Vor- und Nachteile von MQTT gegenüber HTTP in einem Praxisbeispiel

-

Untersuchung alternativer Kommunikationsprotokolle für Biene40

Zitiervorschlag: Langen, Jan; Sahler, Kerstin; Brell, Claus (2023) *Vor- und Nachteile von MQTT gegenüber HTTP in einem Praxisbeispiel. Arbeitsbericht Nr. 8 / Biene40. Polykopie, im Druck.*

Langen, Jan*; Sahler, Kerstin*; Brell, Claus*

*Hochschule Niederrhein, Mönchengladbach

Dezember 2023

Die Überwachung von Bienenstöcken durch Sensoren ermöglicht eine kostengünstige, zeitnahe und minimalinvasive Beurteilung des eigenen Bienenvolkes. Sensoren, die mit Kabel in die Beute eingebracht werden, stören jedoch die Betriebsweisen in der Imkerei. Verschiedene technische Lösungsansätze bieten sich an, um Daten kabellos über Kommunikationstechnologien des IoT (Internet of Things) aus der Beute zu transportieren (vgl. Brell, 2022). Für die Datenübertragung im Biene-4.0-Projekt wird HTTP (Hypertext Transfer Protocol) als Kommunikationsprotokoll genutzt. Eine Alternative hierzu bietet MQTT. Diese Ausarbeitung untersucht die Unterschiede von HTTP und MQTT an einem Beispiel und ob MQTT im Projekt Biene40 eine Alternative für HTTP sein kann.

1 Ziel und Forschungsfragen

Ziel ist es, die Vor- und Nachteile des HTTP- und des MQTT-Netzwerkprotokolls gegenüberzustellen und eine Handlungsempfehlung für Biene40 abzuleiten. Dazu werden neben einer Literaturanalyse zu MQTT im Imkereikontext und zu HTTP zwei Prototypen entwickelt. Daneben entsteht eine Anleitung zur Implementierung einer beispielhaften MQTT-Lösung. Auf Basis der Artefakte werden die Vor- und Nachteile von MQTT und HTTP untersucht. Folgende handlungsleitende Forschungsfragen lassen sich aus dem Ziel ableiten:

F1: Was ist MQTT und wie unterscheidet sich MQTT und HTTP?

F2: Welche Anwendungsdomänen sind geeigneter für MQTT und welche für HTTP?

F3: Welche Vor- und Nachteile ergeben sich aus der praktischen Umsetzung?

F4: Welche der beiden Möglichkeiten der Implementierung ist besser geeignet für eine praktische Anwendung in der Imkerei?

2 Methodik

Als Methoden werden die Literaturanalyse in Kombination mit Design Science Research genutzt.

Für die Literaturrecherche zu den Eigenschaften von MQTT und HTTP wird unsystematisch nach dem Schneeballsystem vorgegangen. Als Start der

Recherche dient die Literaturrecherche in Arbeitsbericht Nr. 3 des Projektes Biene40 (Wurm&Brell 2022)

Die Methodik der Prototypen orientiert sich am gestaltungsorientierten Forschungsansatz, der dem designorientierten Forschungsansatz nach Hevner (vgl. Hevner et al., 2004, S.75-100) zugeordnet werden kann.

Ziel ist es, Prototypen zu bauen, die über MQTT bzw. HTTP kommunizieren können. Im Konkreten wird hierzu ein Bewegungsmelder an einen Mikrocontroller angeschlossen, der über Internet-Services eine Funksteckdose schaltet, die sowohl das MQTT- als auch das HTTP-Protokoll beherrscht. Dieser Bewegungsmelder könnte beispielsweise am Bienenstock genutzt werden, um Diebstahl eine Bewegung der Beute durch Wildtiere zu detektieren und anschließend eine Überwachungskamera einzuschalten. Dabei sollen sich die Prototypen an den Zielen des Biene 4.0 Projektes orientieren - einfache, kostengünstige und minimalinvasive Unterstützung der Betriebsweise in der Imkerei (vgl. Brell, o.D.-a.). Der erstellte Prototyp wird anschließend in gesicherten WLAN-Netzwerken getestet und entsprechend optimiert.

3 Ergebnisse der Recherche nach MQTT und HTTP

3.1 Eigenschaften von MQTT und HTTP im Kontext von IoT

IoT (Internet of Things) beschreibt die Vernetzung verschiedener Entitäten über das Internet. Diese können so miteinander interagieren und die Durchführung von Aufgaben erleichtern (vgl. Brell, o.D.-b.). IoT besteht im Wesentlichen aus drei Komponenten: den Geräten, Anwendungen und dem Kommunikationskanal. Die Geräte-Komponente umfasst Sensoren, Controller und Aktoren, die direkt mit der physischen Umwelt interagieren (vgl. Herrero, 2022, S.7). Im Gegensatz dazu sind die Anwendungen dafür verantwortlich, die generierten Daten zu verarbeiten und sie entweder visuell zu präsentieren oder sie einer Analyse zu unterziehen, um Erkenntnisse zu gewinnen und potenzielle Handlungen abzuleiten (vgl. Herrero, 2022, S.8). Der Kommunikationskanal ist das Medium, das die Übertragung zwischen den Geräten und den entsprechenden Anwendungen ermöglicht (vgl. Herrero, 2022, S.7).

Für den Kommunikationskanal sind Kommunikationsprotokolle von erheblicher Bedeutung. Sie dienen als Verständigungsmittel im Internet, indem sie Sprach- und Handlungsregeln für den Datenaustausch bereitstellen (vgl. Abts, 2015, S.15).

Schichtenmodelle wie das OSI-7-Schichtenmodell oder korrespondierend das Internet-Modell beschreiben die Datenkommunikation auf verschiedenen Ebenen. In den Ebenen 4 und 3 des OSI-Modells und des Internet-Modells ist TCP/IP-angesiedelt mit der Differenzierung: Verbindungsschicht, Vermittlungsschicht, Transportschicht. Darüber befindet sich die Anwendungsschicht, die die Schichten 5-7 des OSI-Modells zusammenfasst. In jeder Schicht gibt es verschiedene Kommunikationsprotokolle, die aufeinander abgestimmt sind (vgl. Abts, 2015, S.15 f.).

Sowohl HTTP als auch MQTT sind Protokolle der Anwendungsschicht. Ihre verschiedenen Eigenschaften werden im Folgenden auf Basis der Architektur, Nachrichtenadressierung, Funktionsweise, Aufbau von Nachrichten, Kommunikation,

persistenten Verbindungen, Datentypen, Übertragungsqualität, Anwendungsbereiche und Verbreitung beschrieben.

3.2 HTTP

HTTP (Hypertext Transfer Protocol) wurde bereits 1990 als Grundlage für die Kommunikation im World Wide Web (WWW) entwickelt (Vgl. RFC, 1999).

3.2.1 Architektur

HTTP basiert auf einem Request/Response-Prinzip mit einer Client-Server-Architektur (vgl. RFC, 1999). Diese ist in Abbildung 1 dargestellt.

3.2.2 Nachrichtenadressierung

Die Adressierung von Nachrichten erfolgt bei HTTP auf Basis des Uniform Resource Identifiers (URI) und beinhaltet alle Grundsätze und Regeln, die mit der Verwendung einhergehen (vgl. RFC, 1999). Eine Ressource wird über einen Uniform Resource Locater (URL) adressiert. Dabei handelt es sich um eine standardisierte Adresse, die sich aus verschiedenen Teilen zusammensetzt. Neben dem verwendeten Protokoll und dem adressierten Server (Hostname oder IP), können außerdem eine Portnummer oder die Bezeichnung der Ressource Teil der URL sein (vgl. Abts, 2015, S.161).

3.2.3 Aufbau von HTTP-Nachrichten

HTTP-Nachrichten setzen sich aus Kopfdaten (Header) und dem Nachrichteninhalt (Body) zusammen (vgl. RFC 1999). Eine HTTP-Anfrage besteht dabei aus einer Kopfzeile, Anfrageparametern, einer Leerzeile und einem Nutzdatenanteil, wobei die Anfrageparameter und der Nutzdatenanteil optional sind. Die Kopfzeile der Anfrage enthält die HTTP-Methode (GET, POST, HEAD, PUT und DELETE), den Ressourcennamen und die Protokollversion (vgl. Abts, 2015, S.164 f.). Eine HTTP-Antwort wiederum besteht aus einer Kopfzeile, Antwortparametern, einer Leerzeile und einem Nutzdatenanteil. Auch bei der HTTP-Antwort sind die Anfrageparameter und der Nutzdatenanteil optional. Der Kopfteil setzt sich aus der Protokollversion, einem Status-Code und einer Status-Meldung zusammen. Die Status-Meldung ist ebenfalls optional (vgl. Abts, 2015, S.168). Die Kopfdaten haben eine Größe von mindestens acht Byte. Es gibt keine maximale Nachrichtengröße (vgl. Craggs, 2022).

3.2.4 Funktionsweise von HTTP

Ein Client stellt bei HTTP über eine TCP-Verbindung eine Anfrage an den Server (HTTP-Request) und fragt mithilfe der URL eine Ressource an. Der Server verarbeitet diese Anfrage und sendet die Ressource oder eine Fehlermeldung (HTTP-Response) an den Client. Nachdem der Client die Antwort verarbeitet hat, wird die TCP-Verbindung wieder geschlossen (vgl. Abts, 2015, S.162). Daraus ergibt sich eine One-to-One-Kommunikation (vgl. Craggs, 2022). Bei HTTP handelt es sich um ein zustandsloses (stateless) Protokoll (vgl. RFC 1999). Jede Anfrage wird unabhängig von anderen Transaktionen vom Server bearbeitet, da er keinerlei Kenntnis über vorherige Anfragen des Clients hat (vgl. Abts, 2015, S.162). Dies kann umgangen werden, indem Cookies eingesetzt werden. Diese Add-on Mechanismen beinhalten einen String, der IoT-Geräten ermöglicht verschiedene Anwendungen zu identifizieren (vgl. Herrero, 2022, S.118).

3.2.5 Kommunikation

Resultierend aus dem Request/Response-Prinzip erfolgt die Kommunikation bei HTTP synchron. Für jede Anfrage des Clients wird eine Antwort übertragen (vgl. Herrero, 2022, S.111).

3.2.6 Persistente Verbindung

Seit der Version HTTP/1.1 unterstützt HTTP persistente Verbindungen. Mehrere Anfragen können dabei über eine einzelne TCP-Verbindung übertragen werden, solange diese geöffnet ist (vgl. Abts, 2015, S.163). Daraus resultierend besteht bei HTTP die Möglichkeit, einzelne Request-Response-Transaktionen oder eine persistente TCP-Verbindung zu nutzen (vgl. Herrero, 2022, S.114).

3.2.7 Datentypen

Die Datenübertragung bei HTTP ist textbasiert (vgl. Trojan, 2017, S.20). Somit liegen die Daten, die übertragen werden, in einem für den Menschen lesbaren Textformat vor. Dies erleichtert die Fehlersuche und das Debugging über Netzwerk-Sniffer (vgl. Herrero, 2022, S.116). Binäre Dateien wie Bilder werden beispielsweise mithilfe der Base64-Kodierung übertragen (vgl. Trojan, 2017, S.20).

