinander vorbeiführen zu können, wurden alle Pins der Steckverbindung für den ESP-01, die nicht verwendet wurden, vorher entfernt (siehe Abbildung 5-34 rechts). Aufgrund des einheitlichen Verbindungselements, können alle vorliegenden Sensoren verwendet und (bis auf den Temperatursensor) auch ausgetauscht werden.

Der ESP8266 Chip verfügt auch über einen sogenannten Deep Sleep, bei dem das Modul in einen sehr stromsparenden Modus wechselt. Beim ESP-01 fehlt jedoch die Verbindung zum Reset-Kontakt, um selbstständig wieder aus diesem Zustand zu „erwachen“. Dieser Kontakt kann jedoch auch aufgelötet werden (siehe Abbildung 5-35). Sofern ein Sensor am ESP-01 betrieben wird, welches selbst auch über Spannung versorgt wird, reduziert sich der Stromverbrauch jedoch nur um den des ESP-01.

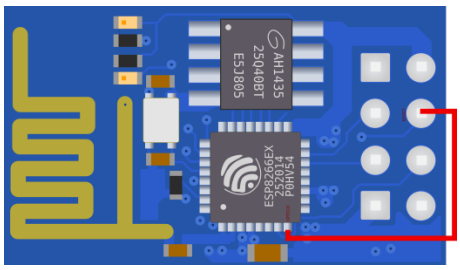

Abbildung 5-37: Verbindung zum „Aufwachen“ aus dem Deep Sleep [119]

Der Deep Sleep kann über folgenden Befehl in der Arduino IDE aktiviert werden. Die Variable sleepTimeS beinhaltet dabei einen Wert in Sekunden.

```
ESP.deepSleep(sleepTimeS * 1000000);
```

Quellcode 5-3: Einleiten des Deep Sleep auf dem ESP-01

### 5.2.4 IoT Button

Nachdem festgestellt wurde, dass der AWS IoT Button trotz WLAN keinen Strom verbraucht, wenn er nicht aktiviert wurde, gab es die Überlegung, mithilfe des ESP-01 einen solchen nachzubauen, der dadurch jedoch dann beliebig programmierbar wäre. Um das Modul vollständig von der Energieversorgung zu trennen, gab es verschiedene Ansätze mit Transistoren. Nach mehreren gescheiterten Versuchen, musste diese Idee jedoch verworfen werden. Diese Erkenntnis machte dann auch alle anderen bisherigen Nachforschungen zu Nichte. Bis auf die Stromversorgung wäre alles möglich gewesen. Der ESP-01 kann auch als Accesspoint eingerichtet werden und es ist möglich, Daten im Speicher zu sichern. Dadurch hätte man die Funktion des Buttons, oder zumindest Teile davon, auch ohne erneutes Überspielen der Firmware ändern können.

Aus diesem Grund, wurde ein Button entwickelt, der anstelle eines Sensors auf die vorgestellten Platinen aufgesteckt werden kann. Ein einfacher Taster hätte nicht funktioniert, da dieser in dem Modus, in dem der Eingang beim Betrieb mit einem Sensor ist, das am Eingang anliegende Signal nur einmal geändert hätte. Um die Funktionalität des Tasters zu ermöglichen, musste ein Widerstand vom Pin GPIO2 auf GND Gelegt werden. Da dies beim Bootvorgang einen anderen Modus einleitet, muss der Taster beim Einschalten des Moduls gedrückt sein.

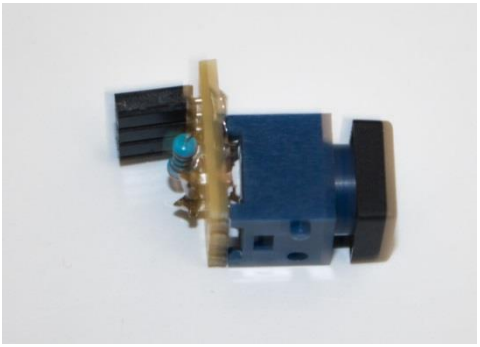

Abbildung 5-38: Taster zur Verwendung an einem für einen Sensor eingerichteten Modul

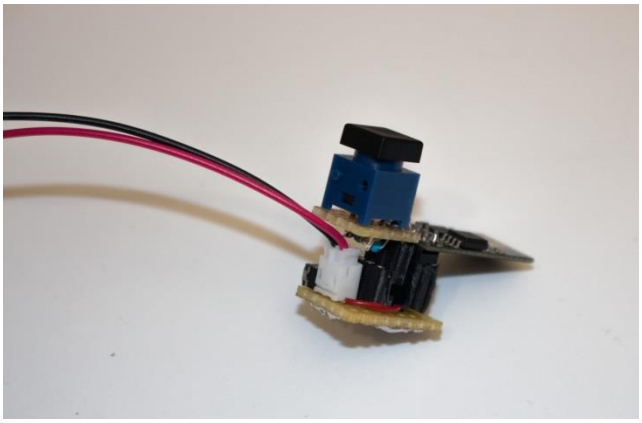

Abbildung 5-39: Taster aufgesteckt auf dem Basismodul

## 5.3 Zusammenspiel mehrerer Komponenten

Um festzustellen, wie die ganzen bisher vorgestellten Elemente und Verwaltungssoftware zusammenarbeiten, werden diese in einer kleinen Installation in der Wohnung eingesetzt. Dabei kommt vor allem das MQTT-Protokoll zum Einsatz, da es in der Verwaltungssoftware Node-RED sehr einfach zu verwenden ist. Um die Verwendung von HTTP mit Node-RED zu testen, wird ein Relais damit angesteuert.

### 5.3.1 Verwendete Elemente

Die in dieser Demonstration verwendeten Elemente werden im Folgenden aufgelistet.

- Raspberry Pi 3 Model B
  - Raspbian Jessie
  - Node-RED
  - MQTT Broker Mosquitto
- Hue Color Matrix
  - 16 Hue White & Color Birnen verbaut in einem 4x4 KALLAX Regal
- ESP8266/ESP-01
  - 2 x mit DHT11 Temperatursensor
  - 1 x mit Lichtsensor für das Erkennen der Türklingel
  - 1 x mit einem Relais zur Steuerung einer Stehlampe (über HTTP)

#### 5.3.1.1 Programmierung der ESP8266 Module

Dank der Arbeit der Entwickler bei Adafruit [120] konnte die sehr einfach zu bedienende Arduino IDE zur Programmierung in C verwendet werden. Ein weiterer Vorteil sind die vielfältigen Beispiele, die mit wenig Aufwand angepasst werden können und dank einer guten Dokumentation einfach zu verstehen sind. Im Folgenden werden die angepassten Quellcodes aufgelistet.

