Hochschule für Technik
und Wirtschaft Berlin
University of Applied Sciences

Luftfeuchtigkeits-Sensornetzwerk zur zeitnahen Detektion von Wasserschäden auf Basis von LoRa(WAN)

Projektabschlussbericht

von

Sidney Göhler und Ilja Buschujew

Fachbereich 1 – Energie und Information –
der Hochschule für Technik und Wirtschaft Berlin

im Modul
Projekt Netzbasierte Systeme

des Studienganges
Informations- und Kommunikationstechnik (M. Eng.)

Tag der Abgabe: 25.02.2022

Technischer Input:	Prof. Dr. Thomas Scheffler
Projektmanagement:	Hanna Full

Inhaltsverzeichnis

Kapitel 1: Einleitung

Die Digitalisierung hat unsere Art und Weise wie die Gesellschaft lebt und wie verrichtete Arbeit wertgeschätzt wird, grundlegend verändert. Es sind nicht mehr die Menschen, sondern Computer und Maschinen, die den Takt vorgeben und die Maßstäbe setzen. Arbeit und soziales Zusammenleben werden in einer kapitalistischen Gesellschaft durch die Digitalisierung neu bestimmt. Begriffe wie Homeoffice und Telearbeit sind aus unserem heutigen Arbeitsleben kaum mehr wegzudenken, was schlussendlich in unserer globalisierten Welt zu einem Optimierungswahn geführt hat. Weitere Folgen sind unter anderem die Privatisierung von Wissen und Information, sowie die Ausbeutung von Mensch und Natur.

Aus diesen und weiteren Gründen wünschen sich immer mehr Menschen einen Rückschritt zu einer Gesellschafft, bei der moralische Werte über den wirtschaftlichen Erfolg gestellt werden. Sie wünschen sich mehr Selbstbestimmung, unter Rücksichtnahme der vorhandenen natürlichen Ressourcen und beteiligten Personen, um Schlussendlich die vorherrschende Ellenbogengesellschaft durch eine sozialere auszutauschen.

Im Diskurs werden unter Anderem Begrenzung Anderer, Grenzsetzung gegenüber Anderen, aber auch Ausgrenzung Anderer bzw. die eigene Ausgrenzung thematisiert und in Frage gestellt, wodurch sich unter anderem die Bewegung der „Urban Commoner“ herauskristallisiert hat.

Urban Commons zielt auf eine Entwicklung von individuellen und gesellschaftlichen Werte und Normen, auf Basis eines Zusammenschlusses einzelner Individuen, um ein bestimmtes Gebiet oder eine bestimmte Ressource unabhängig zu gestalten.

Frei nach dem Motto: „Besitz statt Eigentum: [Es] zählt, wer etwas tatsächlich braucht und gebraucht, und nicht das Recht zum Ausschluss anderer oder zum Verkauf“ und „Beitragen statt Tauschen: tätig werden aus innerer Motivation – bei gesichertem Ressourcenzugang“[Hab15]

1.1 Vorstellung der Projektidee

Wir, Ilja Buschujew und Sidney Göhler, möchten mit unserem Luftfeuchtigkeit-Temperatur-Sensor Netzwerk unseren Beitrag dazu leisten, um prinzipiell jedem die Möglichkeit zu bieten, seine eigenen Daten zu sammeln und diese mit seinem Umfeld zu teilen, um dem sich fortschreitenden Konkurenzgedanken innerhalb seines Umfeldes entgegen zu wirken ohne dabei auf seine individuellen Bedürfnisse verzichten zu müssen.
In erster Linie wollen wir in diesem Projekt[Bus22] ein Produkt entwickeln, welches ausschließlich aus freien Inhalten zusammen gesetzt ist, da wir der Meinung sind, dass die Basis einer werteorientierten Gesellschaft das Teilen von Resscourcen bedeutet.

1.2 Ausgangslage und Zielsetzung

Wie schon im Abschnitt 1.1 beschrieben, versucht unser Projekt „Luftfeuchtigkeits-Sensornetzwerk auf Basis von LoRa(WAN)“ das Thema des „Urban-Commons”, was aus dem englischen kommt und so viel wie: „ gesellschaftliches- oder städisches Gemeingut“ bedeutet, aufzugreifen. Aber was versteht man jetzt genau unter dem Begriff „gesellschaftliches Gemeingut“ eigentlich?

Wir Menschen sind eine soziale und kooperative Spezies, die zu weitaus wundervollen Erzeugnissen fähig ist. Der Begriff der Emergenz stellt ein wunderbares Beispiel dafür dar. Es bezeichnet die Möglichkeit der Herausbildung von neuen Eigenschaften oder Strukturen eines Systems infolge des Zusammenspiels seiner Elemente. So ist das auch in der Gesellschaft; wenn die Menschen zusammen an einer Aufgabe oder einem Projekt arbeiten, können neue Strukturen und Eigenschaften der Gesellschaft daraus wachsen. Das Kollektiv ist also mehr als die Summe der einzelnen Individuen, denn die individuelle Identität ist immer auch Teil kollektiver Identitäten. Es gibt daher kein isoliertes Ich, sondern ein Ich-in-Bezogenheit [BH19]. Unser Identität wird von Anfang an aus Beziehungen zu anderen heraus gebildet.

„Die Welt als Commons zu denken und zu gestalten bedeutet, unsere Kooperationsfähigkeit so zu nutzen, dass sich niemand über den Tisch gezogen fühlt, aber auch niemandem ein Platz am Tisch verweigert wird.“ [BH19].

Unserer Meinung nach betrifft es vor allem, die kooperative Gestaltung des eigenen Wohnumfeldes, welches unabhängig vom Staat, Markt, den sozialen Status, Herkunft,

oder dem eigenen Einkommen stattfinden soll. Dabei spielt die Selbstbestimmung eine zentrale Rolle. Daher haben wir uns auch für das Projekt „Luftfeuchtigkeits-Sensornetzwerk auf Basis von LoRa(WAN)“ entschieden, dass an dem Prinzip des Commons anknüpfen soll.

Bei unserem Projekt soll jeder der Lust oder das Bedürfnis hat die Möglichkeit haben, einen eigenen Luftfeuchtigkeits- und Temperatursensor im Keller anzubringen und so bei Tagen an dem z.B. viel Regen fällt, oder es zu einem Rohrbruch im Keller kommt, wo das Wasser sich ansammelt und eventuell zu Sachschäden oder ähnlichen führen kann, zu warnen und mit anderen Menschen dies zu teilen.

Die Sensorwerte sollen über die LoRa-Funktechnik, dessen Frequenzband, genau wie WLAN oder Bluetooth, im unlizenziertem ISM-Band liegt, versendet und damit keine Gebühren für die Nutzung des Frequenzbandes bezahlt werden. Darüber hinaus weist LoRa eine durchaus hohe Reichweite auf, sodass damit auch mehrere Gebiete gleichzeitig im Umkreis von mehreren Kilometern abgedeckt werden und damit mehr Menschen sich an dem Netzwerk anschließen können, welches bei dem LoRaWAN (Longe Range Wide Area Network) der Fall ist. Jedoch beschränken wir uns in unserem Projekt nur auf eine Punkt-zu-Punkt LoRa Kommunikation, die man aber später noch ausbauen und zu LoRaWAN erweitern könnte.