3.2.8 Übertragungsqualität

HTTP basiert auf dem TCP/IP Protokoll (vgl. Abts, 2015, S.161) und nutzt somit dessen Übertragungsqualität (vgl. Trojan, 2017, S.16). TCP als Transportprotokoll ermöglicht die zuverlässige Übermittlung von Datenpaketen in der richtigen Reihenfolge und initiiert eine erneute Übertragung von Paketen, die verloren gegangen oder beschädigt wurden (vgl. Abts, 2015, S.43).

3.2.9 Anwendungsbereiche

Da HTTP ursprünglich für das WWW für Menschen. Im Kontext von IoT eignet sich HTTP insbesondere für Anwendungen mit wenigen Datenübertragungen pro Zeiteinheit (vgl. Craggs, 2022).

3.2.10 Verbreitung

HTTP ist die Basis des WWW und daher weit verbreitet. Damit geht sowohl eine verbreitete Expertise in der Implementierung von HTTP-Lösungen als auch eine leichte Verfügbarkeit von Softwarebibliotheken für Clients und Server einher. Entwickler können auf vorgefertigte Softwarekomponenten zurückgreifen, anstatt diese von Grund auf neu entwickeln zu müssen (vgl. Craggs, 2022).

3.2.11 Performance

Der Overhead von HTTP ist im Vergleich zu anderen Protokollen im IoT-Bereich größer. Dies ergibt sich aus der Tatsache, dass HTTP ursprünglich für die Übertragung von Dokumenten und nicht für wenige Bytes entwickelt wurde. HTTP ist bei seltenen Nachrichtenübertragungen, bei der jede neue Nachricht einen neuen Verbindungsaufbau erfordert, oder bei regelmäßigen Übertragungen mit langen Phasen der Inaktivität gegenüber anderen Protokollen zu bevorzugen (vgl. Šikić et al., 2020, S.85). Der Energieverbrauch und die Latenzzeit, also die Zeit von der Nachrichtenerstellung bis zur Nachrichtenauslieferung sind bei HTTP größer als bei anderen Protokollen (vgl. Nicholas, 2012, Šikić et al., 2020, S.86). Im Gegenzug dazu ist der Verbrauch des Speicherplatzes bei HTTP niedriger im Vergleich zu anderen IoT-Protokollen (vgl. Kakakhel et al., 2019, S.214).

3.3 MQTT

MQTT (Message Queing Telemetry Transport) wurde als Anwendungsprotokoll 1999 von IBM entwickelt (vgl. Trojan, 2017, S.11). Seit 2010 steht MQTT lizenzfrei zur Verfügung. 2014 hat OASIS mit MQTT 3.1.1 einen Standard für MQTT definiert (vgl. Trojan, 2017, S.20). Seit 2018 ist der Nachfolger MQTT 5 als Standard verfügbar (vgl. Herrero, 2022, S.138). In Anlehnung an die Anforderungen des IoT handelt es sich bei MQTT um „ein leichtgewichtiges, ereignis- und nachrichtenorientiertes

Protokoll zur effizienten und asynchronen Kommunikation zwischen Geräten auch über limitierte Netzwerke.“ (Trojan, 2017, S.12)

3.3.1 Architektur

Die Kommunikation in MQTT erfolgt nach einem nachrichtenbasierten Ansatz und folgt dem Publish/Subscribe-Prinzip (vgl. Abbildung 2) (vgl. Trojan, 2017, S.12.; vgl. Herrero, 2022, S.137). Diese zeichnet sich durch einen zusätzlichen Broker aus (vgl. Fritz et al., 2021, S.70). Durch die Nutzung des Publish/Subscribe-Prinzips kann die Latenz und die Auslastung des Netzwerks verbessert werden. Gleichzeit ist die Zuverlässigkeit des gesamten Systems stark von der Verfügbarkeit des Brokers abhängig (vgl. Herrero, 2022, S.111).

3.3.2 Funktionsweise mit Publisher, Broker und Topics

Publisher. Publisher initiieren die Veröffentlichung von Nachrichten auf bestimmten Kanälen (sogenannten Topics), indem sie Nachrichten an den Broker übermitteln (vgl. Herrero, 2022, S.137).

Topics. Topics sind eine Art Betreff der Nachricht (vgl. Trojan, 2017, S.13). Ein Topic ist ein UTF-8-String, der, ähnlich einer URL, verschiedene Hierarchiestufen aufweisen kann, aber nicht muss (vgl. Trojan, 2017, S.14).

Broker. Der Broker agiert als zentrale Vermittlungsinstanz (vgl. Fritz et al., 2021, S.70). Er aggregiert eingehende Nachrichten von den Publishern, archiviert sie und leitet sie als Benachrichtigung an die jeweiligen Subscriber weiter. Subscriber, die sich für ein Topic registriert haben, empfangen die vom Broker weitergeleiteten Nachrichten, die ursprünglich von den Publishern übermittelt wurden (vgl. Herrero, 2022, S.137). Ein Subscriber kann sich parallel für verschiedenen Topics anmelden und ein Topic kann mehrere Subscriber haben (vgl. HiveMQ, 2023-a). Publisher können nicht nachvollziehen, an wie viele Subscriber die Nachricht weitergeleitet wird und andersrum können Subscriber nicht nachvollziehen, wie viele Publisher oder welche Publisher Nachrichten veröffentlichen. Es erfolgt durch den Broker eine Entkopplung von Publisher und Subscriber. Daraus resultiert eine einfache Skalierbarkeit bei der Implementierung von MQTT (vgl. Fritz et al., 2021, S.70). Weiterhin ergibt sich so eine sogenannte One-to-Many-Kommunikation (Vgl. Craggs, 2022).

3.3.3 Aufbau von Nachrichten, Header und Payload

Nachrichten bestehen bei MQTT aus dem Fixed Header, dem Variable Header und dem Payload. Der Fixed Header wird für alle Pakete benötigt, während der Variable Header und der Payload nur in manchen Pakettypen erforderlich ist. Insgesamt gibt es 14 verschiedene Pakettypen.

Der Fixed Header benötigt zwei Bytes (vgl. Trojan, 2017, S.20). Die Größe des Variable Headers und des Payloads sind abhängig vom Pakettyp und den verschiedenen Parametern, die übermittelt werden sollen (vgl. OASIS, 2014). Die maximale Nachrichtengröße beträgt 256 MB (vgl. Craggs, 2022).

3.3.4 Asynchrone Kommunikation

Die Publish/Subscriber-Architektur von MQTT ermöglicht eine asynchrone Kommunikation, bei der Publisher und Subscriber nicht gleichzeitig aktiv sein müssen (vgl. Herrero, 2022, S.137). Der Broker kann die letzte gesendete Nachricht des Publishers, die über ein Retain-Flag gekennzeichnet wurde, abspeichern und an den später erst aktiven Subscriber ausliefern. Dabei kann der Subscriber ebenfalls durch den Retain-Flag nachvollziehen,

dass es sich hierbei um eine aufbewahrte Nachricht handelt. Die Nachricht wird vom Broker aufbewahrt, bis er eine neue Nachricht mit Retain-Flag in demselben Topic erhält (Vgl. Trojan, 2017, S.18).

3.3.5 Persistente Verbindung über mehrere Nachrichten

Es ist möglich mit MQTT persistente Sessions zwischen Subscriber und Broker zu erzeugen (vgl. HiveMQ, o.D.-a). Wenn ein Endpunkt eine persistente Session mit einem Broker herstellt, speichert der Broker die Abonnementinformationen sowie eventuell nicht zugestellte Nachrichten, die für den Subscriber bestimmt sind. Auf diese Weise kann der Endpunkt, wenn die Verbindung unterbricht und später erneut hergestellt wird, die Kommunikation nahtlos fortsetzen. Persistente Sessions ermöglichen somit die Aufrechterhaltung des Kommunikationszustands eines Endpunkts über Verbindungsunterbrechungen hinweg (vgl. HiveMQ, 2023-b).

3.3.6 Beliebige Datentypen

MQTT ist datenunabhängig. Neben binären Daten können auch Texte, XML- und JSON-Strukturen übertragen werden (vgl. Trojan, 2017, S.13).

3.3.7 Übertragungsqualität

MQTT basiert ebenfalls auf dem TCP/IP Protokoll (vgl. Fritz et al., 2021, S.70). Darüber hinaus wird von MQTT eine abgestufte Qualität mit Liefergarantie für unzuverlässige Netze angeboten (vgl. Trojan, 2017, S.16). Es gibt insgesamt drei verschiedene Quality-of-Service-Stufen (QoS) bei MQTT: QoS 0, QoS 1 und QoS 2. Die QoS-Stufen werden im Vorfeld vom Sender bzw. Empfänger definiert und sind neben anderen Faktoren auch abhängig von der Netzqualität. Daraus resultierend benötigt ein Sender, der sich in einem stabilen Hausnetzwerk befindet, nur eine QoS-Stufe von 0. Ein Empfänger, der über Mobilfunk verbunden ist, braucht mindesten eine QoS-Stufe von 1 (vgl. Trojan, 2017, S.17).

3.3.8 Anwendungsbereiche

MQTT wurde speziell für den Einsatz in Umgebungen mit begrenzter Übertragungsgeschwindigkeit entwickelt. Aufgrund seiner einfachen Struktur und geringen Datenübertragungsbelastung wird MQTT für die Kommunikation in IoT-Netzwerken eingesetzt. Die Geräte in diesen Szenarien haben in der Regel eine geringe Rechenleistung und begrenzten Speicherplatz (vgl. Herrero, 2022, S.138).

Aufgrund der drei Qualitätsstufen und der persistenten Sessions bietet sich der Einsatz von MQTT auch für Anwendungsbereiche mit instabilen Netzen an (vgl. Craggs, 2022).

3.3.9 Verbreitung

MQTT ist im Vergleich zu HTTP weniger weit verbreitet. In den letzten Jahren gewinnt MQTT durch vielfältige IoT-Szenarien an Bedeutung (vgl. Craggs, 2022).

3.3.10 Performance

MQTT ist auf eine effiziente Übertragung binärer Daten ausgelegt. Der einzelne Datentransfer selbst erzeugt wenig Overhead. Allerdings ist beim Verbindungsaufbau bei MQTT eine hohe Netzauslastung zu beobachten (vgl. Šikić et al., 2020, S.85). Im Vergleich zu anderen Netzwerkprotokollen ist der Energieverbrauch und die Latenzzeit bei MQTT niedriger (vgl. Šikić et al., 2020, S.86). MQTT ist ein leichtgewichtiges Protokoll. Aufgrund seiner persistenten Nachrichten- und Verbindungsstruktur benötigt es jedoch ggf. mehr Speicherplatz (vgl. Kakakhel et al., 2019, S.214).