##### 5.3.1.1.1 Temperatur- und Feuchtigkeitssensoren

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include "DHT.h"

// Pin
#define DHTPIN 2

// Use DHT11 sensor
#define DHTTYPE DHT11

// Update these with values suitable for your network.

const char* ssid = "PINET";
const char* password = "SchneeOfen987";
const char* mqtt_server = "192.168.1.7";
```

```
  // Initialize DHT sensor
  DHT dht(DHTPIN, DHTTYPE, 15);

WiFiClient espClient;
PubSubClient client(espClient);

void setup() {
  Serial.begin(115200);
  //pinMode(2, INPUT);

  // Init DHT
  dht.begin();

  setup_wifi();
  client.setServer(mqtt_server, 1883);
  //client.setCallback(callback);
}

void setup_wifi() {

  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

void callback(char* topic, byte* payload, unsigned int
length) {
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  Serial.println();
}

void reconnect() {
  // Loop until we're reconnected
```

```
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
    if (client.connect("TempWohnzimmer")) {
      Serial.println("connected");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}
void loop() {
  long now = millis();

  if (!client.connected()) {
    reconnect();
    Serial.println("reconnect...");
  }
  client.loop();
     // Read temperature as Celsius
     float t = dht.readTemperature();

     char msg[100];
     char temp[10];

     dtostrf(t,1,0,temp);

     // Display data
     Serial.print(msg);

     client.publish("temp/wohnzimmer", temp);

     delay(2000);

}
```

Quellcode 5-4: Temperaturabfrage mit Hilfe eines DHT11 Sensors

Bei einem Programmcode, der wie dieser von mehreren Sensoren für unterschiedliche Anwendungen verwendet werden kann, ist es besonders wichtig, dass bei dem Aufruf client.connect(„Clientname“) jeweils ein eigener Name für jedes Modul eingetragen wird, da diese sich sonst gegenseitig behindern.

#### 5.3.1.1.2 Lichtsensor

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

// Update these with values suitable for your network.

const char* ssid = "PINET";
const char* password = "SchneeOfen987";
const char* mqtt_server = "192.168.1.7";

int inputState = 0;
int oldState = 0;

WiFiClient espClient;
PubSubClient client(espClient);

void setup() {
  Serial.begin(115200);
  pinMode(2, INPUT);

  setup_wifi();
  client.setServer(mqtt_server, 1883);
}

void setup_wifi() {

  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
    if (client.connect("Klingel")) {
```

```
      Serial.println("connected");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}
void loop() {
  long now = millis();

  if (!client.connected()) {
    reconnect();
    Serial.println("reconnect...");
  }
  client.loop();
     inputState = digitalRead(2);
     if( inputState == HIGH && oldState != inputState){
        client.publish("klingel", "1");
        oldState=inputState;
        delay(10000);
     }
     else
     oldState = inputState;


}
```

Quellcode 5-5: Lichtsensor zum Erkennen der Türklingel (Sprechanlage blinkt)

Sofern man diese Funktion der Türklingelerkennung dauerhaft nutzen möchte, würde es sich anbieten, den ESP-01 in die Gegensprechanlage zu integrieren. Zum einen könnte dieser dort über die verfügbare Betriebsspannung versorgt werden, zum anderen könnte man beide GPIOs verwenden um z.B. auch per Fernzugriff auf das User Interface oder mit einer Antwort SMS nach Benachrichtigung über ein Klingeln den Türöffner zu aktivieren. In diesem Fall wurde zum Testen der Lichtsensor verwendet, der das erste Aufleuchten des Knopfes erkennt und dann 10 Sekunden wartet um von einem Klingeln nicht mehrere Klingelmeldungen auszulösen.

#### 5.3.1.1.3 Relais (über HTTP)

```
#include <ESP8266WiFi.h>

const char* ssid = "PINET";
const char* password = "SchneeOfen987";

// Create an instance of the server
// specify the port to listen on as an argument
WiFiServer server(80);

void setup() {
  Serial.begin(115200);
  delay(10);

  // prepare GPIO2
  pinMode(2, OUTPUT);
  digitalWrite(2, 0);

  // Connect to WiFi network
  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.hostname("ESPRelaisWeb");
  WiFi.begin(ssid, password);


  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");

  // Start the server
  server.begin();
  Serial.println("Server started");

  // Print the IP address
  Serial.println(WiFi.localIP());
}

void loop() {
  // Check if a client has connected
  WiFiClient client = server.available();
  if (!client) {
    return;
  }

  // Wait until the client sends some data
```

```
  Serial.println("new client");
  while(!client.available()){
    delay(1);
  }

  // Read the first line of the request
  String req = client.readStringUntil('\r');
  Serial.println(req);
  client.flush();

  // Match the request
  int val;
  if (req.indexOf("/gpio/0") != -1)
    val = 0;
  else if (req.indexOf("/gpio/1") != -1)
    val = 1;
  else {
    Serial.println("invalid request");
    client.stop();
    return;
  }

  // Set GPIO2 according to the request
  digitalWrite(2, val);

  client.flush();

  // Prepare the response
  String s = "HTTP/1.1 200 OK\r\nContent-Type:
text/html\r\n\r\n<!DOCTYPE HTML>\r\n<html>\r\nGPIO is now
";
  s += (val)?"high":"low";
  s += "</html>\n";

  // Send the response to the client
  client.print(s);
  delay(1);
  Serial.println("Client disonnected");

  // The client will actually be disconnected
  // when the function returns and 'client' object is
detroyed
}
```

Quellcode 5-6: Relaissteuerung über HTTP GET Request

Um auch die Funktionalität und Verwendung von HTTP mit Node-RED testen zu können, wurde ein 230V Relais an eine Mehrfachsteckdose mit einer Stehlampe angeschlossen. Der Quellcode konnte ohne große Änderungen aus einem vorhandenen Beispiel zum Betrieb des ESP-01 als „Webserver“ übernommen werden. Lediglich die Daten für das WLAN mussten eingetragen werden. Im Gegensatz zu den mit MQTT betriebenen Modulen musste dieses Programm einmal auf dem Development Board gestartet werden, um die IP Adresse des Relais zu kennen.

### 5.3.1.2 Einrichten der zentralen Verwaltung über Node-RED

Zur Verwaltung der einzelnen Module und den Ereignissen, wurde das bereits vorgestellte und eingerichtete Node-RED verwendet. Mit einer einfachen grafischen Oberfläche können hier einfache und komplexe Abfolgen erstellt werden. Vor allem die MQTT-Nodes sind sehr einfach zu verwenden. Jedes Node, ob MQTT oder nicht, hat ein Topic Feld. Dies ermöglicht es dem Entwickler, ein Ausgangs-Node für MQTT ohne Topic einzurichten und mehrere andere Nodes daran anzuschließen, die in ihren Einstellungen das Topic festgelegt haben.