1.3 Strukturierung des Projektberichtes

Im nachfolgenden Kapitel 2. wird die herangehensweise der Produktentwicklung, verbunden mit dem Pflichtenheft, welches aus der Auswertung des Umfragebogens heraus entsteht, sowie die resultierende theoretische Realisierung der Projektidee im Form eines Systemkonzeptes und Blockschaltbildes. Zum Schluss wird die Art und Weise des Managements für unser Projekt beschrieben.

Im Kapitel 3. wird auf die Grundlagen, wie der Funktionsweise von LoRa und LoRaWAN, sowie dem MQTT-Protokoll, eingegangen.

Im 4. Kapitel wenden wir uns der praktischen Realisierung zu. Dabei beschreiben wir zunächst einmal die Hardware, die wir für das Projekt verwendet haben. Wir gehen auf die Funktionsweise und die besonderen Eigenschaften der Mikrocontroller, der Development-Boards und den Sensortyp ein. In Form eines Schaltplans wird die Verdrahtung der einzelnen Hardware-Komponenten dargestellt und beschrieben.

Im zweiten Teil der praktischen Realisierung wird die Umsetzung in der Software beschrieben. Dabei gehen wir auf die Umsetzung der Punkt-zu-Punkt LoRa-Kommunikation zwischen dem LoRa-Sender und -Empfänger ein. Es wird die Einbindung des MQTT-Protokolls in der Software beschrieben und die Visualisierung der Sensordaten im Form eines Dashboards dargestellt. Abschließend wird der Stromverbrauch im Batteriebetrieb veranschaulicht und ausgewertet.

Abschließend wird im Kapitel 5. die Arbeit mit einem Fazit, im Form der Projektauswertung, der Probleme und Herausforderungen, die während der Arbeit entstanden sind, sowie ein Ausblick auf zukünftige Verbesserungsmöglichkeiten und eines Abschlusswortes, beendet.

Kapitel 2: Projektplanung

2.1 Pflichtenheft

Auch wenn wir in erster Linie ein Produkt aus freien Inhalten entwickeln wollen, müssen wir an einigen Stellen den Kompromiss zwischen freiem Inhalt und Entwicklungsaufwand eingehen, da wir in der zeitlichen Ressource begrenzt sind. Im Zuge der Projektplanung haben wir eine Umfrage durchgeführt, woraus sich unser Pflichtenheft abgeleitet hat.

- **Anschaffungskosten:** Unser Persona möchte maximal 40 Euro in dieses Projekt investieren, um einen funktionsfähigen Funksensor zu erhalten.
- **Laufende Kosten:** Um die laufenden Kostn gering zu halten, soll der Funksensor möglichst stromsparend und wartungsarm sein. Die Hardware sollte ihren Strom über einen Akku bzw. eine Batterie beziehen.
- **Einsatzgebiet:** Da der Sensor am Mikrocontroller im Bereich von -40°C. . . 80°C arbeitet und auch in Gebieten mit einer relativen Feuchtigkeit von bis zu 100% zum Einsatz kommen soll, muss das fertige Endprodukt mindestens diesen Anforderungen entsprechen. Die Entwicklung eines Gehäuses erfolgt aber erst nach Erreichen der Serienreife.
- **Datenschutz:** Der Endnutzer möchte selbst bestimmen, ob und mit wem er seine gesammelten Daten teilt. Des weiteren möchte er selbst bestimmen, ab welchem Zeitpunkt/Schwellwert er über den aktuellen Datenstand informiert wird.

2.1.1 Vorgangsmodell

Scrum ist ein Vorgehensmodell im Projektmanagement, welches seinen Ursprung in der Softwareentwicklung hat. Der Ansatz von Scrum ist das systematische Sammeln von Erfahrungen (empirisch), das kontinuierliche weiterentwickeln bestehender

Module (inkrementell), sowie dem mehrfachen Wiederholen gleicher oder ähnlicher Prozesse (iterativ) und basiert auf der Erkenntnis, dass viele Entwicklungsprojekte zu komplex sind, um sie in einem vollumfänglichen Plan fassen zu können, was wiederum den Grund hat, dass wesentliche Teile der Ursprungsanforderung bzw. deren Lösungsansätze zu beginn unklar sind.

Ein weiteres Merkmal von Scrum ist, dass neben dem Produkt auch die Planung kontinuierlich verändert bzw. weiterentwickelt, wobei der langfristige Plan, auch Product Backlog genannt, iterativ verfeinert und verbessert wird.

Resultierende Arbeitspakete werden zyklisch in sogenannten Sprints detailliert formuliert und in einem Detailplan, auch Sprint Backlog genannt, zur Bearbeitung abgelegt, sodass diese, fokussiert auf die aktuelle Problemstellung, abgearbeitet werden können.

Ziel ist dabei eine schnelle und kostngünstige Entwicklung hochwertiger Produkte, wobei die jeweiligen Anforderungen aus der Anwendersicht formutliert werden.

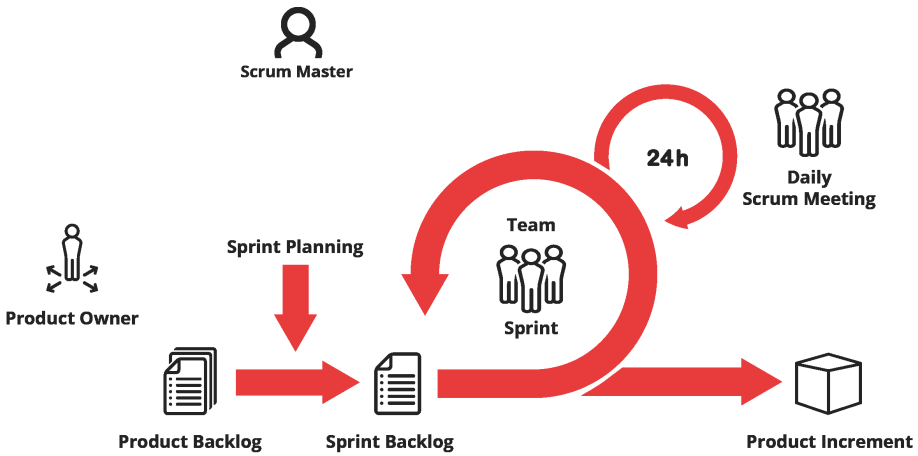

Abbildung 2.1: Agiles arbeiten mit Scrum[Gol18]

Die Verantwortlichkeiten liegt beim sogenannten Scrum-Team, welches sich aus folgenden Rollen ergibt:

Rolle	Besetzung in unserem Projekt	Anmerkung
Product Owner	HTW Berlin	v.d. Prof. Dr. Thomas Scheffler
Scrum Master	-	-
Projektmanager	Sidney Göhler	-
Scrum Team	Sidney Göhler, Ilja Buschujew	-

Tabelle 2.1: Verantwortlichen im Scrum-Team

2.1.2 Projektstruktur

Aus den uns gesetzten Pflichten, haben sich für uns die folgende Projektstruktur grob herauskristalisiert, welche im laufenden Prozess immer weiter in Arbeitspakete verfeinert wurde.