	MQTT	**http**
Ursprung	IBM, 1999	1990
Technische Spezifikation	OASIS, ISO/IEC 20922	RFC 7231
Schicht	Anwendungsschicht	Anwendungsschicht
Transportprotokoll	TCP/IP	TCP/IP
Architektur	Publish/Subscribe-Prinzip	Request/Response-Prinzip
Adressierung	Topics	URI (URL)
Header	Zwei Byte Fixed Header und Variable Header abhängig vom Pakettyp	*Anfrage*: HTTP-Methode, Ressourcennamen und Protokollversion, *Antwort*: Protokollversion, Status-Code und Statusmeldung, insgesamt mind. Acht Byte
Funktionsweise	Publisher sendet Nachricht auf Topic an den Broker, Broker verwaltet die Nachricht und leitet sie an alle Subscriber weiter, die das Topic abonniert haben -> One-to-Many	Ein Client fragt beim Server (HTTP-Request) eine Ressource an. Der Server verarbeitet diese Anfrage und sendet die Ressource oder eine Fehlermeldung (HTTP-Response) zurück -> One-to-One
Kommunikation	Asynchron	Synchron
Persistente Verbindung	Persistente Sessions	Persistente Verbindung seit HTTP 1.1
Zustand	persistente Sessions (Broker speichert Zustand von Subscriber je nach QoS)	Eigentlich zustandslos, kann über Cookies ergänzt werden
Datentypen	alle Datentypen	textorientiert
Übertragungsqualität	Zusätzlich zu TCP/IP drei QoS	entsprechend TCP/IP
Anwendungsbereiche	IoT, insbesondere bei instabilen Netzen und häufigen Datenübertragungen	WWW, IoT, insbesondere Anwendungsfälle mit wenig Datenübertragungen
Verbreitung	derzeit zunehmend	bereits weit verbreitet durch WWW
Performance	Weniger Overhead, weniger Latenz, geringerer Energieverbrauch, mehr Speicherverbrauch	mehr Overhead, mehr Latenz, höherer Energieverbrauch, weniger Speicherverbrauch
Demonstratorcode* Nachrichtengröße	61 Byte (ohne Berücksichtigung von Topic und Payload)	252 Byte (ohne Berücksichtigung von Host, URI und Nachricht)
Demonstratorcode* Quelltextzeilen	265	167

Tabelle 1: Gegenüberstellung der Eigenschaften von MQTT und HTTP; (*) bezeichnet den Quellcode für die hier entwickelten Prototypen.

Name des Bauteils	conrad.de	voelkner.de	reichelt.de	kombiniert
ESP 8266 D1 Mini (zum Auflöten)	9,49 €	6,49 €	7,99 €	6,49 €
Joy-it SEN-HC-SR501 Sensor	3,99 €	3,99 €	2,30 €	2,30 €
Shelly Plus Plug S	23,68 €	23,48 €	30,95 €	23,48 €
Gesamt	37,16 €	33,96 €	41,24 €	32,27 €

Tabelle 2: Bauteile und Preise (Stand: 29.06.2023); Quelle: (Vgl. Machhamer et al., 2020; vgl. EHI Retail Institute GmbH, 2022; vgl. Conrad Electronic SE, o.D.-a.; vgl. Re-In Retail International GmbH, o.D.-a.; vgl. Reichelt Elektronik GmbH & Co KG, o.D.-a.; vgl. Conrad Electronic SE, o.D.-b.; vgl. Re-In Retail International GmbH, o.D.-b.; vgl. Reichelt Elektronik GmbH & Co KG, o.D.-b.; vgl. Conrad Electronic SE, o.D.-c.; vgl. Re-In Retail International GmbH, o.D.-c.; vgl. Reichelt Elektronik GmbH & Co KG, o.D.-c.

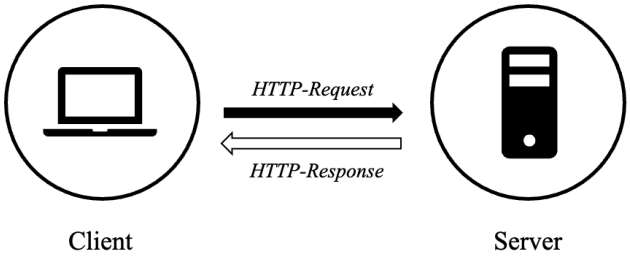

Abbildung 1: Request/Response-Prinzip; Quelle: Eigene Darstellung in Anlehnung an Abts, 2015, S.162.

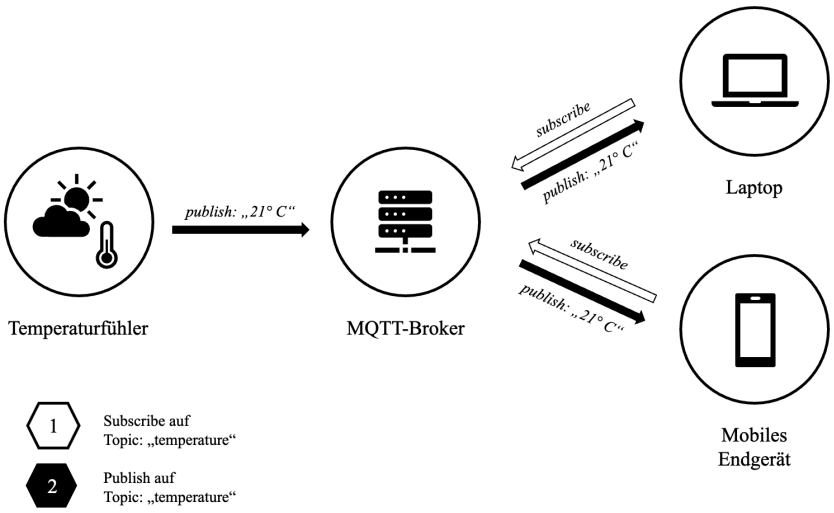

Abbildung 2: MQTT Publish/Subscribe-Prinzip; Quelle: Eigene Darstellung in Anlehnung an Trojan, 2017, S.13.

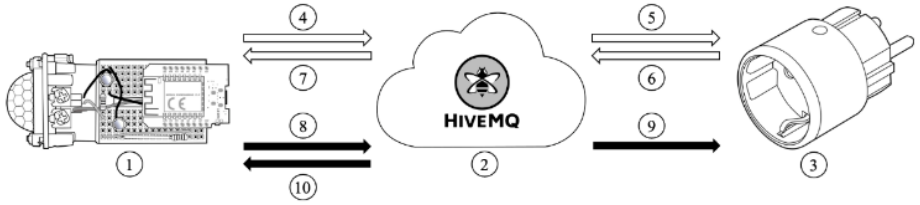

Abbildung 3: Topologie des MQTT-Prototyps; Quelle: Eigene Darstellung. Es bedeuten: (1) Der Mikrocontroller ESP8266 D1 Mini, als auch (3) die Steckdose Shelly Plus Plug S senden und empfangen Nachrichten als Publisher bzw. Subscriber. Der ESP8266 D1 Mini (4) veröffentlicht eine „status update"-Nachricht auf dem Topic „MQTT-Prafix/status/switch:0". So wird die Nachricht an den (2) HiveMQ Cloud Broker gesendet. (5) Dieser leitet die Nachricht weiter an die Shelly Steckdose, die das Topic abonniert hat. (6) Die Steckdose veröffentlicht den Status im JSON-Format auf dem gleichen Topic. (7) Der Broker leitet den Status weiter an den ESP8266 D1 Mini. (8) Abhängig vom Auslösen des Sensors am ESP8266 D1 Mini veröffentlicht dieser auf dem Topic „MQTT-Praefix/command/switch:0" eine Nachricht mit „on" bzw. „off". Der Broker leitet die Nachricht weiter (9) an die Shelly Steckdose und (10) den ESP8266 D1 Mini, die beide das Topic abonniert haben.

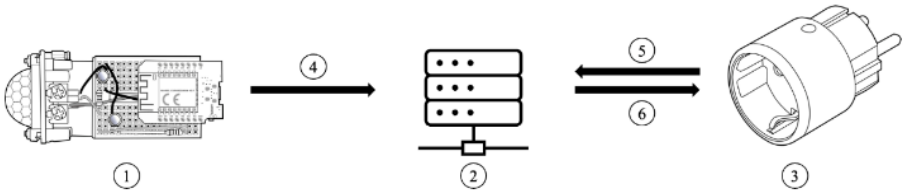

Abbildung 4: Topologie des HTTP-Prototyps; Quelle: Eigene Darstellung. Es bedeuten: (1) Der Mikrocontroller ESP8266 D1 Mini sendet (4) eine Nachricht mit dem Inhalt „on" oder „off" an den (2) Webserver. Dieser ändert den Dateieintrag in der Datei „status.txt". Der (3) Shelly Plus Plug S fragt (5) den Status zyklisch beim Webserver an und erhält (6) eine Antwort. Anschließend schaltet sich der Shelly Plus Plug S in den gewünschten Status.

4 Ergebnisse zum Bau der Prototypen

4.1 Verwendete Materialien und Anbietervergleich

Es werden zwei Prototypen auf Basis eines Microcontrollers (ESP8266 D1 Mini) entwickelt, welche mittels eines Bewegungssensors (PIR) einen WLAN-Steckdose (Shelly Plus Plug S) an- bzw. ausschalten (vgl. Mahamud et al., 2019, Machhamer et al., 2020). Die benötigten Komponenten (für je ein System) stehen in Tabelle 2. Übliches Verbrauchsmaterial wie Stiftleisten, Jumper-Kabel, LEDs, Widerstände, Micro-USB-Kabel sowie Breadboard ist nicht aufgelistet. Die Vergleichbarkeit wird durch die Verwendung jeweils identischer Hardware gewährleitet.

Für die Preisbestimmung der Prototypen werden große umsatzstarke Onlineshops (conrad.de, voelkner.de und reichelt.de, vgl. EHI Retail Institute GmbH, 2022) in Deutschland betrachtet.

Der Mikrocontroller ESP8266 D1 Mini und der Joy-it Bewegungssensor sind zusammen bei allen Onlineshops zu einem Preis zwischen 10,29 € und 13,48 € bestellbar. Je nach Anwendungsszenario kann die schaltbare Steckdose „Shelly (Plus) Plug S“ zu einem Preis von ca. 30 € mitbestellt werden (vgl. Tabelle 2).

Es ist auch möglich, auf einem Raspberry Pi einen MQTT-Broker selbst zu betreiben. Im Rahmen dieser Ausarbeitung wurde ein Selbstbau des Brokers nicht verfolgt und ein gängiger MQTT-Cloud-Broker verwendet.

Es gibt eine Vielzahl von MQTT-Cloud-Brokern auf dem Markt. Eine Übersicht bietet das Github-Repository von mqtt.org (Übersicht über verschiedene MQTT-Broker: https://github.com/mqtt/mqtt.org/wiki/server-support#capabilities, Zugriff am 30.06.2023). Die Wahl fiel auf den HiveMQ Broker. Dieser wurde in einigen der untersuchten Quellen erwähnt und unterstützt bis zu 10.000.000 MQTT-Clients pro Cluster (vgl. HiveMQ, o.D.-b.). HiveMQ wird in drei verschiedenen Versionen angeboten: HiveMQ Enterprise, HiveMQ Professional und HiveMQ Community (vgl. HiveMQ, o.D.-c.). Die frei verfügbare serverless Variante (Community Variante) bietet kostenlos Anbindung von bis zu 100 Clients und 10 GB Datentransfer (vgl. HiveMQ, o.D.-d.) und bietet ausreichend Kapazität für den MQTT-Prototyp.