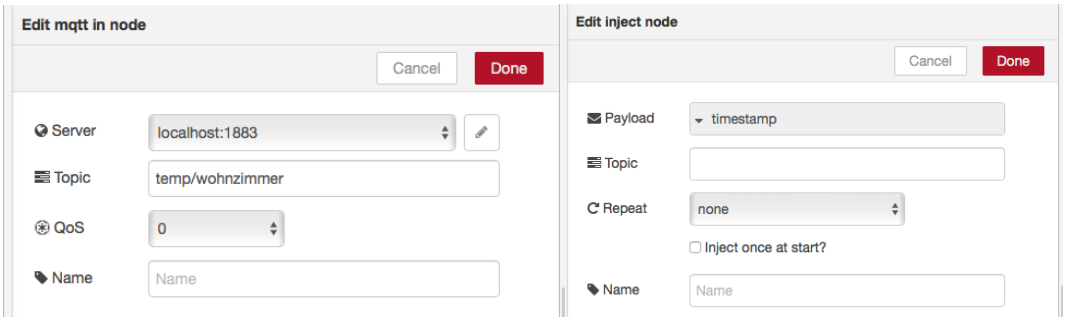


Abbildung 5-40: Einstellungsanzeige von Nodes (beide haben ein Topic Feld)

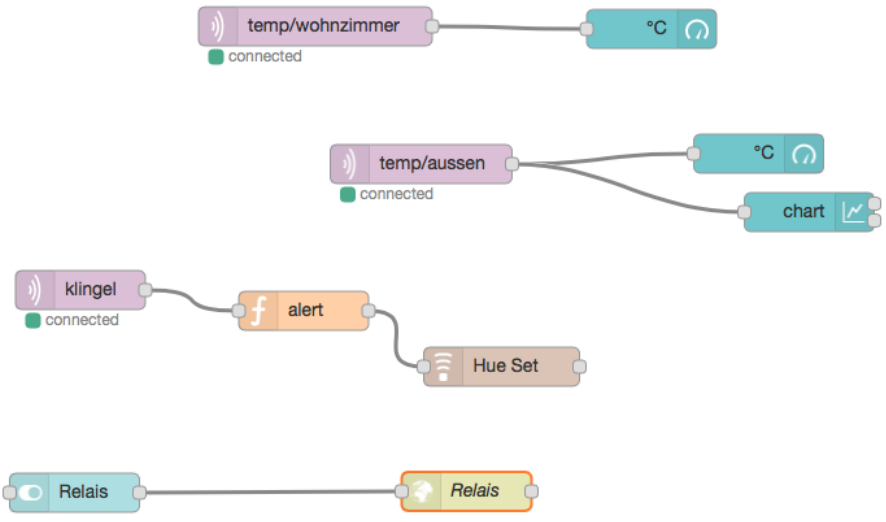


Abbildung 5-41: Verknüpfte Nodes in Node-RED

Node-RED ist vor allem auf die Verwendung von MQTT optimiert. Hier müssen kaum Anpassungen vorgenommen werden und der Nachrichtenaustausch erfolgt reibungslos und mit einer kaum bemerkbaren Verzögerung. Über Node-RED ist es einfach möglich, auch mehrere Ein- und Ausgänge zu verwalten. Über das zusätzlich installierte Dashboard lassen sich in kurzer Zeit auch Benutzeroberflächen zur Steuerung und Überwachung der eingebundenen Geräte erzeugen. In diesem Beispiel wurden die empfangenen MQTT-Daten der Temperaturen an eine Anzeige und die Außentemperatur zusätzlich an ein Diagramm weitergegeben. Das betätigen der Türklingel leitete einen „Alert“ an die Color Matrix weiter, der diese 30 Sekunden lang in der aktuellen Farbe blinken lässt.

#### 5.3.1.3 Ansicht des Webinterfaces

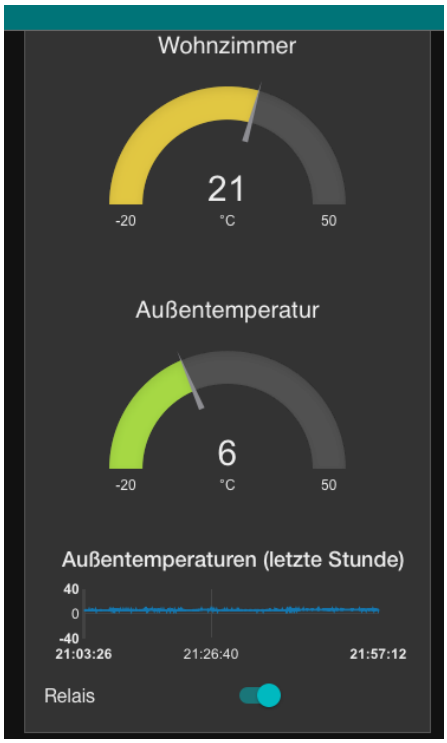


Abbildung 5-42: Webinterface zum Testaufbau

Über die installierte Erweiterung „Dashboard“ können sehr einfach und schnell Benutzeroberflächen erstellt werden. Diese sind responsive und können in verschiedenen Tabs und Gruppen organisiert werden. Dabei kann auch die jeweilige Größe des Elements festgelegt werden. In diesem kurzen Beispiel werden die beiden erfassten Temperaturen in einer Armaturenansicht dargestellt und den Verlauf der Außentemperatur letzten Stunde zusätzlich in einem Liniendiagramm visualisiert.

## 5.4 Fazit

Sofern man in den Bereich des Internet of Things einsteigen möchte, muss man sich zunächst überlegen, wozu es verwendet werden soll. Unter Umständen bietet einer der proprietären Anbieter bereits ein einfach einzurichtendes Paket an, welches im Optimalfall auch mehrere Übertragungstechniken und Protokolle versteht. Aber auch wenn man sich tiefer in die Materie einarbeiten Möchte um mehr Möglichkeiten zu haben, besteht ein großes Angebot an Hard- und Software sowie Forendiskussionen, und für die Entwicklung optimierte Development Boards.

Der Espruino kam in dieser Arbeit nicht zum Einsatz, ist aber vor allem für Einsteiger eine gute Möglichkeit erste Schritte in der Microcontroller-Programmierung in JavaScript zu machen.

Der ESP8266/ESP-01 wurde bei allen im Beispiel verwendeten Modulen eingesetzt. Aufgrund seiner Größe der Steckverbindungen ist es optimal, um ihn flexibel zu verwenden. Diese Module sind zudem sehr robust und haben mehrere nicht korrekt angeschlossene Signale und Spannungen verziehen. Worauf jedoch geachtet werden muss ist, dass keine 5V am VCC Pin angelegt werden. Dies wurde nicht getestet, soll laut Datenblatt aber nicht gemacht werden.

Das ESP8266/ESP-12E Development Board bietet sich vor allem dann an, wenn ausprobiert werden soll, was alles mit dem Board gemacht werden kann. Zusätzlich verfügt dieser über mehr Ein- und Ausgänge, einen Analog/Digital-Wandler und der Möglichkeit, ihn über Deep Sleep in einem sehr geringen Verbrauchsmodus (im Inaktiven Zustand) verwenden zu können, von dem er sich selbst wieder „aufwecken“ kann. Im Gegensatz zum ESP-01 kann dieses Modul auf einem Steckbrett verwendet werden, da die Pins nicht so nahe zusammenliegen. Dies fördert zudem die Möglichkeiten, vor allem am Anfang viel ausprobieren zu können.

Der ESP32 verspricht aufgrund seiner beiden Chips (WiFi und Bluetooth) den ESP8266 sehr bald ablösen zu werden. Die vielfältigen Möglichkeiten auch zum Deaktivieren nicht verwendeter Chips ermöglichen unzählige Anwendungen zu einem sehr günstigen Preis.