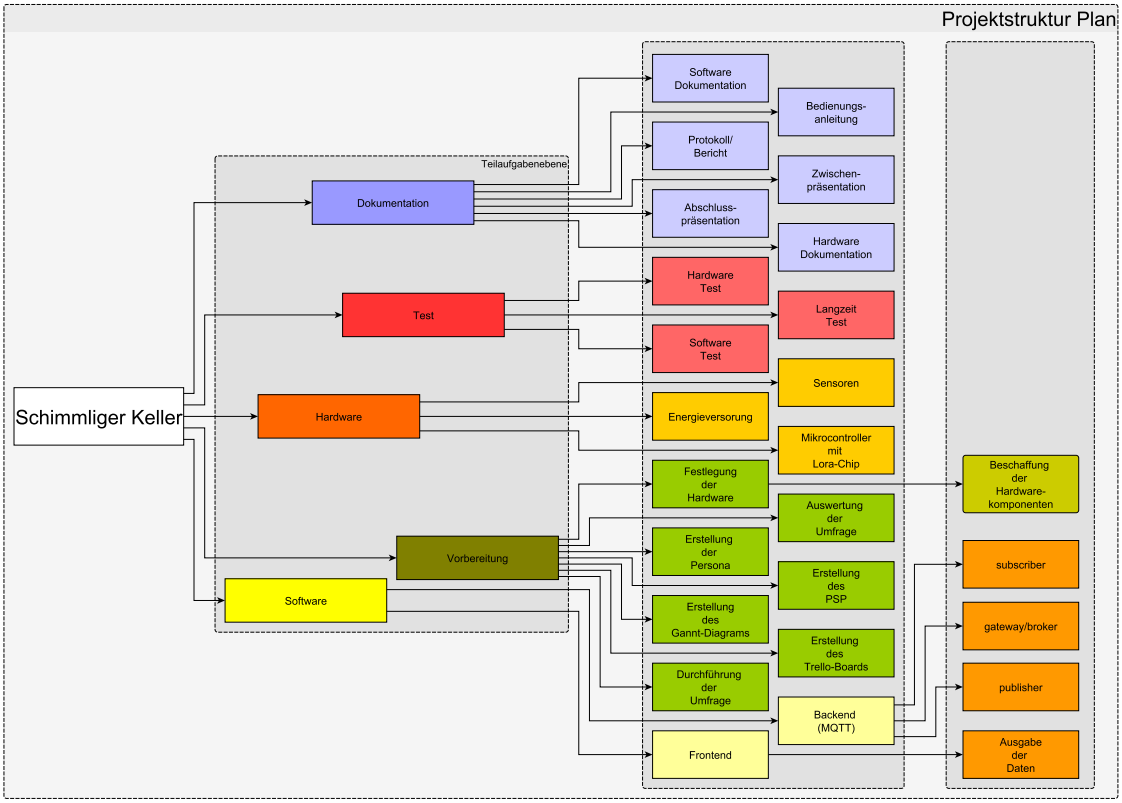

Abbildung 2.2: Projektstrukturplan

Mithilfe des Projektstrukturplanes ließen grob die einzelnen Arbeitspakete ableiten, wobei diese sich, wie bereits erwähnt, immer weiter verfeinert haben. Der enstehende Aufwand ließ sich zu Beginn noch nicht richtig abschätzen, dennoch konnten wir einen groben Zeitplan für die einzelnen Meilensteine abschätzen.

2.1.3 Zeitplanung

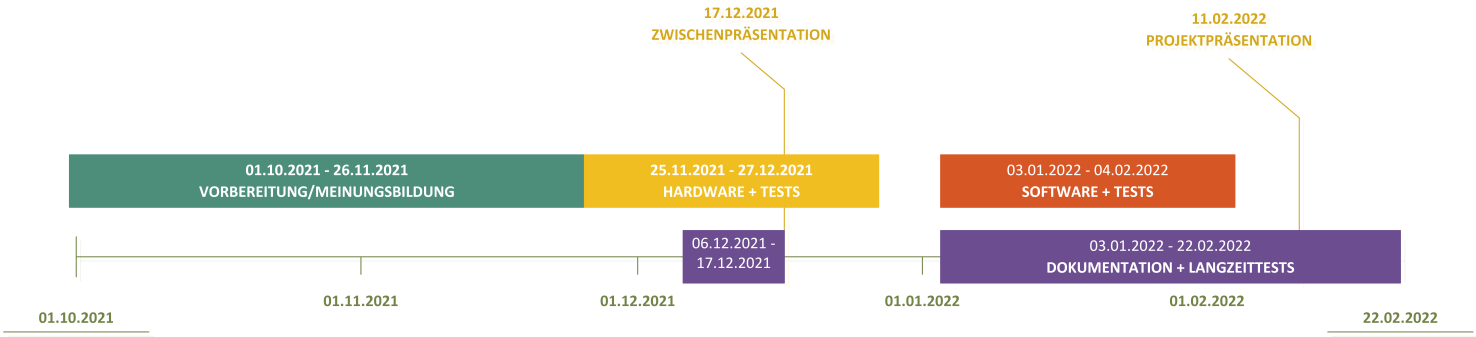

Abbildung 2.3: Übersicht unserer Zeitplanung

Illustriert wird unsere grobe Zeitplanung zu Beginn des Projektes. Wir wollten aufgrund unserer agilen Arbeitsweise nur grobe Zeiträume definieren. Wie sich schlussendlich herausgestellt hat, haben manche Teilaspekte länger, andere wiederum kürzer gerdauert.

Nachfolgend wird eine tabellarische Übersicht unserer Zeitplanung aufgeführt, wobei wir die Daten aus unserem Trello-Board entnehmen:

Vorgangsname	Anfang	Ende	Bearbeiter	Arbeitszeit
Vorbereitung				
Sichtung der Quellenlage	22.10.2021	11.02.2022	S + I	20h
Erstellung Trelloboard	22.10.2021	22.10.2021	S	30m
Erstellung eines Git repositories	22.10.2021	22.10.2021	S	30m
Erstellung des PSP	29.10.2021	29.02.2022	I	2h
Erstellung Zeitplan	01.11.2021	04.11.2022	S	2h
Erstellung/Durchführung der Umfrage	05.11.2021	05.11.2021	S + I	2h
Festlegung/Beschaffung der Hardware	15.11.2021	01.12.2021	S + I	3h
Auswahl/Einrichtung Software IDE	01.12.2021	20.12.2021	S + I	2h
Auswertung der Umfrage	15.11.2021	15.11.2021	S + I	1h
Erstellung des Persona	15.11.2021	15.11.2021	S + I	2h
Hardware				
Inbetriebnahme der Hardware	01.12.2021	27.12.2021	S + I	2h
Software				
µC Management	03.01.2022	10.01.2022	S + I	2h
Integration der Sensoren	10.01.2022	04.02.2022	S	6h
Einrichtung einer P2P LoRa-Kommunikation	17.01.2022	11.02.2022	S	6h
Entwicklung eines MQTT Publishers	17.01.2022	11.02.2022	S + I	2h
Entwicklung eines MQTT Subscribers	17.01.2022	11.02.2022	S + I	1h
Einrichtung einer grafischen Schnittstelle	10.01.2022	11.02.2022	I	2h
Softwaretests und Bugfixes	01.10.2021	20.02.2022	S + I	6h
Nachbereitung				
Vorbereitung der Zwischenpräsentation	16.12.2021	17.12.2021	S	3h
Vorbereitung der Abschlusspräsentation	01.02.2022	11.02.2022	S + I	6h
Dokumentation	14.02.2022	25.02.2022	S + I	8h
Langzeittests	20.01.2022	11.02.2022	S + I	2h
Summe				81h