Für den Aufbau des Referenzsystems über HTTP wird ein Webserver im Internet benötigt. Dieser kann bei Anbietern, wie Strato oder Domainfactory preiswert monatlich gemietet werden, hier kam ein Webserver des Letztautors für das Projekt Biene40 und für den Einsatz in der Hochschullehre zum Einsatz.

4.2 Aufbau des MQTT-Prototyps

Um den MQTT-Prototypen aufzubauen, werden der ESP8266 D1 Mini, die HiveMQ Cloud und der Shelly Plug Plus S eingerichtet. Anschließend werden der Shelly Plug Plus S als Subscriber und der ESP8266 D1 Mini als Publisher angebunden. Die Topologie ist in Abbildung 3 dargestellt.

4.2.1 Einrichten des ESP8266 D1 Mini

Zunächst werden Vorbereitungen getroffen, um den ESP8266 D1 Mini über die Arduino IDE (integrierte Entwicklungsumgebung) programmieren zu können. Hierzu zählt auch die Installation

eines Treibers für den USB-to-Serial Chip CH340G des D1 Mini (vgl. makesmart, 2020).

Darüber hinaus wird die Entwicklungsumgebung von Arduino heruntergeladen. Hierbei ist mit der Version 1.8.19 eine ältere Version der Arduino IDE zu wählen, da manche Bibliotheken, die für die Einbindung von MQTT benötigt werden, noch nicht mit aktuelleren Versionen kompatibel sind. Diese ältere Version wird ebenfalls von Arduino bereitgestellt.

Anschließend werden verschiedene Einstellungen in der Arduino IDE vorgenommen, um den ESP8266 D1 Mini in der Entwicklungsumgebung verwenden zu können. Neben dem Hinzufügen eines zusätzlichen Boardverwalters, wird außerdem ein ESP8266 Paket installiert. Anschließend wird der ESP8266 D1 Mini als Board eingestellt und der entsprechende Port des angeschlossenen ESP8266 D1 Mini ausgewählt (vgl. makesmart, 2020).

Um zu überprüfen, ob die Einrichtung erfolgreich war, kann ein Beispielprogramm der Arduino IDE namens „Blink“ ausgeführt werden. Bei erfolgreicher Einrichtung blinkt die interne Leuchte des ESP8266 frequenzartig auf (vgl. makesmart, 2020).

Zusätzlich wird die ArduinoJson Bibliothek in der Arduino IDE hinzugefügt. Die Funktionen dieser Bibliothek werden im weiteren Verlauf benötigt, um JSON-Dateien deserialisieren zu können, und so Werte, die in der Datei enthalten sind, auszulesen (vgl. makesmart, 2021).

Bei der Kalibrierung des PIR Bewegungssensors wird die Impulsrate auf einmaliges Auslösen eingestellt. Die Empfindlichkeit und Haltezeit werden auf die höchste Einstellung gestellt (vgl. SIMAC Electronics GmbH, o.D.).

4.2.2 Einrichten der HiveMQ Cloud

Nach der Erstellung eines Accounts bei HiveMQ, wird die von HiveMQ empfohlene PubSubClient-Bibliothek heruntergeladen und in der Arduino IDE eingebunden (vgl. HiveMQ, o.D.-e.).

Für die Nutzung der TLS-Zertifikate werden Zeitfunktionen benötigt. Die NTPClient-Bibliothek stellt eben solche Funktionalitäten für Zeit und Datum bereit. Dementsprechend wird die Bibliothek heruntergeladen und in der Entwicklungsumgebung hinzugefügt (vgl. HiveMQ, o.D.-e.).

Um sich sicher mit der HiveMQ Cloud verbinden zu können, wird im nächsten Schritt der LittleFS Filesystem Uploader heruntergeladen und in der Arduino IDE eingerichtet. Mithilfe des LittleFS Dateisystems können Sicherheitszertifikate auf den Microcontroller hochgeladen werden (vgl. HiveMQ, o.D.-e.).

Die Sicherheitszertifikate selbst können über ein bereitgestelltes Skript erstellt werden. Dieses Skript wird über Visual Studio Code als Entwicklungsumgebung ausgeführt, da das Skript in der Programmiersprache Python verfasst ist. Auch andere Entwicklungsumgebungen, die Python unterstützen, können bei der Erstellung der Sicherheitszertifikate genutzt werden. Eine weitere Entwicklungsumgebung, die hier beispielhaft zu nennen ist, ist PyCharm. Im nächsten Schritt werden die Zertifikate über das LittleFS Dateisystem auf das Board geladen (vgl. HiveMQ, o.D.-e.).

Der letzte Schritt bei der Einrichtung der HiveMQ Cloud ist das Erstellen von Credentials, die eine zusätzliche Absicherung beim Zugriff auf die Cloud ergänzen. Diese können im Nutzerbereich

von HiveMQ im Reiter „Access Management“ erstellt werden.

4.2.3 Einrichten des Shelly Plus Plug S

Für die Einrichtung des Shelly Plus Plug S gibt es zwei Optionen: Über die Cloud-Services des Herstellers oder über den Webserver auf dem Device.

Cloud Services des Herstellers. Die erste Option erfolgt über die vom Hersteller bereitgestellte mobile Applikation „Shelly Smart Control“. Die App wird heruntergeladen und ein Nutzer-Account erstellt. Zur Einbindung des Gerätes in der App wird ein neuer Raum hinzugefügt. Sobald der Shelly Plus Plug S in die Steckdose gesteckt wird, kann das Gerät über das Dashboard ausgewählt werden.

Webserver auf dem Device. Bei der zweiten Option wird der Webserver des Shelly genutzt und ist es nicht notwendig die Applikation herunterzuladen und ein Benutzerkonto zu erstellen. Ein PC wird über den Access-Point des Shellys mit dem Shelly verbunden. Über die IP 192.168.33.1 kann das Access-Point-Dashboard des Shellys geöffnet und die Einstellungen entsprechend angepasst werden (vgl. Shelly, o.D.-a.).

4.2.4 Anbinden des Subscribers

Um den Shelly Plus Plug S als Subscriber über MQTT an den Broker anzubinden, werden die MQTT-Einstellungen des Geräts angepasst. Hierzu wird über die IP-Adresse des Geräts über den Browser direkt auf die Einstellungen zugegriffen und die Cluster-URL der HiveMQ Cloud, die zuvor erstellten Credentials als auch ein MQTT-Präfix in den MQTT-Einstellungen ergänzt. Das MQTT-Präfix entspricht dem Topic, auf dem Nachrichten an den Shelly Plus Plug S gesendet werden. Weiterhin werden „MQTT Control“ als auch „Generic status update over MQTT“ aktiviert (vgl. Shelly, o.D.-b.).

4.2.5 Anbinden des Publishers

Für die Anbindung des Publishers (der hier erstellet Prototyp) an den Broker (Hive MQ Cloud) wird ein Code in der Arduino IDE geschrieben und auf den ESP8266 geladen (s. Anhang)

4.3 Aufbau des HTTP-Prototyps

Für den Aufbau des HTTP-Prototttyps werden für MQTT der ESP8266 D1 Mini und der Shelly Plug Plus S verwendet (Abbildung 4). Zusätzlich kommt ein Webserver zum Einsatz. Auf diesem befindet sich ein PHP-Skript mit dem Namen „Shelly.php“, welches den gewünschten Status in der Textdatei „status.txt“ abspeichert. Durch den Weg über einen Webserver wird das „Konzept des hohlen Baumstamms“ (vgl. Brell, o.D.-c.) umgesetzt, indem eine vermittelnde Instanz, in diesem Fall der externe Server, eingesetzt wird. Der Shelly Plus Plug S schickt zyklische Statusabfragen an den Webserver. Im Code sind mit der Zeile „/* number of miliseconds */ 5000“ 5000 Millisekunden (5 Sekunden) verankert. Der Shelly fragt also alle 5 Sekunden beim Webserver nach einer Statusänderung. Diese zyklische Abfrage kann verändert und an die Bedürfnisse in der Imkerei - z.B. alle fünf Minuten - angepasst werden. Bei jeder Rückmeldung passt er seinen Status dem Status des Webservers an.

4.3.1 Einrichten des ESP8266 D1 Mini

Die Programmierung des ESP8266 D1 Minis erfolgt über die Arduino IDE. In diesem Aufbau ist als zusätzliche Bibliothek nur die JSON-Bibliothek erforderlich. Es müssen bei dieser

Übertragung, anders als bei MQTT, keine Credentials übertragen oder Zertifikate installiert werden.

Vor dem Überspielen des Programms auf den Mikrocontroller müssen auch hier die WLAN-Daten, sowie die URL des PHP-Skripts im Code angepasst werden. Nach dem Hochladen auf den Mikrocontroller können die Ausgaben ebenfalls über den seriellen Monitor eingesehen werden.

4.3.2 Einrichten des Shelly Plus Plug S

Die Einrichtung ist gleich der bei MQTT.

5 Vor- und Nachteile von MQTT und HTTP

Aus der Literaturrecherche und der Entwicklung der beiden Prototypen lassen sich Vor- und Nachteile von MQTT und HTTP ableiten.

5.1 Vor- und Nachteile aus der Literaturrecherche

MQTT bietet durch das Publish/Subscribe-Prinzip eine One-to-Many-Kommunikation, die die **Skalierbarkeit** in verteilten Systemen erleichtert. Bei HTTP hingegen ist die Skalierbarkeit durch eine One-to-One-Kommunikation erschwert.

Darüber hinaus wird durch das Publish/Subscribe-Prinzip und den **Einsatz eines Brokers** die Latenz und die Auslastung des Netzwerks verbessert. Im Gegensatz dazu ist das gesamte System von der Erreichbarkeit und Zuverlässigkeit des Brokers abhängig. Wenn der Broker also ausfällt oder nicht mehr funktioniert, kann die gesamte Kommunikation im Netzwerk gestört oder unterbrochen werden.

Der Overhead von MQTT ist generell kleiner als der von HTTP. Nichtsdestotrotz ist HTTP bei einer einmaligen Nachrichtenübertragung oder bei regelmäßigen Übertragungen mit langen Phasen der Inaktivität zu bevorzugen, da in diesem Fall durch HTTP eine geringere **Netzauslastung** erzeugt wird als von MQTT. Auch der **Energieverbrauch** und die bereits angesprochene **Latenzzeit** sind niedriger bei MQTT. Im Gegensatz dazu verbraucht MQTT aufgrund seiner Struktur mehr **Speicherplatz**.