Der Raspberry Pi ist zwar im Vergleich das teuerste Element, lohnt sich jedoch in jedem Fall. Über den Raspberry Pi könnte man auch mit Node-RED sehr einfach die Pins ansteuern und hier ebenso Sensoren anschließen. Er ist aber vor allem aufgrund seiner Einsatzmöglichkeiten als zentrale Verwaltungseinheit besonders geeignet. Hinzu kommen die vielen Programme und ganze Images, die auf den Raspberry Pi aufgespielt werden können. Da alles über die eingesteckte MicroSD-Karte läuft, kann auch sehr schnell zwischen verschiedenen Anwendungen gewechselt werden.

Die Hue Birnen wurden im Rahmen der Arbeit erst recht spät geliefert, weshalb die Color Matrix nicht so vielfältig eingesetzt werden konnte, wie es mit ihr eigentlich möglich wäre. Philips bietet mit den Lampen und der API für die Bridge ein System, das einfach und anschaulich verwendet werden kann. Seit der neuen Generation (3rd) haben die Birnen auch bessere Farben, was sich vor allem bei Grün und Blau zeigt.

Beacons wurden für dieses Projekt nicht verwendet, bieten mit ihrem geringen Stromverbrauch aber eine Vielzahl an Anwendungsfällen und sind auch im Bereich IoT auch in Zukunft sicher noch lange vertreten.

Der AWS IoT Button ist eine nette Idee von Amazon, um neben den Produktspezifischen Dash Buttons eine selbst einzurichtende Version anzubieten. Damit ist man jedoch in der Verwendung auf die Amazon Produkte beschränkt, weshalb er in dieser Arbeit nicht zum Einsatz gekommen ist.

Mit der Arduino IDE ließen sich alle vorgestellten Anwendungen einfach umsetzten. Vor allem das große Angebot an Support und zusätzlichen Bibliotheken macht die Arduino IDE zum optimalen Werkzeug.

Im Beispielprojekt konnten die entwickelten Elemente sehr einfach und schnell miteinander verbunden werden. Hier konnte man auch einen Eindruck davon bekommen, was das Internet of Things ausmacht: Es besteht aus vielen keinen Dingen, die entweder etwas Triviales melden oder ausführen. Im großen Gesamten zeigt sich jedoch dann, wie wichtig jedes einzelne Element ist welche Möglichkeiten in deren Zusammenwirken entstehen.

Bei den verwendeten beiden Protokollen hat sich MQTT eindeutig als das bessere Protokoll in der Anwendung und Verwaltung abgezeichnet. HTTP macht z. B. da Sinn, wo kein zentraler Verwalter verwendet werden soll/kann oder ein Wert direkt von einem Sensor abgefragt werden soll, da man bei MQTT ja auf den Publisher warten muss. Neben den gezeigten

# 6 Zusammenfassung und Ausblick

Bei der Untersuchung der Protokolle ist schnell aufgefallen, dass eine klare Unterscheidung der Schichten nach dem OSI-Modell hier immer mehr aufgegeben wird. Ein Einfaches austauschen einer Schicht mit einem anderen Protokoll ist hier nicht ohne weiteres möglich. Teilweise werden Nachrichten bereits in der Transportschicht administriert oder ein „Protokoll“ übernimmt Aufgaben einer darunter liegenden Schicht und ist mehr oder weniger eher als Middleware zu sehen. Es gibt weder bei den Protokollen aus auch bei den Übertragungstechniken mehrere Standards, die alle unterschiedliche Vor- und Nachteile haben. Vor allem im Bereich Smart Home müssen neu auf den Markt kommende Geräte eine Vielzahl an Protokollen und Übertragungstechniken unterstützen um sich durchsetzen zu können. Denn was bringt einem eine zentrale Steuerungseinheit, die nur eine Teilmenge der im Haus verwendeten Geräte abdeckt. Hier wäre es dingend wichtig, sich auf wenige Übertragungstechniken und Anwendungsschicht-Protokolle zu einigen. Es scheint derzeit einfach jeder bei diesem neuen „Internet of Things“ mitmachen zu wollen.

Bereits zu Beginn der Untersuchung war zu vermuten, dass es nicht ein bestimmtes Protokoll geben wird, welches dann als das ultimative IoT-Anwendungsschicht-Protokoll definiert werden kann. Hinzu kommt, dass keines der untersuchten Protokolle als reines IoT-Protokoll entwickelt wurde. Ein ganz wichtiges Element in Bezug auf die Protokolle des IoT sind derzeit Gateways in jeglicher Form, über die die unterschiedlichen Netze und auch Protokolle verbunden werden können. Zudem bestehen ganze Geräte, die sowohl mehrere Protokolle verwenden, als auch die zentrale Steuerung übernehmen.

MQTT wird als leichtgewichtiges Protokoll, das vielfältig unterstützt wird, zukünftig eine wichtige Rolle im Bereich der Publish/Subscribe-Architektur spielen. Durch seinen geringen Overhead und seine einfache Verwendung ist es optimal für Netzwerke, die über eingeschränkte Ressourcen verfügen.

HTTP/1.1 genießt aufgrund seiner bisher schon vielfältigen Verwendung eine weitreichende Unterstützung sämtlicher Geräte. Mit der Anwendung des REST-Architekturstils auf HTTP/1.1 wird es auch weiterhin ein wichtiges Protokoll im Bereich Request/Response-Architektur sein.

Der große Nachteil von HTTP/1.1, nämlich sein sehr großen Overhead, wurde bei HTTP/2 verbessert, was vor allem beim Austausch von mehreren Nachrichten über eine Verbindung durch das mögliche Weglassen von Head-Attributen erkennbar ist. HTTP/2 wird nativ aber noch von keiner IoT-Plattform unterstützt.

Am Beispiel von REST ist schnell zu erkennen, wie sehr Protokolle verschiedener Sichten, Middleware und Architekturstile missverständlich verwendet werden. REST ist kein Protokoll sondern nur ein Architekturstil zur Verwendung eines Protokolls und kann auf verschiedene Protokolle angewendet werden. Sofern von einer REST API oder einfach nur von REST gesprochen wird, ist in der Regel die Unterstützung von HTTP/1.1 im RESTful-Design gemeint.

Mit CoAP kann eine einfache Verbindung eines Sensornetzwerks mit dem Internet vorgenommen werden. Dies liegt vor allem an der einfachen Möglichkeit der Umwandlung in HTTP. Damit wird der geringe Overhead im Sensornetzwerk genutzt und über die Umwandlung in HTTP steht eine breite Unterstützung durch andere Geräte zur Verfügung. Für diese Verbindung kommt aber auch wieder ein Gateway ins Spiel, welches in diesem Fall aber nur geringer Ressourcen bedarf.

Die große Flexibilität von XMPP und die unzähligen Erweiterungen machen XMPP sehr interessant für vielfältige Anwendungen. Jedoch kann hier die Abhängigkeit von einem Namensserver zum einen hinderlich sein, zum anderen durch global eindeutige Adressen auch vorteilhat. Im Bereich IoT wird XMPP nur sehr wenig verwendet.