Tabelle 2.2: Übersicht der Arbeitspakete und Arbeitszeiten

2.1.4 Kostenaufstellung

Auch wenn wir unser Projekt in erster Linie als Freie Software bereitstellen wollen Aus unseren Anforderungen geht hervor, dass die meisten potentiellen Nutzer möglichst wenig für unser Produkt bezahlen möchten. Wie bei

PyCom			**DIY**		
Name	Anzahl	Kosten	Name	Anzahl	Kosten
LoPy4	2	38,45 €	ESP32 DevKit	1	9,99 €
Pytrack	1	40,65 €	Breadboard	1	5,99 €
Antennen-Kit	1	9,00 €	LoRa Transceiver + Antenne	1	11,98 €
Batterie-Halterung	1	9,00 €	Batterie-Halterung	1	9,00 €
DHT22 Sensor	1	9,99 €	DHT22 Sensor	1	9,99 €
			Passive Bauelemente		2,00 €
Gesamtkosten		145,54 €	**Gesamtkosten**		48,95 €

Tabelle 2.3: Kostenaufstellung für das Projekt

Anzumerken ist hier, dass die Entwicklung des Produktes mithilfe der PyCom Plattfrom zwar deutlich kostenintensiver ist, aber besonders für einen Prototypen doch recht komfortabel, da jeder einzelne Baustein dafür gemacht ist, miteinander zu funktionieren.
Tendenziell ließe sich aber mit einem S̈elbstbaud̈er Preis um knapp die hälfte reduzieren.
Neben der PyCom Plattform existieren noch andere Entwicklungsplattformen, wie z.B. das *TOOGOO WiFi ESP-32 Entwicklungs Board*, welches den LoRa Transreceiver implementiert und teilweise auch schon eine Halterung für Batterien anbietet.
Anzumerken ist auch, das der zweite LoPy4 entfallen würde, wenn man sich direkt mittels LoRaWAN an einem Gateway anmeldet und nicht wie wir es gemacht haben, zwei Microcontroller miteinander kommunizieren zu lassen.

2.2 Systemkonzept und theoretische Realisierung

Nachfolgend wird das Systemkozept und die Realisierung für unser Projekt „Luftfeuchtigkeits-Sensornetzwerk zur zeitnahen Detektion von Wasserschäden auf Basis von LoRa(WAN)“ in Form eines Blockschaltbildes beschrieben.

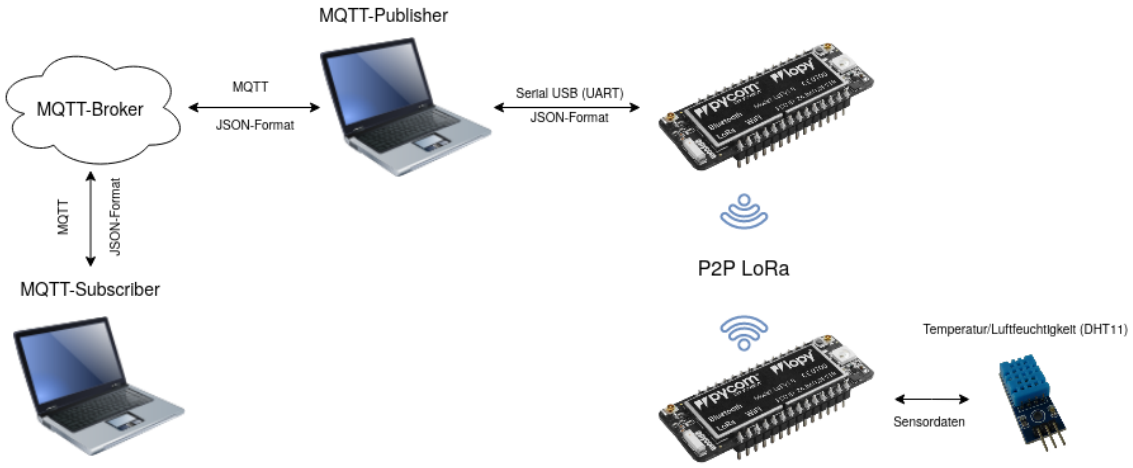

Abbildung 2.4: Systemkonzept für unser Projekt

Für unser Projekt haben wir uns, nach der Beratung mit Prof. Scheffler, dafür entschieden mit den Pycom-Entwicklungsboards zu arbeiten. Diese Boards haben den Vorteil, dass dort schon ein LoRa-Transceiver-Chip integriert ist.

Da nun sich ein Board mit dem Sensor für die Ausmessung der Luftfeuchtigkeit im Keller befinden soll, wird das andere Board dafür benötigt, um die Sensordaten über LoRa-Kommunikation, in einem Bereich wo es eine Internetverbindung gibt, zu empfangen und über das MQTT-Protokoll an den MQTT-Broker weiterzuleiten, damit andere, die an den Sensordaten interessiert sind, darauf zugreifen können. Als Datenformat wurde sowohl für die Übertragung mittels LoRa, als auch MQTT, JSON verwendet.

Zur Weiterleitung der Sensordaten an den MQTT-Broker wurden die von dem Board empfangenen Sensordaten mithilfe einer UART-Verbindung am Computer ausgelesen und in Form eines Publisher-Clients versendet. Der Subscriber-Client kann nun die jeweiligen Sensorwerte auf den jeweiligen Topics, die er einsehen möchte, empfangen.

Für die Visualisierung der Sensordaten haben wir uns für die Adafruit IO Plattform entschieden[1]. Diese Plattform bietet abgesehen von den tollen Visualisierungen der Sensorwerte im Form eines Dashboards, auch einen MQTT-Broker an, den man mit einigen bestimmten Einschränkungen bei der kostenlosen Variante für die eigene Anwendung nutzen kann.