Im Vergleich zu HTTP ist MQTT besonders von Vorteil bei instabilen Netzen. Dies ist auf die asynchrone Kommunikation, die Verfügbarkeit von drei QoS-Ebenen und persistente Sessions zurückzuführen, bei denen der Kommunikationszustand gespeichert wird. Dies ermöglicht die **Aufrechterhaltung des Kommunikationszustands** über Verbindungsunterbrechungen hinweg. Im Gegensatz dazu erfolgt die Kommunikation von HTTP synchron. Auf jede Anfrage eines Clients erfolgt eine Antwort des Servers. Obwohl HTTP seit der Version 1.1 auch persistente Verbindungen anbietet, bei der mehrere Nachrichten über die gleiche TCP-Verbindung ausgetauscht werden können, bleibt es zustandslos. Die vorherige Kommunikation hat somit keinen Einfluss auf aktuelle Anfragen. Dies birgt das Risiko von Paketverlusten, wenn die TCP-Verbindung unterbrochen wird. Zur Speicherung des Zustands bei HTTP kann ein Add-on Mechanismus namens Cookies implementiert werden. Dies erfordert jedoch zusätzlichen Aufwand. Daraus lässt sich ableiten, dass das MQTT-Protokoll **Funktionen** anbietet, die bei HTTP nur durch einen Mehraufwand implementiert werden können.

Bei MQTT ist die **Nachrichtengröße** auf 256 MB begrenzt, während bei HTTP eine unbegrenzte Nachrichtengröße zugelassen ist.

Ein weiterer Vorteil von MQTT besteht darin, dass MQTT datenunabhängig ist, während HTTP textbasiert ist. Hieraus resultiert jedoch auch, dass HTTP-Nachrichten in einem von Menschen lesbaren **Format** vorliegen, was die Fehlerbehebung erleichtern kann.

Ein Vorteil von HTTP gegenüber MQTT liegt in der weit verbreiteten Nutzung des Netzwerkprotokolls und der dadurch vorhandenen **Expertise** sowie der **Verfügbarkeit von Ressourcen** wie Softwarebibliotheken.

5.2 Vor- und Nachteile aus der Analyse der Prototypen

MQTT erfordert bei der Einrichtung des hier gewählten Szenarios mehr Expertise als HTTP. So ist die Dokumentation der Shelly-Steckdose zum Einstellen des MQTT-Protokolls unübersichtlich. Blog-Beiträge, Erfahrungsberichte im Internet oder weitere Ressourcen, auf die man sich bei der Fehlerbehebung stützen könnte, waren nur spärlich vorhanden.

Der Code für den Publisher auf dem ESP8266 D1 Mini in der Arduino IDE hat für MQTT im Vergleich mehr Lines of Code (LOC) als der Web-Client für HTTP. Jedoch ist bei der MQTT-**Implementierung**, im Vergleich zu der HTTP-Implementierung, eine zusätzliche Abfrage vom Shelly vorgenommen worden. Weiterhin wurde für die MQTT-Implementierung eine SSL-Verschlüsselung der Datenübertragung mit programmiert. Dementsprechend ist die **Sicherheit** der Übertragung bei der MQTT-Implementierung höher einzustufen als bei der HTTP-Implementierung. Darüber hinaus mussten für die HTTP-Implementierung zusätzliche Skripte für den Server, als auch für die Shelly-Steckdose geschrieben werden, was die LOC-Anzahl relativiert.

Für die Nutzung des **HiveMQ Cloud Brokers** ergaben sich ebenfalls Vor- und Nachteile. Einerseits ist die Nutzung des Brokers kostenlos bei einer Anbindung von bis zu 100 Geräten und 10 GB Datenübertragungen pro Monat. Andererseits muss ein Nutzerprofil mit persönlichen Daten angelegt werden, um den Broker verwenden zu können. Darüber hinaus ist der HiveMQ Cloud Broker sparsam mit Informationen. Das Dashboard im Browser zeigt die aktuelle Nutzung:

- Wie viele Endpunkte sind aktuell verbunden.
- Wieviel Daten wurden insgesamt übermittelt.

Es kann nicht eingesehen werden - und erschwert die Fehlersuche:

- Welche Geräte mit dem Broker verbunden sind.
- Wie sehen die Nachrichten aus, die übermittelt werden.
- Welchen Umfang haben einzelne Nachrichten.

Ein weiterer Aspekt ist die standardmäßige Nutzung des 8883-Ports von HiveMQ. Einerseits wird so eine **sichere Übertragung** mittels SSL-Verschlüsselung garantiert. Andererseits können Nachrichten, die über den 8883-Port übertragen werden, nicht einfach mithilfe von Netzwerk-Sniffer wie Wireshark ausgelesen werden, was sowohl die **Fehlerbehebung** als auch die **Messung**

von Parametern wie der Headergröße erschwert. Weiterhin ergaben sich bei der Generierung der SSL-Zertifikate in Microsoft Fehler, deren Behebung durch die limitierte **Dokumentation** ebenso erschwert war.

Die **Kosten** für die HTTP-Implementierung, als auch für die MQTT- Implementierung sind vergleichbar. Für 35-40 € können die Prototypen nachgebaut werden. Bei der HTTP-Implementierung kommen noch die Kosten für einen externen Server wie in diesem Falle Domainfactoy hinzu, wenn die Portweiterleitung im eigenen **Netzwerk** aus Sicherheitsgründen nicht genutzt werden soll. MQTT entfaltet seine Vorteile insbesondere in verteilten Netzwerken mit einer hohen Anzahl an Geräten und Datenübertragungen in einem instabilen Netz. Diese Kriterien waren in dem vorliegenden Anwendungsfall nicht gegeben.

6 Fazit und Ausblick

Zur Unterstützung der Bienenhaltung können moderne Kommunikationstechnologien einen relevanten Beitrag leisten. Das Ziel dieser Untersuchung war es, den Stand der HTTP- und der MQTT-Technologie gegenüberzustellen. Unterschiede und Gemeinsamkeiten der Technologien mit waren mit Hilfe von Literatur und der Entwicklung von zwei Prototypen aufzuzeigen.

In Tabelle 1 sind die Eigenschaften von HTTP und MQTT gegenübergestellt.

6.1 Forschungsfrage F1 (Unterschiede)

Zur Beantwortung der Forschungsfrage F1 „Was ist MQTT und inwiefern unterscheiden sich die Eigenschaften von MQTT und HTTP?“ kann zusammengefasst werden, dass es sich bei MQTT, wie auch bei HTTP, um ein Protokoll der Anwendungsschicht (OSI-Modell Ebene 7) handelt. Beide Protokolle basieren auf TCP/IP in der Transportschicht (OSI-Modell 4 inkl. Schicht 3).

6.1.1 Client-Server-Technologie

MQTT ist nach einer ereignisgesteuerten Architektur mit einem zwischengeschalteten Broker aufgebaut, die nach dem Publish/Subscribe-Prinzip funktioniert. Die Datenübertragung verläuft asynchron. Mit der One-to-Many Kommunikation können über den Broker Nachrichten an alle Subscriber versandt werden, die ein Topic abonniert haben.

HTTP hingegen basiert auf dem Request/Response-Prinzip und hat eine Client-Server-Architektur. Infolgedessen erfolgt die One-to-One Kommunikation bei HTTP synchron. MQTT bietet zusätzliche Qualitätsstufen an, sodass auch unzuverlässige Verbindungen bedient werden können.

6.1.2 Persistenz

Neben persistenten Sessions wird abhängig von der Qualitätsstufe bei **MQTT** auch der Kommunikationszustand des Subscribers durch den Broker gespeichert, wodurch eine Verbindung auch nach einer Unterbrechung wieder aufgenommen werden kann.

HTTP bietet ebenfalls persistente Verbindungen an, da es sich aber um ein zustandsloses Protokoll handelt, ist nach einem Verbindungsabbruch die vorhergegangene Kommunikation nicht mehr nachzuvollziehen.

6.1.3 Datenarten

MQTT kann verschiedene Datentypen übertragen. **HTTP** hingegen versendet textbasierte Nachrichten, was eine höhere Bandbreite erfordert.

6.2 Forschungsfrage F2 (Anwendungsdomänen)

Die Forschungsfrage F2 „Welche Anwendungsdomänen sind geeignet für MQTT und welche für HTTP?“ kann wie folgt beantwortet werden: HTTP wurde ursprünglich für den Anwendungsbereich des WWW - Abruf von verlinkten Informationen von Menschen - entwickelt. MQTT hingegen wurde mit dem Ziel entwickelt, ein effizientes Protokoll für IoT-Anwendungen bereitzustellen. Beide Protokolle werden heute für die Datenübermittlung im IoT genutzt. HTTP eignet sich dabei vor allem für Anwendungsdomänen mit einer einmaligen Nachrichtenübertragung oder bei regelmäßigen Übertragungen mit langen Phasen der Inaktivität. Mit MQTT hingegen lässt sich bei periodischer, häufiger Datenübertragung Datenvolumen einsparen. Weiterhin eignet sich MQTT aufgrund des asynchronen Aufbaus und der Qualitätsstufen insbesondere für Anwendungsbereiche, bei denen nur limitierte und ggf. instabile Netzwerke zur Verfügung stehen.

6.3 Forschungsfrage F3 (Vor- und Nachteile)

Zur Beantwortung der Forschungsfrage F3 „Welche Vor- und Nachteile ergeben sich aus der praktischen Umsetzung?“ werden folgende Faktoren berücksichtigt: die **Kosten** sind für beide Prototypen ähnlich hoch. Ein Vorteil den HTTP bietet ist die weite Verbreitung und die damit einhergehende breitgestreute Expertise bei der Implementierung von HTTP-Lösungen. Für MQTT steht wenig Dokumentation in Form von Blogartikeln oder ähnlichem zur Verfügung. Die Anzahl der Codezeilen ist bei der hier untersuchten MQTT-Lösung höher. Dafür werden für den MQTT-Prototyp keine separaten selbsterstellten Serverprozesse benötigt.

Der kostenlose Cloud Broker von HiveMQ ist einfach zu implementieren, jedoch muss ein eigenes Benutzerkonto angelegt und somit auch Daten des Nutzers bei HiveMQ hinterlegt werden. Der kostenlose Cloud Broker ist eine "Black Box" ohne Möglichkeit, Nachrichtenpakete oder angebundene Endpunkte zu überprüfen.

6.4 Forschungsfrage F4 (Praxis)

Die Forschungsfrage F4 „Welche der beiden Möglichkeiten der Implementierung ist besser geeignet für eine praktische Anwendung in der Imkerei?“ lässt sich folgendermaßen beantworten: Durch die hohe Verbreitung von HTTP und den bisherigen Einsatz im Biene40-Projekt ist HTTP auch den technisch nicht affinen Imker:innen bekannt. Die im Projekt arbeitenden Imker:innen hatten bereits Kontakt mit HTTP und konnten einen Prototyp in der Beute betreiben. In einigen der untersuchten Literaturquellen wird MQTT bereits im Imkereikontext verwendet. Imker:innen mit einer Neigung zur Technik können das System selbst zusammenbauen, für weniger affine Imker:innen könnte ein kommerzielles vorgefertigtes Produkt sinnvoll sein. Generell sind die Vor- und Nachteile, je nach Anwendungszweck abzuwägen. MQTT ist leichtgewichtig und bietet vor allem bei instabilen Netzen, wie sie oft bei Bienenständen vorkommen, einen Vorteil. Außerdem ist es leicht skalierbar. Durch die reduzierte Dokumentation kann die Einrichtung eines Brokers schwierig sein und Sicherheitsprobleme

nach sich ziehen, sofern der Imker oder die Imkerin keine ausreichenden Kenntnisse in diesem Bereich aufweist. Dies stellt eine Hürde für Imker:innen dar, die technisch nicht versiert sind.