DDS kann nicht als einfaches Protokoll gesehen werden, da es viele administrative Funktionalitäten abdeckt. Die Möglichkeit der Verwendung einer Publish/Subscribe-Architektur ohne zentrale Einheit beinhaltet zwar sehr einfach austauschbare Nachrichten, ist jedoch mit einem zusätzlichen Verwaltungsaufwand verbunden. Als datenzentrische Lösung mit den ausgefeilten QoS-Einstellungen wird es zwar im Consumerbereich wohl keine Verwendung finden, jedoch im Bereich der Industrie, wo der Datendurchsatz und eine geringe Latenz wichtiger sind als ressourcensparendes Verhalten. Hier steht DDS, was größtenteils in den USA verwendet wird, in Konkurrenz zu OPC UA, was sich im europäischen Raum als das IIoT-Protokoll/Middleware durchzusetzen scheint.

Das anfangs auch erwähnte Google Cloud Pub/Sub ist ebenfalls eine Middleware bzw. PaaS und SaaS (Software as a Service), was auf die Bedürfnisse des IoT eingeht und neben einem Publish/Subscribe-Verfahren auch eine Request/Response-Anbindung unterstützt.

Letztendlich sollte das zu verwendende Protokoll anhand der Anwendung ausgewählt werden. Wichtig dabei ist: welche Geräte werden verwendet, welche Übertragungstechniken werden genutzt, benötige ich eine zentrale Verwaltung?

Im praktischen Teil wurde größtenteils die Hardware behandelt. Diese intensive Auseinandersetzung konnte ein weitreichendes Verständnis für zukünftige Anwendungen schaffen. Hierbei hat sich abgezeichnet, dass das Internet der Dinge auch in Zukunft nicht von einem Protokoll oder einer Übertragungstechnik übernommen wird. Ein Zusammenwirken verschiedener Elemente und eine sinnvolle Verknüpfung unter Berücksichtigung der Anwendung ermöglichen den größtmöglichen Nutzen.

Der Espruino kam in dieser Arbeit nicht zum Einsatz, ist aber vor allem für Einsteiger eine gute Möglichkeit erste Schritte in der Microcontroller-Programmierung in JavaScript zu machen.

Der ESP8266/ESP-01 wurde bei allen im Beispiel verwendeten Modulen eingesetzt. Aufgrund seiner Größe der Steckverbindungen ist es optimal, um ihn flexibel zu verwenden. Diese Module sind zudem sehr robust und haben mehrere nicht korrekt angeschlossene Signale und Spannungen verziehen. Worauf jedoch geachtet werden muss ist, dass keine 5V am VCC Pin angelegt werden. Dies wurde nicht getestet, soll laut Datenblatt aber nicht gemacht werden.

Das ESP8266/ESP-12E Development Board bietet sich vor allem dann an, wenn ausprobiert werden soll, was alles mit dem Board gemacht werden kann. Zusätzlich verfügt dieser über mehr Ein- und Ausgänge, einen Analog/Digital-Wandler und der Möglichkeit, ihn über Deep Sleep in einem sehr geringen Verbrauchsmodus (im Inaktiven Zustand) verwenden zu können, von dem er sich selbst wieder „aufwecken“ kann. Im Gegensatz zum ESP-01 kann dieses Modul auf einem Steckbrett verwendet werden, da die Pins nicht so nahe zusammenliegen. Dies fördert zudem die Möglichkeiten, vor allem am Anfang viel ausprobieren zu können.

Der ESP32 verspricht aufgrund seiner beiden Chips (WiFi und Bluetooth) den ESP8266 sehr bald ablösen zu werden. Die vielfältigen Möglichkeiten auch zum Deaktivieren nicht verwendeter Chips ermöglichen unzählige Anwendungen zu einem sehr günstigen Preis.

Der Raspberry Pi ist zwar im Vergleich das teuerste Element, lohnt sich jedoch in jedem Fall. Über den Raspberry Pi könnte man auch mit Node-RED sehr einfach die Pins ansteuern und hier ebenso Sensoren anschließen. Er ist aber vor allem aufgrund seiner Einsatzmöglichkeiten als zentrale Verwaltungseinheit besonders geeignet. Hinzu kommen die vielen Programme und ganze Images, die auf den Raspberry Pi aufgespielt werden können. Da alles über die eingesteckte MicroSD-Karte läuft, kann auch sehr schnell zwischen verschiedenen Anwendungen gewechselt werden.

Die Hue Birnen wurden im Rahmen der Arbeit erst recht spät geliefert, weshalb die Color Matrix nicht so vielfältig eingesetzt werden konnte, wie es mit ihr eigentlich möglich wäre. Philips bietet mit den Lampen und der API für die Bridge ein System, das einfach und anschaulich verwendet werden kann. Seit der neuen Generation (3rd) haben die Birnen auch bessere Farben, was sich vor allem bei Grün und Blau zeigt.

Beacons wurden für dieses Projekt nicht verwendet, bieten mit ihrem geringen Stromverbrauch aber eine Vielzahl an Anwendungsfällen und sind auch im Bereich IoT auch in Zukunft sicher noch lange vertreten.

Der AWS IoT Button ist eine nette Idee von Amazon, um neben den Produktspezifischen Dash Buttons eine selbst einzurichtende Version anzubieten. Damit ist man jedoch in der Verwendung auf die Amazon Produkte beschränkt, weshalb er in dieser Arbeit nicht zum Einsatz gekommen ist.

Mit der Arduino IDE ließen sich alle vorgestellten Anwendungen einfach umsetzten. Vor allem das große Angebot an Support und zusätzlichen Bibliotheken macht die Arduino IDE zum optimalen Werkzeug.

Im Beispielprojekt konnten die entwickelten Elemente sehr einfach und schnell miteinander verbunden werden. Hier konnte man auch einen Eindruck davon bekommen, was das Internet of Things ausmacht: Es besteht aus vielen keinen Dingen, die entweder etwas Triviales melden oder ausführen. Im großen Gesamten zeigt sich jedoch dann, wie wichtig jedes einzelne Element ist welche Möglichkeiten in deren Zusammenwirken entstehen.

Bei den verwendeten beiden Protokollen hat sich MQTT eindeutig als das bessere Protokoll in der Anwendung und Verwaltung abgezeichnet. HTTP macht z. B. da Sinn, wo kein zentraler Verwalter verwendet werden soll/kann oder ein Wert direkt von einem Sensor abgefragt werden soll, da man bei MQTT auf den Publisher warten muss.

Abschließend kann noch gesagt werden, dass die Bereiche Internet der Dinge, Industrie 4.0, Smart Home etc. gerade erst am Beginn ihrer Entwicklung sind. Es bleibt spannend zu sehen, welche Protokolle sich durchsetzen und was in Zukunft noch alles möglich sein wird. Es ist jedoch wichtig, eine homogenere Protokoll-Landschaft zu schaffen und vor allem die Übertragungstechniken auf wenige Standards zu reduzieren. Die derzeitige Situation, dass viele verschiedene Firmen eher deshalb an IoT-Applikationen arbeiten um diesen Bereich auch mit abzudecken als aktiv neue Dinge zu entwickeln bremst den Fortschritt aus und sorgt beim einfachen Anwender eher für Verwirrung.