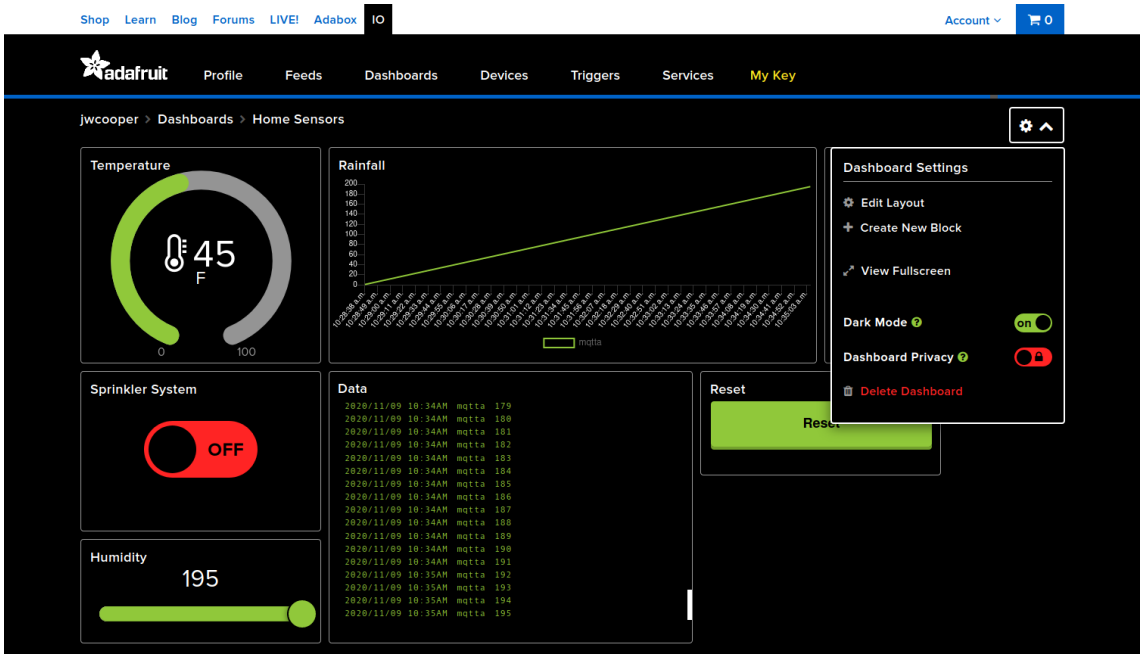

Abbildung 2.5: Beispielhaftes Adafruit Dashboard[ada20]

[1] https://io.adafruit.com/

Kapitel 3: Grundlagen

3.1 LoRa und LoRaWAN

3.1.1 LoRa

Der Begriff LoRa steht für **L**ong **R**ange und definiert dabei ein für weite Strecken ausgelegtes, funkbasiertes Übertragungsverfahren auf der Bitübertragungsschicht (physical layer) im OSI-Schichtenmodell. Es wurde von Nicolas Sornin und Olivier Seller in deren französischen Firma Cycleo, welche später von Semtech Corporation abgekauft wurde, im Jahr 2009 entwickelt [Sla20].

LPWAN LoRa kann zu den sogenannten **L**ow-**P**ower-**W**ide-**A**rea-**N**etwork (LPWAN) Technologien zugeordnet werden, die einen energiesparsamen Betrieb und eine hohe Übertragunsreichweite aufweisen. Im Vergleich zu WLAN oder Mobilfunk, fällt jedoch die Datenrate bei diesen Technologien relativ gering aus, sodass diese hauptsächlich bei drahtlosen Sensornetzwerken Anwendung finden, wo es darum geht Sensordaten mit einer geringen Datenrate über weite Funkstrecken zu übertragen. Ein Vergleich zu den funkbasierten Technologien (WLAN, LPWAN und Mobilfunk) stellt die Abb. 3.1 dar.

Frequenzband Die hohe Reichweite, bei gleichzeitig energiesparsamem Betrieb, erzielen die LPWAN Technologien mithilfe von Frequenzen unterhalb des 1 GHz Bereiches. Da LoRa ebenfalls zu den LPWAN Technologien zählt, nutzt diese in Europa die lizenfreien 433 und 868 MHz ISM-Frequenzbänder. Die Frequenzbänder unterscheiden sich jenach Land und Region auf der ganzen Welt. In den USA z.B. liegt der nutzbare Frequenzband bei 915 MHz, genauer gesagt zwischen 902 MHz und 928 MHz. Das 433 MHz Frequenzband ist jedoch nur für unidirektionale Übertragung, also entweder nur für Down- oder Uplink-Übertragung, vorgesehen. Hingegen ist das

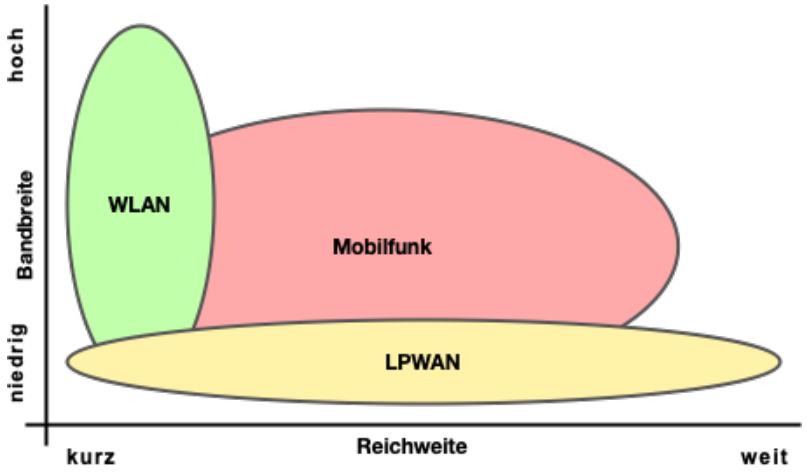

Abbildung 3.1: Vergleich zwischen WLAN, Mobilfunk und LPWAN bezüglich der Bandbreite und Reichweite [kom22]

868 MHz Band bidirektional, das heißt, dass sowohl Downlink-, als auch Uplink-Übertragung möglich ist [kom22].

Funkkanal Die jeweiligen Frequenzbänder haben eine bestimmte Frequenzbandbreite und werden wiederum in sogenannte Funkkanäle unterteilt, mit einer bestimmten Kanalbandbreite, um das gegenseitige stören gleicher Frequenzen zu minimieren. So weist z.B. das 868 MHz Frequenzband eine Frequenzbandbreite zwischen 863 und 870 MHz auf [Sta20]. So können benachbarte Funkknoten, die sich im gleichen 868 MHz Frequenzband befinden, aber einen unterschiedlichen Funkkanal nutzen, gleichzeitig senden und empfangen, ohne sich dabei zu stören.

Duty Cycle Darüberhinaus werden weitere regulatorische Maßnahmen ergriffen, wie die Festlegung einer Zeit in Form eines **D**uty-**C**ycles (DC), welches angibt, wie lange ein Funkknoten auf das Medium pro Tag zugreifen darf. Dieser ist ein prozentualer Wert und liegt bei LoRa normalerweise bei 1% oder 10% je nach Funkkanal und Sendeleistung [Sta20].

Die Abbildung 3.2 zeigt das 868 MHz-Frequenzband mit den jeweiligen Funkkanälen, deren Bandbreite (in kHz), der äquivalenten Strahlungsleistung (in dBm) und des jeweiligen Duty Cycles an .