6.5 Ausblick

Betrachtet wurden die Übertragungsprotokolle HTTP und MQTT, hier über eine WLAN-Verbindung, die nicht zwangsläufig am Bienenstand vorhanden ist. Zukünftige Forschungen können sich mit weiteren Vernetzungslösungen, wie Zigbee, Sigfox, LoRaWAN, NB-IoT oder LTE-M beschäftigen. In dieser Untersuchung wurden nur kurze Datensätze eines Bewegungsmelders versandt. Zukünftige Arbeiten können daran anknüpfen und weitere wichtige Sensordaten für die Überwachung von Bienenstöcken berücksichtigen. Beispielsweise sind hier Temperatur, Soundkennwerte wie prägnanteste Frequenz oder Spitzenamplitude oder Gewicht des Bienenstocks zu nennen.

Das Projekt Biene40 setzt derzeit auf HTTP(S) und nicht auf MQTT. Die hier erarbeiteten Ergebnisse geben heute keinen Anlass, davon abzuweichen, wenn die inhaltlichen Rahmenbedingungen von Biene40 unverändert bleiben:

- Seltene Datenübertragung
- Wenige Daten

7 QUELLEN

Abts, D. (2015): Masterkurs Client/Server-Programmierung mit Java: *Anwendungen entwickeln mit Standard-Technologien, 4. Aufl., Wiesbaden:* Springer Fachmedien Wiesbaden, 2015.

BMEL (o.D.): Beenovation, https://www.beenovation.de, Zugriff am 04.07.2023.

Brell, C. (o.D.-a): Biene40 – Entwicklung digitaler vernetzter Sensoren für vitalere Bienen, http://bieneviernull.de, Zugriff am 04.07.2023.

Brell, C. (o.D.-b): Was ist IoT (Internet of Things)?, https://cbrell.de/blog/was-ist-iot-internet-of-things/, Zugriff am 23.09.2023.

Brell, C. (o.D.-c): Konzept des hohlen Baumstamms – was IoT und Drogendealer gemein haben. https://cbrell.de/blog/konzept-des-hohlen-baumstamms-was-iot-und-drogendealer-gemein-haben/, Zugriff am 05.10.2023.

Brell, C. (2022): Technischer Bericht – Sensoren in Bienenstöcken (Zustand), http://bieneviernull.de/technischer-bericht-sensoren-in-bienenstoecken-zustand/, Zugriff am 04.07.2023.

Conrad Electronic SE (o.D.-a): Joy-it ESP8266-12F Entwickler-Platine 1 St., https://www.conrad.de/de/p/joy-it-esp8266-12f-entwickler-platine-1-st-1707668.html, Zugriff am 29.06.2023.

Conrad Electronic SE (o.D.-b): Joy-it SEN-HC-SR501 Bewegungssensor 1 St., https://www.conrad.de/de/p/joy-it-sen-hc-sr501-bewegungssensor-1-st-2361790.html, Zugriff am 29.06.2023.

Conrad Electronic SE (o.D.-c): Shelly Plus Plug S Steckdose Wi-Fi, Bluetooth, https://www.conrad.de/de/p/shelly-plus-plug-s-steckdose-wi-fi-bluetooth-2757312.html, Zugriff am 29.06.2023.

Craggs, I. (2022): MQTT Vs. HTTP for IoT, https://www.hivemq.com/blog/mqtt-vs-http-protocols-in-iot-iiot/, Zugriff am 25.04.2023.

Deutscher Imkerbund e.V. (o.D.-a): Bienen als Bestäuber, https://deutscherimkerbund.de/163-Bienen_Bestaeubung_Zahlen_die_zaehlen, Zugriff am 04.07.2023.

Deutscher Imkerbund e.V. (o.D.-b): Bienen als Bestäuber helfen beim Erhalt der Artenvielfalt, https://deutscherimkerbund.de/235-Echter_Deutscher_Honig_Honig_und_Natur_Schutz, Zugriff am 04.07.2023.

EHI Retail Institute GmbH (2022): Top 100 Onlineshops in Deutschland, https://www.ehi.org/news/top-100-onlineshops-in-deutschland/, Zugriff am 30.06.2023.

Fritz, K.P./Strauß, H./Rathfelder, C./Bülau, A./Gaida, D./Girdvainis, D./Marki, G. (2021): Digitaler Retrofit: von Maschinen und Produktionsanlagen, 1. Aufl., Würzburg: Vogel Communications Group, 2021.

Herrero, R. (2022): Fundamentals of IoT Communication Technologies, 1. Aufl., Cham: Springer Nature Switzerland, 2022.

Hevner, A./ March, S/ Park, J./Ram, S. (2004): Design Science in Information SystemsResearch, in: MIS Quarterly 28 -1, S.75-100.

HiveMQ (o.D.-a): Key Features, https://www.hivemq.com/mqtt/mqtt-protocol/, Zugriff am 26.04.2023.

HiveMQ (o.D.-b): Features, https://www.hivemq.com/hivemq/features/, Zugriff am 30.06.2023.

HiveMQ (o.D.-c): HiveMQ Editions, https://www.hivemq.com/hivemq/editions/, Zugriff am 30.06.2023.

HiveMQ (o.D.-d): HiveMQ Cloud now grows with your needs, https://www.hivemq.com/article/hivemq-cloud-offers-pay-as-you-go-plan/, Zugriff am 30.06.2023.

HiveMQ (o.D.-e): Getting started with Arduino ESP8266, https://console.hivemq.cloud/clients/arduino-esp8266?uuid=42ef8d8b455248368b9538e9282ad896, Zugriff am 15.06.2023.

HiveMQ (2023-a): Introducing the MQTT Protocol – MQTT Essentials: Part 1, https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt/, Zugriff am 23.09.2023.

HiveMQ (2023-b): Understanding Persistent Sessions and Clean Sessions – MQTT Essentials: Part 7, Zugriff am 23.09.2023.

Kakakhel, S. R. U./Westerlund, T./Daneshtalab, M./Zou, Z./Plosila, J./Tenhunen, H. (2019): A Qualitative Comparison Model for Application Layer IoT Protocols, 2019 Fourth International Conference on Fog and Mobile Edge Computing (FMEC), Rome, Italy, 2019, S.210-215, Verfügbar unter: 10.1109/FMEC.2019.8795324.

Machhamer, R. / Altenhofer, J. / Ueding, K. / Czenkusch, L. / Stolz, F. / Harth, M. / Mattern, M. / Latif, A. / Haab, S. / Herrmann, J. / Schmeink, A. / Gollmer, K.-U. / Dartmann, G. (2020): Visual Programmed IoT Beehive Monitoring for Decision Aid by Machine Learning based Anomaly Detection, https://ieeexplore.ieee.org/document/9134323, Zugriff am 06.07.2023.

Mahamud, S. / Rakib, A. / Faruqi, T. / Haque, M. / Rukaia, S. / Nazmi, S. (2019): Mouchak - An IoT Basted Smart Beekeeping System Using MQTT, https://ieeexplore.ieee.org/document/9043815, Zugriff am 06.07.2023.

makesmart (2020): ESP8266 D1 Mini programmieren, https://makesmart.net/esp8266-d1-mini-programmieren/, Zugriff am 11.05.2023.

makesmart (2021): Arduino IDE – Arbeiten mit JSON Objekten für Einsteiger, https://makesmart.net/arduino-ide-arbeiten-mit-json-objekten-fur-einsteiger/, Zugriff am 20.06.2023.

Nicholas, S.D. (2012): Power Profiling: HTTPS Long Polling vs. MQTT with SSL, on Android, http://stephendnicholas.com/posts/power-profiling-mqtt-vs-https, Zugriff am 30.09.2023.

OASIS (2014): MQTT Version 3.1.1, http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718018, Zugriff am 06.07.2023.

O'Leary, N. (o.D.): API Documentation: Library version: 2.8, https://pubsubclient.knolleary.net/api, Zugriff am 20.06.2023.

Peffers, K./ Tuunanen, T./ Rothenberge, M./ Chatterjee, S. (2008): A design science research methodology for information systems research, Journal of MIS, 24(3), S.45–77.

Reichelt Elektronik GmbH & Co KG (o.D.-a): DEBO ESP8266-12F Entwicklerboards - ESP8266 WiFi-Auflötmodul, https://www.reichelt.de/entwicklerboards-esp8266-wifi-aufloetmodul-debo-esp8266-12f-p236022.html?&trstct=pos_0&nbc=1, Zugriff am 29.06.2023.

Reichelt Elektronik GmbH & Co KG (o.D.-b): RPI HC-SR501 Raspberry Pi -Infrarot Bewegungsmelder, PIR, HC-SR501, https://www.reichelt.de/raspberry-pi-infrarot-bewegungsmelder-pir-hc-sr501-rpi-hc-sr501-p224216.html, Zugriff am 29.06.2023.

Reichelt Elektronik GmbH & Co KG (o.D.-c): SHELLY PLUSPLUGS Shelly Plus Plug S, https://www.reichelt.de/shelly-plus-plug-s-shelly-plusplugs-p338872.html?&trstct=pos_0&nbc=1, Zugriff am 29.06.2023.

Re-In Retail International GmbH (o.D.-a): Joy-it ESP8266-12F Entwickler-Platine 1St, https://www.voelkner.de/products/1108233/Joy-it-ESP8266- 12F-Entwickler-Platine-1St..html?offer=706816755a7e67164ca200c02c2a3cd1, Zugriff am 29.06.2023.

Re-In Retail International GmbH (o.D.-b): Joy-it SEN-HC-SR501 Bewegungssensor 1St., https://www.voelkner.de/products/4180975/Joy-it- SEN-HC-SR501-Bewegungssensor-1St..html?offer=8f66d9c9bfe184c0f2baf569abdd36d7, Zugriff am 29.06.2023.

Re-In Retail International GmbH (o.D.-c): Shelly Plus Plug S Steckdose Wi-Fi, Bluetooth, https://www.voelkner.de/products/6730053/Shelly-Plus-Plug-S-Steckdose-Wi-Fi-Bluetooth.html?offer=c150762a54034e9b7d482a6cddcafed1, Zugriff am 29.06.2023.

RFC (1999): Hypertext Transfer Protocol – HTTP/1.1, https://www.rfc-editor.org/rfc/rfc2616, Zugriff am 23.09.2023.

Shelly (o.D.-a): Shelly Plus Plug S User and Safety Guide, https://kb.shelly.cloud/knowledge-base/shelly-plus-plug-s-user-and-safety-guide, Zugriff am 15.06.2023.

Shelly (o.D.-b): MQTT, https://shelly-api-docs.shelly.cloud/gen2/0.14/ComponentsAndServices/Mqtt/, Zugriff am 15.06.2023.