# Literaturverzeichnis

[1] A. Wood. (2015, Mar.) The Guardian. [Online]. https://www.theguardian.com/media-network/2015/mar/31/the-internet-of-things-is-revolutionising-our-lives-but-standards-are-a-must

[2] K. Ashton. (2009, Jun.) RFID Journal. [Online]. http://www.rfidjournal.com/articles/view?4986

[3] Springer Gabler Verlag. (2016, Sep.) Digitalisierung, online im Internet. [Online]. http://wirtschaftslexikon.gabler.de/Archiv/-2046143105/digitalisierung-v2.html

[4] T. H. Volker P. Andelfinger, *Internet der Dinge - Technik, Trends und Geschäftsmodelle*. Springer, 2015.

[5] H.-C. Dirscherl. (2016, Jul.) www.pcwelt.de. [Online]. http://www.pcwelt.de/ratgeber/Stau-Warnung-Google-Maps-Tomtom-Verkehrslage-Echtzeitverkehrsinformationen-373385.html

[6] (2016, Oct.) code-n.org. [Online]. https://www.code-n.org/fileadmin/Mediendatenbank_CODE_n/images/downloads/press_releases_download_images/CODE_n15_internet_of_things_infographic.jpg

[7] M. Broy, *CYBER-PHYSICAL SYSTEMS*, a. –. D. A. d. Technikwissenschaften, Ed. München: Springer-Verlag Berlin Heidelberg, 2010.

[8] F. M. Elgar Fleisch, *Das Internet der Dinge*. Springer, 2005.

[9] F. Resatsch, *Ubiquitous Computing - Developing and Evaluating Near Field*, P. D. H. Krcmar, Ed. München: Gabler Verlag | Springer Fachmedien Wiesbaden GmbH, 2010.

[10] A. Roth, *Einführung und Umsetzung von Industrie 4.0*, A. R. .-B. GmbH, Ed. Leinfelden-Echterdingen, Deutschland: Springer-Verlag Berlin Heidelberg, 2016.

[11] (2016, Sep.) Industrie-4-0.org. [Online]. http://www.industrie-4-0.org/dies-academicus-i40/img/Die_vier_Stufen_der_industriellen_Revolution.jpg

[12] C. Baun, *Computernetze kompakt*, 3rd ed. Frankfurth, Deutschland: Springer-Verlag Berlin Heidelberg, 2015.

[13] H. L. Morteza M. Zanjireh, "A Survey on Centralised and Distributed Clustering Routing Algorithms for WSNs," in *IEEE 81st Vehicular Technology Conference*, Glasgow, UK, 2015.

[14] R. K. Markus Kraußе, *Drahtlose ZigBee-Netzwerke*, F. U. o. A. Sciences, Ed. Frankfurt: Springer Fachmedien Wiesbaden, 2014.

[15] (2012, Nov.) EE Times Europe. [Online]. http://www.power-eetimes.com/content/zigbee-rescue-fifth-play-services/page/0/1

[16] (2016, Sep.) IT Wissen. [Online]. http://www.itwissen.info/definition/lexikon/ZigBee-IP.html

[17] The Thread Group. (2016, Jul.) The Thread Group. [Online]. http://threadgroup.org/Portals/0/documents/whitepapers/Thread%20Stack%20Fundamentals_v2_public.pdf

[18] Link Labs. (2016, Mar.) Link Labs – Thread Vs. ZigBee (For IoT

Engineers). [Online]. http://www.link-labs.com/thread-vs-zigbee-for-iot-engineers/

[19] N. Chen, *Bluetooth Low Energy Based CoAP Communication in IoT*. Saskatoon, Kanada, 2016.

[20] A. C. C. K. T. Robert Davidson, *Getting Started with Bluetooth Low Energy*. O'Reilly Media, Inc, 2014.

[21] M. Andersson, "Use case possibilities with Bluetooth low energy in IoT applications," u-blox AG, 2014.

[22] G. ,. D. I. M. &. L. J. L. M. Santillán Martínez, *A packet scheduler for real-time 6LoWPAN wireless networks in manufacturing systems*, J. I. Manuf, Ed. New York: Springer Science+Business Media, 2014.

[23] T. K. Ralf Gessler, *Wireless-Netzwerke für den Nahbereich*, 2nd ed., H. Heilbronn, Ed. Springer Fachmedien Wiesbaden, 2015.

[24] J. B. A. Brandt, *Transmission of IPv6 Packets over ITU-T G.9959 Networks*, I. E. T. F. (IETF), Ed. 02, 2015.

[25] O. S. Heinrich Vaske. (2016, Mar.) Computerwoche. [Online]. http://www.computerwoche.de/a/iot-produkte-und-strategien-der-hersteller,3212897

[26] (2016, Oct.) ALLSEEN ALLIANCE. [Online]. https://allseenalliance.org/

[27] OPEN CONNECTIVITY FOUNDATION. (2016, Jul.) OPEN CONNECTIVITY FOUNDATION. [Online]. https://openconnectivity.org/press-releases/thread-group-open-connectivity-foundation-create-liaison-agreement-increase-application-connectivity-choice-connected-home

[28] K. Dotchkoff. (2016, Aug.) Microsoft. [Online]. https://azure.microsoft.com/de-de/documentation/articles/iot-hub-protocol-gateway/

[29] R. Miller. (2016, Feb.) TechHcrunch. [Online]. https://techcrunch.com/2016/02/03/cisco-buys-jasper-technologies-for-1-4-billion/

[30] Cisco. (2016, Oct.) Cisco. [Online]. https://developer.cisco.com/site/internet-of-things/

[31] Amazon. (2016, Oct.) Amazon AWS. [Online]. http://docs.aws.amazon.com/iot/latest/developerguide/protocols.html

[32] J. Barr. (2016, Sep.) Amazon. [Online]. https://aws.amazon.com/de/blogs/aws/new-http2-support-for-cloudfront/

[33] IBM. (2016, Oct.) IBM Bluemix IoT. [Online]. http://www.ibm.com/cloud-computing/bluemix/internet-of-things/de-de/

[34] Intel. (2016, Oct.) Intel IoT Gateway Protocols. [Online]. https://software.intel.com/en-us/articles/a-comparison-of-iot-gateway-protocols-mqtt-and-modbus

[35] Intel. (2016, Oct.) The Intel IoT Platform. [Online]. http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/iot-platform-reference-architecture-paper.pdf

[36] SAP. (2016, Oct.) SAP HANA Cloud Site. [Online]. https://help.hana.ondemand.com/iot/frameset.htm?9da1c18f6ab947c58052f4d07498a654.html

[37] HP. (2016, Oct.) HP Enterprise IoT Solutions. [Online]. http://h41111.www4.hpe.com/solutions/iot/index.html