Zugriffsverfahren Weitere Möglichkeiten zur Verhinderung der gegenseitigen Störung, sind die definierten Zugriffsverfahren auf das Medium, die bei LPWAN auch

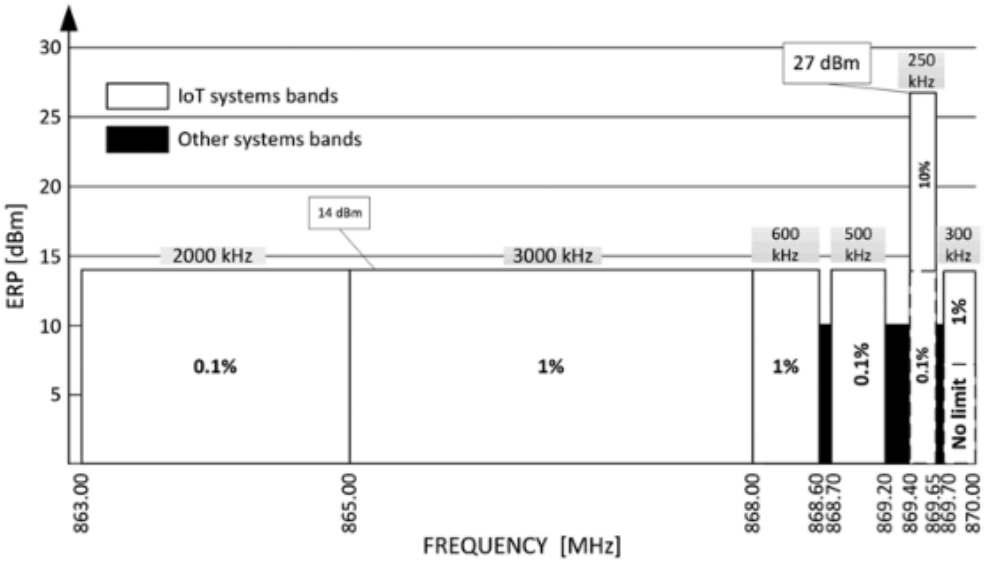

Abbildung 3.2: Aufteilung des 868-ISM-Bandes in Funkkanäle nach ETSI EN 300 220-2 [Sta20]

als **P**olite-**M**edium-**A**ccess (PMA) heißen. Darunter zählt das sogenannte **C**lear-**C**hannel-**A**ssessment (CCA), welches wiederum in **A**daptive-**F**requency-**A**gility (AFA) und **L**isten-**B**efore-**T**alk (LBT) eingeteilt werden kann. Dabei ähnelt das CCA-Zugriffsverfahren sehr stark dem Carrier-Sense-Multiple-Access/Collision-Avoidance (CSMA/CA) Zugriffsverfahren bei z.B. WLAN.

AFA Bei dem AFA Verfahren, wird einerseits die Datenrate hinsichtlich der Kanaleigenschaften angespasst und andererseits ein geeigneter Funkkanal mittels spezieller Algorithmen, die die optimale Lastverteilung zwischen den jeweiligen Funkkanälen berechnen, ausgewählt.

LBT Das LBT Verfahren stellt sicher, dass immer nur ein Funkgerät auf einen Funkkanal zugreifen kann, um die gegenseitige Störung zu verhindern. So „lauscht“ (listen) es zunächst einmal auf den Funkkanal, auf dem es zugreifen möchte, um festzustellen ob es frei ist, bevor es darauf zu „sprechen“ (talk) beginnt. Wenn das Funkgerät feststellt, dass der Kanal momentan besetzt ist, dann wartet es eine gewisse Zeit, die normalerweise zwischen 5 und 10 ms beträgt, ab, bevor es nochmal versucht auf den Kanal zuzugreifen [Sta20].

Reichweite und Datenrate Die Reichweite von LoRa beträgt zwischen 2-5 km in urbanen und 5-15 km in ländlichen Regionen mit keiner direkten Sichtverbindung

(non-line-of-sight). Wenn es eine direkte Sichtverbindung (line-of-sight) zwischen den Funkmasten gibt, dann kann die Reichweite auch weit über 15 km erreichen. LoRa hat zudem eine sehr gute Gebäudedurchdringung, was es zu einer idealen Technologie für den Einsatz in Kellern ausmacht. Die Datenrate liegt dabei im Bereich zwischen 0.3 kBit/s und 5.5 kBit/s und die maximale Übertragungsleistung bei 25 mW [Lie22]. Der aktuelle Rekord, bei dem die Sensorwerte noch empfangen konnten, liegt bei einer unglaublichen Reichweite von 766 km, welcher im Juli 2019 aufgestellt wurde[1].

CSS-Modulationsverfahren Die hohe Reichweite, bei gleichzeitig geringem Energieverbrauch, ist abgesehen von der niedrigen Frequenz, dem speziellen Modulationsverfahren zu verdanken. LoRa benutzt die sogenannte **C**hirp-**S**pread-**S**pectrum (CSS) Modulationstechnik, bei der sich die Frequenz eines Signals (0 oder 1) innerhalb eines definierten Frequenzbereiches gleichmäßig ändert (siehe Abb. 3.3). Dabei steht der Begriff Chirp, nicht nur für das Zwitschern, sondern auch für **C**ompressed **H**igh **I**ntensity **R**adar **P**ulse. Dieses Modulationsverfahren existierte schon vor der Erfindung von LoRa und zwar bei der Unterwasserkommunikation und den Sonargeräten im maritimen Sektor, sowie der Radartechnologie in der Luftfahrt [Sla20].

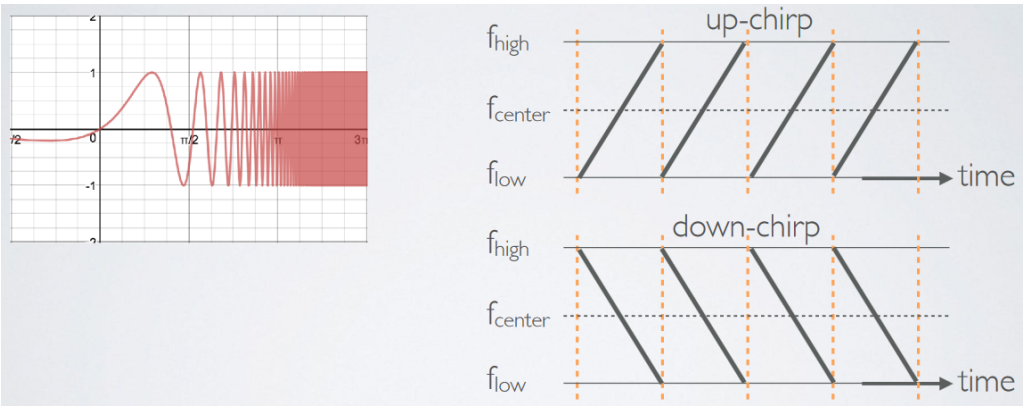

Abbildung 3.3: Darstellung eines Chirp-Signals [Lie22]

Die gleichmäßige Änderung der Frequenz innerhalb eines definierten Frequenzbereiches wird als chirp bezeichnet und ist in der Natur weit verbreitet. So kommunizieren z.B. Vögel, Fledermäuse oder Delphine ebenfalls über die zeitliche Änderung der Frequenz. Dies hat den Vorteil, dass äußere Störungen einen geringeren Einfluss auf die Signalqualität ausüben und somit das Signal-Rausch-Verhältnis sich verbessert. Außerdem ist das CSS Verfahren weniger anfällig gegenüber den sogenannten

[1] https://tech-journal.semtech.com/university-of-zaragoza-breaks-long-range-lorawan-based-signal-record

Doppler-Effekt, bei dem es zu Signalverzerrungen bei dem Empfänger, wenn der Sender oder Empfänger in Bewegung sind, kommt.

Spreizfaktor Durch das hinzufügen eines sogenannten Spreizfaktors (Spreading Factor oder SF), kann außerdem die Geschwindigkeit der zeitlichen Änderung der Frequenz, also die Chirprate geändert und damit auch die Datenrate variiert werden. LoRa definiert dabei insgesamt sechs Abstufungen des Spreizfaktors (SF7 bis inkl. SF12). Die Zahl des Spreizfaktos besagt auch, wieviele Bits in einem Symbol für die Datenübertragung verwendet werden. Je mehr Bits in einem Symbol, desto mehr Daten können in einem Datenpaket übertragen werden [Gho17].