Shelly (o.D.-c): Unsere App Shelly Smart Control, https://www.shelly.cloud/de/shelly-smart-control, Zugriff am 21.07.2023.

Šikić, L./ Janković, J./ Afrić, P./Šilić, M./Ilić, Ž./Pandžić, H./Živić, M./Džanko, M. (2020): A comparison of Application Layer Communication Protocols in IoT-enabled Smart Grid, 2020 International Symposium ELMAR, Zadar, Croatia, 2020, S.83-86, Verfügbar unter: 10.1109/ELMAR49956.2020.9219030.

SIMAC Electronics GmbH (o.D.): Passiver Infrarot-Bewegungssensor: SEN-HC-SR501, https://asset.conrad.com/media10/add/160267/c1/-/gl/002361790ML01/bedienungsanleitung-2361790-joy-it-sen-hc-sr501-bewegungssensor-1-st.pdf, Zugriff am: 21.05.2023.

Trojan, W. (2017): Das MQTT-Praxisbuch, 1. Aufl., Aachen: Elektor Verlag GmbH, 2017.

Wurm, J./Brell, C. (2022): Imkerei und Digitalisierung – Stand und Perspektiven aus technischer Sicht. Arbeitsbericht Nr. 3 / Biene40, http://bieneviernull.de/wp-content/uploads/2022/08/Biene40_Arbeitsbericht_Nr_03_Literaturanalyse_Informatik_final.pdf, Zugriff am 06.07.2023.

Yusof, Z. M. / Billah, M. / Kadir, K. / Mohd Ali, A. M. / Ahmad, I. (2019): Improvement of Honey Production: A Smart Honey Bee Health Monitoring System, https://ieeexplore.ieee.org/document/9057336, Zugriff am 06.07.2023.

Zabasta, A. / Kunicina, N. / Kondratjevs, K. / Ribickis, L. (2019): IoT Approach Application for Development of Autonomous Beekeeping System, https://ieeexplore.ieee.org/document/8883460, Zugriff am 06.07.2023.

8 ANHANG / QUELLTEXTE

8.1 HTTP Implementierung

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <ArduinoJson.h>

HTTPClient sender;
WiFiClient wifiClient;

// Initialisieren der WLAN-SSID- und der WLAN-Passworts-Variable. Entsprechende Parameter muessen erga-
enzt werden.
const char* ssid = “WLAN-SSID”;
const char* password = "WLAN-Passwort";

// Festlegen des Pins für den Sensor und Initalisieren der movement-Variable.
int pin = D6;
int movement = 0;

// Deklarieren einer Boolean-Variable namens active, um die Nachrichtenuebertragung zu minimieren.
boolean active;

// Methode zum Senden von HTTP-Requests (Quelle: Vgl. https://makesmart.net/esp8266-http-get-request/)
void send_request(String url) {
  if (sender.begin(wifiClient, url)) {

    // HTTP-Code der Response speichern
    int httpCode = sender.GET();

    // Anfrage wurde gesendet und Server hat geantwortet; Info: Der HTTP-Code für 'OK' ist 200
    if (httpCode > 0 && httpCode == HTTP_CODE_OK) {

      // Speichern der Antwort des Shelly.
      String response = sender.getString();

      // Ausgabe der Antwort für den Nutzer.
      Serial.println(response);
    }

    else {
      // Ausgabe bei einem HTTP-Error.
      Serial.printf("HTTP-Error: ", sender.errorToString(httpCode).c_str());
    }

    // Wenn alles abgeschlossen ist, wird die Verbindung wieder beendet
    sender.end();
```

```
  }

  else {
    Serial.printf("HTTP-Verbindung konnte nicht hergestellt werden!");
  }
}

// Methode zum Einwaehlen in das WLAN (Quelle: Vgl. https://console.hivemq.cloud/clients/arduino-
esp8266?uuid=42ef8d8b455248368b9538e9282ad896).
void setup_wifi() {
  delay(10);

  // Statusangabe fuer den Nutzer ausgeben.
  Serial.println();
  Serial.print("Verbinden zu ");
  Serial.println(ssid);

  // Einwaehlen in das WLAN.
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  // Solange noch keine Verbindung zum WLAN besteht wird ein Punkt ausgegeben.
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  // Statusangabe fuer den Nutzer ausgeben.
  Serial.println("");
  Serial.println("WLAN verbunden");
  Serial.println("IP-Adresse: ");
  Serial.println(WiFi.localIP());
}


void setup() {
  delay(500);
  // Baudrate für den seriellen Monitor festlegen.
  Serial.begin(9600);
  delay(500);

  // Ins WLAN einwaehlen.
  setup_wifi();
}

void loop() {
// Auslesen des Sensors (Vgl. https://asset.conrad.com/media10/add/160267/c1/-
/gl/002361790ML01/bedienungsanleitung-2361790-joy-it-sen-hc-sr501-bewegungssensor-1-st.pdf).
  movement = digitalRead(pin);

  // Die Bedingung ist nur erfuellt, wenn eine Bewegung erkannt wurde.
  if(movement == HIGH){
    // Die zu uebermitelnde Nachricht schaltet den Status auf dem Webserver ein.
    const String url = "http://XXXXXX.de/iot/mfp/Shelly.php?x=on";

    // Die HTTP-Anfrage wird an den Webserver gesendet.
    send_request(url);

    // Die Variable active wird auf true gesetzt, um eine erneute Nachrichtenuebermittlung bei ange-
schaltetem Shelly zu verhindern.
    active = true;

    // Die Verzoegerung dient dazu, den Shelly eine gewisse Zeit angeschaltet zu lassen bei einer Bewe-
gung. Die Dauer kann angepasst werden.
    delay(5000);
  }
```

```
  // Die Bedingung ist nur erfuellt, wenn keine Bewegung erkannt wurde und die Boolean Variable active
auf true steht (also wenn der Shelly an ist).
  if (movement == LOW && active == true) {

    // Die zu uebermitelnde Nachricht schaltet den Status auf dem Webserver aus.
    const String url = "http://210301.de/iot/mfp/Shelly.php?x=off";

    // Die HTTP-Anfrage wird an den Webserver gesendet.
    send_request(url);

    // Die Variable active wird auf false gesetzt, um eine erneute Nachrichtenuebermittlung bei ausge-
schaltetem Shelly zu verhindern.
    active = false;
   }
}
```

8.2 Serverscript zur HTTP Implementierung

```
<?php
//Wenn x per URL übergeben wird, weise Variable x den Wert zu
if (isset($_GET['x'])) {
    $x = $_GET['x'];
       $filename = "status.txt";
    if ($x === 'on') {
          //Gib „on“ aus und schreibe „on“ in die Textdatei
        echo "Der Modus wurde auf 'on' gesetzt.";
              file_put_contents($filename, "on");
    } elseif ($x === 'off') {
              //Gib „off“ aus und schreibe „off“ in die Textdatei
        echo "Der Modus wurde auf 'off' gesetzt.";
              file_put_contents($filename, "off");
    } elseif ($x === 'sta') {
          //Lies Status aus der Textdatei
        $content = file_get_contents($filename);
              echo $content;
    }else {
        echo "Ungültiger Modus angegeben.";
    }
} else {
    echo "Kein Modus angegeben.";
}
?>
```

8.3 Script auf der Shelly-Steckdose

```
function timerCode() {
/*Rufe die URL auf und untersuche die Antwort, schaltet ggf. den Shelly in den anderen Status*/
 Shelly.call(
  "HTTP.GET", {
    "url": "https://210301.de/iot/mfp/Shelly.php?x=sta",
  },
  function(result) {
    print("Hello ", result.body);
    if(result.body === "off"){
      Shelly.call("Switch.set", {'id': 0, 'on': false});
    }
    if(result.body === "on"){
      Shelly.call("Switch.set", {'id': 0, 'on': true});
    }
  }
 )
};

Timer.set(
  /* Setze Timer auf 5 Sekunden, wiederhole den Timer und führe die Funktion timerCode aus*/
  /* number of miliseconds */ 5000,
  /* repeat? */ true,
```