[38] oneM2M. (2016, Oct.) oneM2M Published Specifications. [Online]. http://www.onem2m.org/technical/published-documents

[39] Google. (2016, Oct.) Google Cloud Platform. [Online]. https://cloud.google.com/pubsub/

[40] Bosch. (2016, Oct.) Bosch IoT Things. [Online]. https://things.apps.bosch-iot-cloud.com/dokuwiki/doku.php?id=005_dev_guide:004_rest_api:004_rest_api

[41] PTC. (2016, Oct.) ThingWorx Support. [Online]. http://support.ptc.com/cs/help/thingworx_hc/thingworx_6.0_hc/index.jspx?id=thingworx10&action=show

[42] (2016, Oct.) Global M2M Association. [Online]. http://www.globalm2massociation.com/

[43] Oracle Corporation. (2016, Oct.) Oracle's Internet of Things Platform. [Online]. http://www.oracle.com/us/solutions/machine-to-machine/m2m-brochure-1951397.pdf?ssSourceSiteId=ocomde

[44] Oracle Corporation. (2016, Oct.) Oracle Help Center. [Online]. https://docs.oracle.com/cd/E37099_01/doc.20/e25066/install.htm#AELIG7017

[45] Oracle Corporation. (2016, Oct.) Oracle.com. [Online]. http://www.oracle.com/technetwork/topics/cloud/downloads/iot-mqtt-bridge-3112671.html

[46] BlackBerry. (2016, Oct.) BlackBerry IoT Platform. [Online]. https://docs.bbryiot.com/guides/messaging/index.html#using-messaging

[47] I. A. A. C. E. I. T. K. U. M. U. Z. O. Vogel, *Software-Architektur*, 2nd ed. Heidelberg: Spektrum Akademischer Verlag , 2009.

[48] (2016, Oct.) JSON.org. [Online]. http://www.json.org/json-de.html

[49] W. Huber, *Industrie 4.0 in der Automobilproduktion*. Wiesbaden: Springer Fachmedien, 2016.

[50] International Business Machines. (2016, Oct.) MQTT 3.1 Protocol Specifications. [Online]. http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html#qos-flows

[51] HiveMQ. (2016, Nov.) HiveMQ – Publish Subscribe. [Online]. http://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe

[52] HiveMQ. (2016, Nov.) HiveMQ – QoS. [Online]. http://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels

[53] HiveMQ. (2016, Nov.) HiveMQ – Topics. [Online]. http://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices

[54] IBM. (2016, Nov.) MQTT V3.1 Protocol Specification. [Online]. http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html#intro

[55] HiveMQ. (2016, Nov.) HiveMQ – Messages. [Online].

http://www.hivemq.com/blog/mqtt-essentials-part-4-mqtt-publish-subscribe-unsubscribe

[56] HiveMQ. (2016, Nov.) HiveMQ – Security. [Online]. http://www.hivemq.com/blog/mqtt-security-fundamentals-authentication-username-password

[57] H. L. T. Andy Stanford-Clark. (2013, Nov.) mqtt.org. [Online]. http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf

[58] C. Wue. (2016, Nov.) GitHub. [Online]. https://github.com/mqtt/mqtt.github.io/wiki/libraries

[59] M. Karow, *Vergleichende Untersuchung von „Platform as a Service“-Angeboten für das „Internet of Things“*. Offenburg, 2016.

[60] J. Wert. (2016, Nov.) deviceWISE. [Online]. https://help.devicewise.com/display/M2MOpen/MQTT+data+usage

[61] (2016, Oct.) OASIS. [Online]. https://www.oasis-open.org/news/announcements/mqtt-version-3-1-1-becomes-an-oasis-standard

[62] S. Shea. (2016, Feb.) techtarget.com. [Online]. http://internetofthingsagenda.techtarget.com/feature/IoT-messaging-The-MQTT-protocol-is-stepping-up-to-the-plate

[63] D. Obermaier. (2015, Nov.) Informatik aktuell. [Online]. https://www.informatik-aktuell.de/betrieb/netzwerke/iot-protokolldschungel-ein-wegweiser.html

[64] J. G. J. M. H. F. L. M. P. L. T. B.-L. R. Fielding. (1999, Jun.) Hypertext Transfer Protocol – HTTP/1.1. [Online]. https://tools.ietf.org/html/rfc2616

[65] D. Abts, *Masterkurs – Client/Server-Programmierung*, 4th ed. Wiesbaden: Springer Fachmedien, 2015.

[66] J. C. M. D. M. K. Balachander Krishnamurthy. (2016, Nov.) Key Differences between HTTP/1.0 and HTTP/1.1. [Online]. http://www8.org/w8-papers/5c-protocols/key/key.html

[67] (2016, Nov.) HTTP/2 – FAQ. [Online]. https://http2.github.io/faq/#will-http2-replace-http1x

[68] R. P. M. T. E. M. Belshe. (2015, May) Hypertext Transfer Protocol Version 2 (HTTP/2). [Online]. https://tools.ietf.org/html/rfc7540

[69] J. Kühner. (2016, Nov.) HTTP2-Spec. [Online]. https://github.com/http2/http2-spec/wiki/Implementations

[70] Amazon webservices. (2016, Sep.) Amazon webservices – cloudfront. [Online]. https://aws.amazon.com/de/about-aws/whats-new/2016/09/amazon-cloudfront-now-supports-http2/

[71] S. Ilya Grigorik. (2016, Nov.) Google Developers – Web. [Online]. https://developers.google.com/web/fundamentals/performance/http2/

[72] S. Fischerländer. (2016, Aug.) Expiredweb. [Online]. https://expiredweb.net/2501-http2-geringe-verbreitung-und-wenige-provider

[73] M. Dazer. (2012) TU-Berlin. [Online]. https://www.snet.tu-berlin.de/fileadmin/fg220/courses/WS1112/snet-project/restful-apis_dazer.pdf

[74] R. T. Fielding. (2000) Architectural Styles and the Design of Network-

based Software Architectures. [Online]. https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[75] S. Ullrich. (2014, Oct.) JAXenter. [Online]. https://jaxenter.de/wer-rest-will-muss-mit-hateoas-ernst-machen-489

[76] R. T. Fielding. (2008, Oct.) Untangled. [Online]. http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven

[77] K. H. C. B. Z. Shelby. (2014, Jun.) The Constrained Application Protocol (CoAP). [Online]. https://tools.ietf.org/html/rfc7252

[78] K. H. E. Z. Shelby. (2010, Oct.) Observing Resources in CoAP. [Online]. https://tools.ietf.org/html/draft-ietf-core-observe-00

[79] C. Bormann. (2016, Nov.) CoAP. [Online]. http://coap.technology/impls.html

[80] P. Saint-Andre. (2011, Mar.) Extensible Messaging and Presence Protocol (XMPP): Core. [Online]. http://xmpp.org/rfcs/rfc6120.html#streams

[81] P. M. R. M. Peter Saint-Andre. (2016, Oct.) XMPP.org – PubSub Extension. [Online]. http://www.xmpp.org/extensions/xep-0060.html