Grundsätzlich gilt; je kleiner der Spreizfaktor, bei gleichbleibender Bandbreite, desto höher ist die Datenrate der Übertragung. Jedoch nimmt die Reichweite der Datenübertragung, je kleiner der Spreizfaktor ist, ab. Bei der Erhöhung des Spreizfaktors um einen Wert, halbiert sich die Chirprate und damit auch die Datenrate, aber die Reichweite wird erhöht [The22].

Auf der anderen Seite nimmt die Batterielaufzeit mit abnehmenden Spreizfaktor zu, da die Zeit, bei der das LoRa-Funkmodul zur Übertragung der Daten aktiv sein muss, aufgrund der erhöhten Chirp- bzw. Datenrate, abnimmt.

Die Abb. 3.4 veranschaulicht die Chirp-Signale, die den gleichen Frequenzbereich, aber unterschiedliche Spreizfaktoren nutzen, gegenüber der Zeit, die sie dafür benötigen.

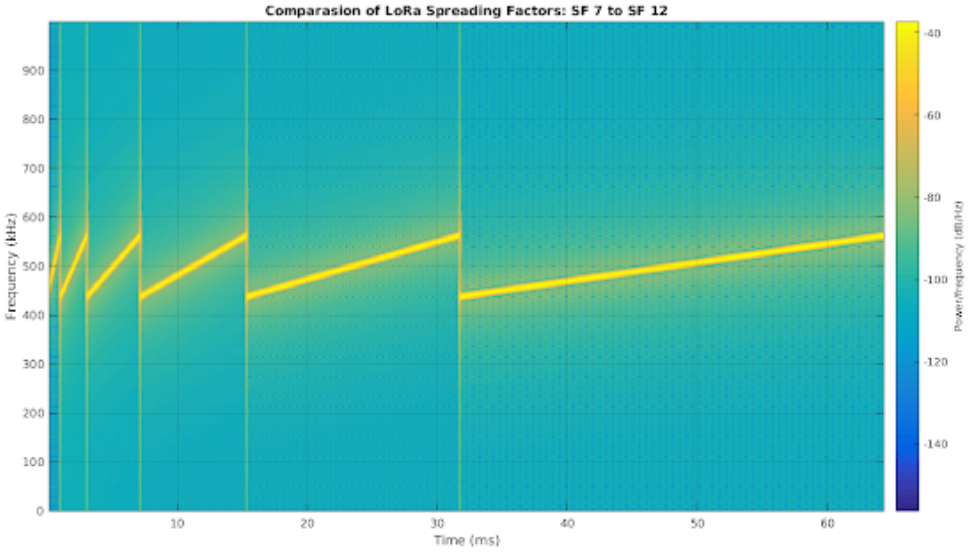

Abbildung 3.4: Chirprate gegenüber der Zeit bei unterschiedlichen SF [Gho17]

Darüberhinaus kann durch die Erhöhung der Kanalbandbreite, bei gleichbleibenden Spreizfaktor, ebenfalls eine Erhöhung der Datenrate bewirkt werden. LoRa nutzt dabei die Kanalbandbreiten 125 kHz, 250 kHz und 500 kHz.

Mit der Änderung des Spreizfaktors und damit auch der Datenrate, können „Datenstaus“ im Netzwerk aktiv reguliert werden [The22].

3.1.2 LoRaWAN

Da LoRa anfangs nur für die Bitübertragungsschicht definiert wurde, musste ein Mechanismus entwickelt werden, bei dem die LoRa-Endgeräte addressiert untereinander und mit den LoRa-Gateways kommunizieren können. Es wurde zunächst einmal das LoRaMAC-Protokoll für die Sicherungsschicht (MAC layer) definiert. Später wurde die LoRa-Alliance[2] gegründet, die es zu LoRa-**W**ide-**A**rea-**N**etwork (LoRaWAN) umbenannt hat [Sla20].

LoRaWAN nutzt für die Kommunikation zwischen den LoRa-Endgeräten und LoRa-Gateways die Sterntopologie (siehe Abb. 3.5).

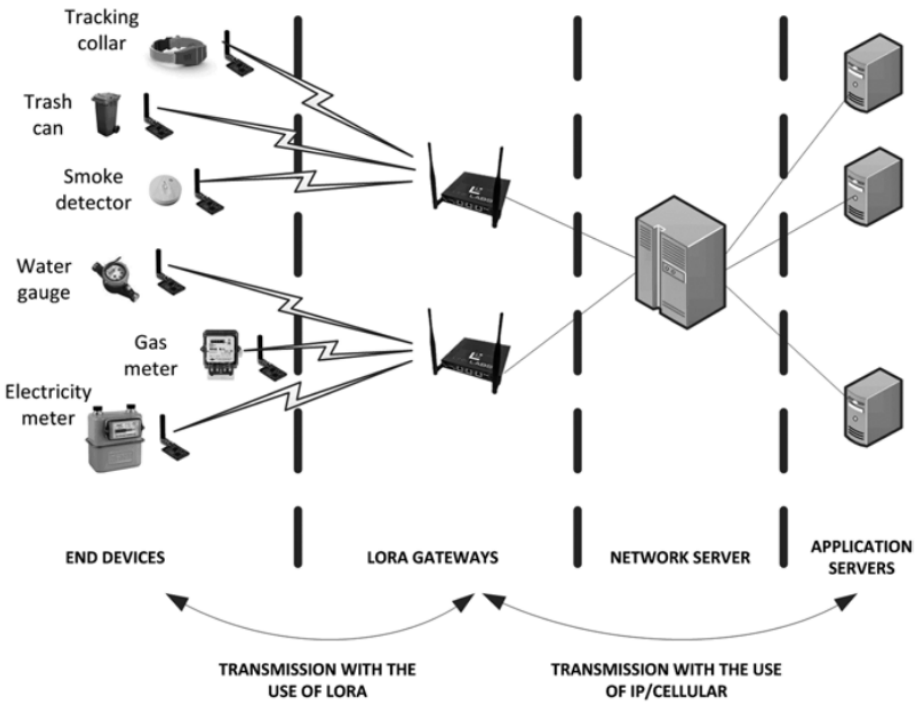

Abbildung 3.5: LoRaWAN Topologie [Sta20]

[2]https://lora-alliance.org/

Die LoRa-Endgeräte sammeln z.B. Sensor- und Messdaten und senden diese mithilfe von LoRa-Funktechnik an die LoRa-Gateways.

Die LoRa-Gateways leiten die Daten dann über das Internet-Protokoll (IP), oder Mobilfunk an die Netzwerk-Server (Network Server) weiter. Es können mehrere LoRa-Gateways, die örtlich von einander getrennt sind, dazu genutzt werden, die Sensordaten der selben LoRa-Endgeräte zu empfangen. Durch die redundante Übertragung der selben Daten an mehrere Gateways, wird die Zuverlässigkeit und damit auch die Ausfallsicherheit der Datenübertragung verbessert.

Der Netzwerk-Server ist für die Entfernung der duplizierten Datenpakete, deren Dekodierung und für das Erstellen neuer Datenpakete, die an die LoRa-Endgeräte gesendet werden, zuständig.