```
  /* callback */ timerCode
);
```

8.4 MQTT Implementierung

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <time.h>
#include <TZ.h>
#include <FS.h>
#include <LittleFS.h>
#include <CertStoreBearSSL.h>
#include <ArduinoJson.h>


// Initialisieren der WLAN-SSID- und der WLAN-Passworts-Variable. Entsprechende Parameter muessen erga-
enzt werden.
const char* ssid = "WLAN-SSID einsetzen";
const char* password = "WLAN-Passwort einsetzen";

// Initialisieren des Credential-Usernamen und Credentials-Passworts.
const char* credentials_user = "Credentials-Username einsetzen";
const char* credentials_password = "Credentials-Passwort einsetzen";

// Initialisieren der Cluster-URL-Variable. Ergaenzen der entsprechenden HiveMQ-Cloud-Cluster-URL.
const char* mqtt_server = "Cluster-URL einsetzen";

// Initialisieren der MQTT-Topic-Variable fuer die Abfrage des Zustands. Ergaenzen des Topics (MQTT-
Praefix/status/switch:0).
const char* mqttTopicStatus = "MQTT-Praefix/status/switch:0";

// Initialisieren der MQTT-Topic-Variable zum Veraendern des Zustands. Ergaenzen des Topics (MQTT-
Praefix/command/switch:0).
const char* mqttTopicControl = "MQTT-Praefix/command/switch:0";

// Deklarieren eines CertStore-Objekts aus der BearSSL-Bibliothek.
BearSSL::CertStore certStore;

// Deklarieren eines PubSubClient-Zeigers.
PubSubClient *client;

// Festlegen des Pins fuer den Sensor und Initialisieren der movement-Variable.
int pin = D6;
int movement = 0;

// Deklarieren einer Boolean-Variable namens active, um die Nachrichtenuebertragung zu minimieren.
boolean active;

// Methode zum Einwaehlen in das WLAN (Quelle: Vgl. https://console.hivemq.cloud/clients/arduino-
esp8266?uuid=42ef8d8b455248368b9538e9282ad896).
void setup_wifi() {
  delay(10);

  // Statusangabe fuer den Nutzer ausgeben.
  Serial.println();
  Serial.print("Verbinden zu ");
  Serial.println(ssid);

  // Einwaehlen in das WLAN.
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  // Solange noch keine Verbindung zum WLAN besteht wird ein Punkt ausgegeben.
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
```

```
  // Statusangabe fuer den Nutzer ausgeben.
  Serial.println("");
  Serial.println("WLAN verbunden");
  Serial.println("IP-Adresse: ");
  Serial.println(WiFi.localIP());
}

// Methode zum Setzen der Zeit. Nur das Datum wird fuer die Validierung der Sicherheits-Zertifikate
benoetigt (Quelle: Vgl. https://console.hivemq.cloud/clients/arduino-
esp8266?uuid=42ef8d8b455248368b9538e9282ad896).
void setDateTime() {
  configTime(TZ_Europe_Berlin, "pool.ntp.org", "time.nist.gov");

  Serial.print("Warten auf NTP Zeitsynchronisation: ");

  time_t now = time(nullptr);

  while (now < 8 * 3600 * 2) {
    delay(100);
    Serial.print(".");
    now = time(nullptr);
  }
  Serial.println();

  struct tm timeinfo;
  gmtime_r(&now, &timeinfo);
  Serial.printf("%s %s", tzname[0], asctime(&timeinfo));
}

// Ueberschreiben der callback-Methode.
void callback(char* topic, byte* payload, unsigned int length) {

  // Initalisieren einer String-Variable ohne Inhalt.
  String message = "";

  // Der Char-Zeiger wird in einen String umgewandelt.
  String topicString = topic;

  // Die Nachricht wird schrittweise der message-Variable hinzugefuegt.
  for (int i = 0; i < length; i++) {
    message = message + (char)payload[i];
  }



  // Wenn die Nachricht aus dem Status-Topic stammt, soll der Status ueberprueft werden (Quelle: Vgl.
https://makesmart.net/arduino-ide-arbeiten-mit-json-objekten-fur-einsteiger/).
  if (topicString == mqttTopicStatus) {

    // Anlegen eines Buffers.
    DynamicJsonDocument config(1024);
    // Deserialisieren des JSON-Objekts.
    DeserializationError error = deserializeJson(config, message);
    if (error) {
      return;
    }

    // Speichern des Status (on oder off) in einer String-Variablen.
    boolean status = config["output"];

    // Ausgeben des aktuellen Status fuer den Nutzer.
    Serial.print("Status erhalten [");
    Serial.print(topicString);
    Serial.print("] ");
    Serial.print(status);
    Serial.println();

    // Die active-Variable entspricht dem uebermittelten Status.
```

```
    //if (status == "true") {
    active = status;

    // Abmelden vom Status-Topic. Status wird nur einmal zu Beginn benoetigt (Vgl. https://pubsubcli-
ent.knolleary.net/api).
    client->unsubscribe(mqttTopicStatus);
  }

  // Der ESP empfaengt auch die eigenen Nachrichten und gibt diese aus. So soll sichergestellt werden,
dass die versendeten Nachrichten auch ankommen.
  if (topicString == mqttTopicControl) {
    Serial.print("Nachricht erhalten [");
    Serial.print(topicString);
    Serial.print("] ");
    Serial.print (message);
    Serial.println();
  }

}

// Methode fuer die Verbindung zum MQTT Broker.
void reconnect() {
  // Solange der Client noch nicht verbunden ist, soll die Verbindung aufgebaut werden.
  while (!client->connected()) {
    // Statusausgabe fuer den Nutzer zum Verbindungsaufbau.
    Serial.print("Verbindungsaufbau MQTT-Verbindung…");

    // Initialisieren einer Variable fuer die Client-ID.
    String clientId = "ESP8266Client - MyClient";

    // Verbindungsaufbau, Credentials-Username und Credentials-Password ergaenzen.
    if (client->connect(clientId.c_str(), credentials_user, credentials_password)) {
      // Ausgabe einer Verbindungsbestaetigung.
      Serial.println("verbunden");

      // Anmelden fuer das Topic, um ein Status Update vom Geraet zu erhalten.
      client->subscribe(mqttTopicStatus);
      // Status des Shelly Plus Plug S erfragen.
      client->publish(mqttTopicStatus, "status_update");

      // Fuer das Topic wird sich auch als Subscriber angemeldet, um zu pruefen, ob die Nachrichten an-
kommen.
      client->subscribe(mqttTopicControl);
    }
    else {
      // Fehlermeldung bei fehlgeschlagener Verbindung ausgeben.
      Serial.print("failed, rc = ");
      Serial.print(client->state());
      Serial.println(" try again in 5 seconds");
      // 5 Sekunden warten bis zum naechsten Verbindungsversuch.
      delay(5000);
    }
  }
}

// Wird nur einmalig ausgefuehrt (Vgl. https://console.hivemq.cloud/clients/arduino-
esp8266?uuid=42ef8d8b455248368b9538e9282ad896).
void setup() {
  delay(500);
  // Baudrate fuer den seriellen Monitor festlegen.
  Serial.begin(9600);
  delay(500);

  // Initialisieren des LittleFS-Dateisystems.
  LittleFS.begin();
  // Ins WLAN einwaehlen.
  setup_wifi();
  // Setzen der Zeit fuer die Zertifikate.
```

```
  setDateTime();

  // Auslesen der vorhandenen Zertifikate aus dem LittleFS-Dateisystem.
  int numCerts = certStore.initCertStore(LittleFS, PSTR("/certs.idx"), PSTR("/certs.ar"));

  // Ausgabe fuer den Nutzer mit Anzahl gelesener Zertifikate.
  Serial.printf("Anzahl gelesener CA-Zertifikate: %d\n", numCerts);
  // Ausgabe, falls keine Zertifikate gelesen wurden.
  if (numCerts == 0) {
    Serial.printf("Keine Zertifikate gefunden. Wurde certs-from-mozilla.py ausgefuehrt und das LittleFS
Dateisystem vor dem hochladen geladen?\n");
    return;
  }

  // Aufbau einer sicheren Verbindung zum Broker.
  BearSSL::WiFiClientSecure *bear = new BearSSL::WiFiClientSecure();
  // Zugriff auf die Zertifikate.
  bear->setCertStore(&certStore);

  // Festlegen des Client als Objekt der PubSubClient-Klasse. Uebergabe des Zeigers bear als Verweis
auf das sichere WiFiClient-Objekt.
  client = new PubSubClient(*bear);

  // Erstellen der Verbindung zum Broker. Uebergabe der Cluster-URL und der Portnummer der MQTT Cloud.
  client->setServer(mqtt_server, 8883);

  // Die Callback-Funktion wird dem Client zugeordnet.
  client->setCallback(callback);
}

// Wird in einer Schleife ausgefuehrt.
void loop() {
  // Es soll eine Verbindung zum Broker aufgebaut werden, wenn der Client noch nicht verbunden ist.
  if (!client->connected()) {
    reconnect();
  }
  // Erlaubt dem Client eingehende Nachrichten zu verarbeiten und haelt die Verbindung zum Broker auf-
recht.
  client->loop();

  // Auslesen des Sensors (Vgl. https://asset.conrad.com/media10/add/160267/c1/-
/gl/002361790ML01/bedienungsanleitung-2361790-joy-it-sen-hc-sr501-bewegungssensor-1-st.pdf).
  movement = digitalRead(pin);

  // Die Bedingung ist nur erfuellt, wenn eine Bewegung erkannt wurde.
  if(movement == HIGH){
    // Die zu uebermittelnde Nachricht schaltet den Shelly ein.
    const char* payload = "on";

    // Die veroeffentlichte Nachricht wird dem Nutzer angezeigt.
    Serial.print("Veroeffentlichte Nachricht: ");
    Serial.println(payload);

    // Die Nachricht wird auf dem entsprechenden Topic veroeffentlicht.
   client->publish(mqttTopicControl, payload);

    // Die Variable active wird auf true gesetzt, um eine erneute Nachrichtenuebermittlung bei ange-
schaltetem Shelly zu verhindern.
    active = true;

    // Die Verzoegerung dient dazu, den Shelly eine gewisse Zeit angeschaltet zu lassen bei einer Bewe-
gung. Die Dauer kann angepasst werden.
    delay(5000);
  }

  // Die Bedingung ist nur erfuellt, wenn keine Bewegung erkannt wurde und die Boolean Variable active
auf true steht (also wenn der Shelly an ist).
  if (movement == LOW && active == true) {
```

```
    // Die zu uebermitelnde Nachricht schaltet den Shelly aus.
    const char* payload = "off";

    // Die veroeffentlichte Nachricht wird dem Nutzer angezeigt.
    Serial.print("Veroeffentlichte Nachricht: ");
    Serial.println(payload);

    // Die Nachricht wird auf dem entsprechenden Topic veroeffentlicht.
    client->publish(mqttTopicControl, payload);

    // Die Variable active wird auf false gesetzt, um eine erneute Nachrichtenuebermittlung bei ausge-
schaltetem Shelly zu verhindern.
    active = false;
   }

}
```

8.5 Subscriber für MQTT

Im ersten Abschnitt des Codes werden mehrere Variablen initialisiert, die bei einer Übernahme des Codes für eigene Arbeiten anzupassen sind. Darüber hinaus werden im ersten Abschnitt Methoden definiert. Neben einer Methode zum Einwählen in das WLAN gehört hierzu auch eine Methode, zum Setzen des Datums, welches für die Validierung der Sicherheits-Zertifikate benötigt wird (vgl. HiveMQ, o.D.-e.). Weiterhin wird die Callback-Methode überschrieben. Diese Methode der PubSubClient-Bibliothek legt fest, wie eingehende Nachrichten verarbeitet werden sollen (vgl. O'Leary, o.D.). Die reconnect-Methode erstellt die Verbindung des ESP8266 mit der HiveMQ-Cloud. Hier werden auch die Topics für die Statusabfrage und Kontrolle der Shelly Steckdose abonniert (vgl. HiveMQ, o.D.-e.).

In der setup-Methode werden die Methoden für die Initialisierung des LittleFS-Dateisystems, für die Einwahl ins WLAN und zum Setzen des Datums aufgerufen. Weiterhin werden die Zertifikate geladen und die Verbindung zum Broker gesetzt.

In der loop-Methode erfolgt durch den Aufruf der reconnect-Methode die Verbindung zum Broker. Die Daten des Sensors werden kontinuierlich ausgelesen. Wird eine Bewegung erkannt, wird über den Broker die Nachricht „on“ auf dem Command-Topic veröffentlicht und so die Shelly Steckdose eingeschaltet. Eine zusätzliche Variable namens active dient der Reduzierung der Datenübertragung. Somit wird immer nur dann eine „off“ Nachricht im Topic veröffentlicht, wenn der Shelly bereits an ist, active also gleich true ist, und keine Bewegung registriert wird.

Ausgaben, zum Beispiel ob der ESP8266 sich erfolgreich mit dem WLAN verbunden hat oder eine Verbindung zum Broker erstellt werden konnte, können über den seriellen Monitor eingesehen werden.

Ansprechpartner / in:

Projektkoordination:
Julia Wurm M.A.
E-Mail: julia.wurm@hs-niederrhein.de

Leitung:
Prof. Dr. rer. nat. Claus Brell
E-Mail: claus.brell@hs-niederrhein.de

Forschungsinstitut GEMIT
Der Hochschule Niederrhein
Richard-Wagner-Str. 97
41065 Mönchengladbach

Die Förderung des Vorhabens erfolgt (bzw. erfolgte) aus Mitteln des Bundesministeriums für Ernährung und Landwirtschaft (BMEL) aufgrund eines Beschlusses des deutschen Bundestages. Die Projektträgerschaft erfolgt (bzw. erfolgte) über die Bundesanstalt für Landwirtschaft und Ernährung (BLE) im Rahmen des Programms zur Innovationsförderung.

Gefördert durch

Bundesministerium für Ernährung und Landwirtschaft

aufgrund eines Beschlusses des Deutschen Bundestages

Projektträger

Bundesanstalt für Landwirtschaft und Ernährung