[82] P. Saint-Andre. (2013, Mar.) XMPP.org – XEP0305 Extension Pipelinging. [Online]. http://xmpp.org/extensions/xep-0305.html

[83] J. H. Peter Saint-Andre. (2015, Sep.) XMPP.org – Extension XEP-0033 Extended Stanza Addressing. [Online]. https://xmpp.org/extensions/xep-0033.html

[84] P. Saint-Andre. (2011, Mar.) Extensible Messaging and Presence Protocol (XMPP): Address Format. [Online]. https://tools.ietf.org/html/rfc6122

[85] E. ,. K. Z. E. A. Melnikov. (2006, Jun.) Simple Authentication and Security Layer (SASL). [Online]. https://tools.ietf.org/html/rfc4422

[86] (2016, Nov.) XMPP.org – Libraries. [Online]. https://xmpp.org/software/libraries.html

[87] A. Forster. (2016, Jul.) Elektroniknet.de. [Online]. http://www.elektroniknet.de/elektronik/embedded/iot-daten-in-echtzeit-132608-Seite-2.html

[88] D. Barnet. (2013, Mar.) MQTT and DDS Comparsion. [Online]. http://de.slideshare.net/RealTimeInnovations/comparison-of-mqtt-and-dds-as-m2m-protocols-for-the-internet-of-things

[89] S. Roth. (2016, Aug.) Informatik Aktuell. [Online]. https://www.informatik-aktuell.de/betrieb/netzwerke/zuverlaessige-datenkommunikation-im-industrial-internet-of-things-mit-dds.html

[90] B. F. R. W. Gerardo Pardo-Castellote. (2005, Aug.) OMG.org – An Introduction to DDS and Data-Centric Communications. [Online]. http://www.omg.org/news/whitepapers/Intro_To_DDS.pdf

[91] RTI. (2016, Nov.) RTI – RTI Connext DDS Secure. [Online]. https://www.rti.com/products/secure.html

[92] RTI. (2016, Nov.) RTI.com. [Online]. https://community.rti.com/static/documentation/connext-dds/5.2.3/doc/manuals/connext_dds/html_files/RTI_ConnextDDS_CoreLibraries_GettingStarted/index.htm#GettingStarted/An_Introduction_to_.htm

[93] Real Time Innovations. (2016, Nov.) RTI – Connext DDS Micro. [Online]. http://www.rti.com/products/micro.html

[94] OMG. (2016, Nov.) OMG.org. [Online]. http://www.omg.org/spec/#DDS

[95] (2016, Nov.) Google Trends. [Online]. https://www.google.de/trends/explore?cat=5&date=2013-10-26%202016-11-26&q=mqtt,coap,xmpp,RESTful,IoT

[96] (2016, Nov.) Espruino. [Online]. http://www.espruino.com/

[97] (2016, Nov.) Amazon. [Online]. http://ecx.images-amazon.com/images/I/41eYgFYY8BL.jpg

[98] (2016, Nov.) Adafruit – ESP32 Development Board. [Online]. https://www.adafruit.com/products/3269

[99] (2016, Nov.) Raspberry Pi.org. [Online]. https://www.raspberrypi.org/wp-content/uploads/2016/03/pi3.jpg

[100] (2016, Nov.) Amazon – Hue Starterset. [Online]. https://images-na.ssl-images-amazon.com/images/I/41UW3RgCakL.jpg

[101] (2016, Nov.) Kontakt.io. [Online]. https://store.kontakt.io/next-generation/31-card-beacon.html

[102] W. Chan. (2016, Nov.) Passkit.com. [Online]. https://blog.passkit.com/configure-iphone-ibeacon-transmitter/

[103] (2016, Nov.) Heavy Bubbles. [Online]. https://www.happybubbles.tech/presence/

[104] M. Terhaag. (2016, Sep.) Leagal Tribune Online. [Online]. http://www.lto.de/recht/hintergruende/h/amazon-dash-button-rechtliche-probleme-fragen/

[105] (2016, Nov.) Amazon webservices – AWS IoT Dash button. [Online]. https://aws.amazon.com/de/iot/button/

[106] (2016, Nov.) GitHub – Arduino IDE ESP8266. [Online]. https://github.com/esp8266/Arduino

[107] (2016, Nov.) Installing MQTT Broker Mosquitto. [Online]. http://www.instructables.com/id/Installing-MQTT-BrokerMosquitto-on-Raspberry-Pi/

[108] (2016, Nov.) Meet Hue – Getting Started. [Online]. https://developers.meethue.com/documentation/getting-started

[109] (2016, Nov.) Linuxcircle – node.js on bootup. [Online]. http://www.linuxcircle.com/2013/12/30/run-nodejs-server-on-boot-with-forever-on-raspberry-pi/

[110] C. Mobberley. (2016, Nov.) Adafruit – Waht is Node-RED. [Online]. https://learn.adafruit.com/raspberry-pi-hosting-node-red/what-is-node-red

[111] (2016, Nov.) Node-RED.org – Upgrading. [Online]. https://nodered.org/docs/getting-started/upgrading

[112] (2016, Nov.) GitHub – Node-RED Dashboard. [Online]. https://github.com/node-red/node-red-dashboard

[113] u. Jochen Scheib. (2016, Nov.) Node-RED – hueplus. [Online]. http://flows.nodered.org/node/node-red-contrib-hueplus

[114] (2016, Nov.) Fritzing. [Online]. http://fritzing.org/

[115] (2016, Nov.) BOXTEC. [Online]. http://playground.boxtec.ch/doku.php/wireless/esp8266

[116] (2016, Nov.) DOIT.am. [Online]. http://doit.am/

[117] jambox_josh. (2014, Apr.) Pixeljoint. [Online]. http://pixeljoint.com/forum/forum_posts.asp?TID=18755

[118] (2016, Nov.) Blogspot – 4x4 Pixel Art. [Online]. http://4x4pixels.blogspot.de/

[119] (2016, Nov.) Adlerweb. [Online]. https://www.adlerweb.info/blog/wp-content/uploads/2016/03/esp01-deepsleep.png

[120] (2016, Nov.) Adafruit – Website. [Online]. https://www.adafruit.com/

# Anhang

## I. Bezugsquellen der Einzelteile

Espruino
http://www.watterott.com/en/Espruino-Pico-Pinned

Raspberry Pi 3 Model B
Relais 5V
Temperatursensor DHT11 Waveshare
Lichtsensor
Abstandssensor
ESP8266 ESP-01
USB to RS232 Serial TTL Module PL2303TA USB UART PL-2303TA
https://eckstein-shop.de/

ESP8266 ESP-12E Dev board
https://www.amazon.de/ELEGIANT-NodeMcu-ESP8266-ESP-12E-Development/dp/B018E741G4/

AWS IoT Button
Über Amazon, Lieferung aber nur in die USA
https://aws.amazon.com/de/iot/button/

Philips Hue White & Color 3rd Generation Starter Set
https://www.amazon.de/Philips-Starter-inklusive-Generation-app-gesteuert/dp/B01LZ8QYPI/