Schließlich verarbeiten die Anwendungs-Server (Application Server) die von den Netzwerk-Server erhaltene Daten für eigene Zwecke dann weiter.

3.2 MQTT

MQTT steht für **M**essage-**Q**ueue-**T**elemetry-**T**ransport und ist ein Machine-to-Machine (M2M) Kommunikationsprotokoll, welches ursprünglich von IBM zur Überwachung von Gaspipelines entwickelt, es jedoch seit 2013 von **O**rganization for **A**dvanced of **S**tructured **I**nformation **S**tandards (OASIS) als Open-Source-Protokoll übernommen wurde und weiterentwickelt wird [MB19].

Es wird auf der Anwendungsschicht im OSI-Schichtenmodell eingegliedert und weist folgende Eigenschaften auf:

- geringe Komplexität
- hohe Flexibilität
- geringer Protokoll-Overhead
- Verwendung von TCP als Transportprotokoll

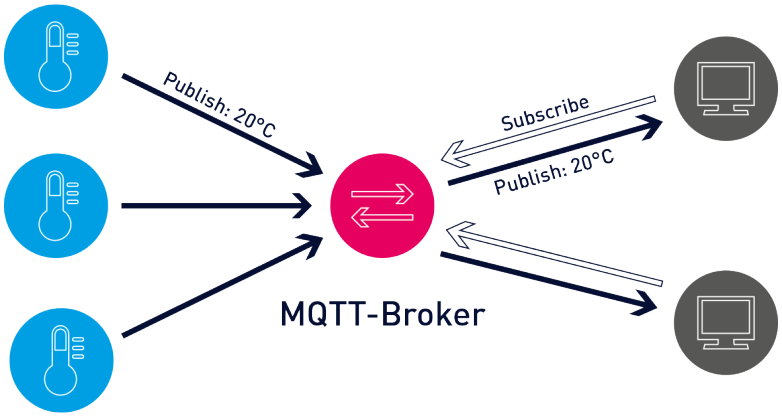

Abbildung 3.6: MQTT Publish-Subscribe-Architektur [AG22]

Publish-Subscribe-Architektur Im Gegensatz zu der klassischen Client-Server-Architektur, verwendet MQTT die sogenannte Publish-Subscribe-Architektur (siehe Abb. 3.6).

Die MQTT-Publish-Subscribe-Architektur besteht aus drei verschiedenen Komponenten. Einem Publisher, welcher auch als Producer genannt wird, einem Subscriber (dem Consumer) und einer zentralen Komponente, dem Broker. Der Publisher ist dafür zuständig seine Sensor- bzw. Messdaten an den Broker zu schicken. Der Broker, der als ein zentraler Server dient, verteilt diese Daten dann an die jeweiligen Subscriber, die an diesen Daten von dem jeweiligen Publisher interessiert sind. Publisher und Subscriber werden auch als Clients bezeichnet. Ein Client kann sowohl ein Publisher, als auch ein Subscriber sein. Die Clients können aber keine direkte Verbindung mit einander aufbauen, sondern die Kommunikation erfolgt immer stets über den Broke [MB19].

Topics Der Nachrichtenaustausch erfolgt bei MQTT mithilfe von sogenannten Topics. Diese sind dem URL-Schema wie bei HTTP angelehnt und sind nichts weiter als eine Zeichenkette von Wörtern, die über den *Topic-Level-Seperator* »/« getrennt werden. Somit ergibt sich eine Topic-Hierarchie mit verschiedenen *Topic-Levels*. Ein Subscriber kann jetzt mithilfe der Topics, auf die Sensorwerte der gewünschten Publisher zugreifen (subscriben).

Topic Levels Die oberste Ebene ist das Level-0-Topic. Betrachtet man das beispielsweise auf ein Gebäude, so stellt es das gesamte Gebäude dar.

```
Gebäude/
```

Die danachfolgende Ebene ist das Level-1-Topic, welches im Beispiel des Gebäudes die Etagen darstellen könnte.

```
Gebäude/Etage/
```

Das Level-2-Topic wären die einzelnen Wohnungen und das Level-3-Topic die einzelnen Zimmer.

```
Gebäude/Etage/Wohnung/Zimmer/
```

Das Level-4-Topic könnten die unterschiedlichen Sensorwerte, also z.B. Temperatur oder Luftfeuchtigkeit sein.

```
Gebäude/Etage/Wohnung/Zimmer/Temperatur
Gebäude/Etage/Wohnung/Zimmer/Luftfeuchtigkeit
```

Wildcards Mit Hilfe der Wildcards kann das Subscriben für Clients vereinfacht werden. Man unterscheidet dabei zwischen den Single-Level- und Multi-Level-Wildcards. So können z.B. die Temperaturwerte für alle Etagen ausgewählt werden, wenn man im Level-1-Topic das Single-Level-Wildcard-Zeichen setzt. Diese werden mit dem Plus-Zeichen (+) definiert.

```
Gebäude/+/Wohnung/Zimmer/Temperatur
```

Mit dem Multi-Level-Wildcard, welches mit dem Hashtag-Zeichen (#) definiert wird, können, wie der Name schon sagt, mehrere Topic-Levels, die nach dem Wildcard kommen, ausgewählt werden. Ein Beispiel mit zwei Zimmern in einer Wohnung und das setzen des Multi-Level-Wildcards [MB19]:

```
Gebäude/Etage-1/Wohnung-1/#
```

Dann werden folgende Topics ausgewählt:

```
Gebäude/Etage-1/Wohnung-1/Zimmer-1/Temperatur
Gebäude/Etage-1/Wohnung-1/Zimmer-1/Luftfeuchtigkeit
Gebäude/Etage-1/Wohnung-1/Zimmer-2/Temperatur
Gebäude/Etage-1/Wohnung-1/Zimmer-2/Luftfeuchtigkeit
```

Quality of Service Das Quality of Service auch QoS genannt, wird bei MQTT der Grad der Zuverlässigkeit für die Datenübertragung bezeichnet. Das MQTT Protokoll definiert dabei drei Levels von QoS. Beim QoS-Level 0 wird jede Nachricht höchstens einmal versendet. Beim QoS-Level 1 wird jede Nachricht mindestens einmal versendet und beim QoS-Level 2 wird jede Nachricht genau einmal versendet. Bei dem QoS-Level 0, bei dem die Nachricht höchstens einmal versendet wird, gibt es keine Garantie, dass der Empfänger diese auch wirklich erhält. Wobei das QoS-Level 2 die Garantie des Empfanges gewährleistet. Jedoch wird mit dem Erhöhen des QoS-Levels auch mehr traffic generiert, weil damit mehr Nachrichten für die Empfangsbestätigung zwischen den Publisher und Subscriber versendet werden.

Kapitel 4: Praktische Umsetzung

4.1 Verwendete Hardware

4.1.1 LoPy4-Development-Board

Das LoPy4 ist ein von Pycom Ltd. hergestelltes Development-Board für die Anwendungen im IoT-Bereich. Es unterstützt viele Schnittstellen und Übertragungsprotokolle, die für IoT-Anwendung von großer Relevanz sind. Die Abbildung **??** stellt die Funktionalitäten des LoPy4-Development-Boards in Form eines Blockdiagrammes dar.

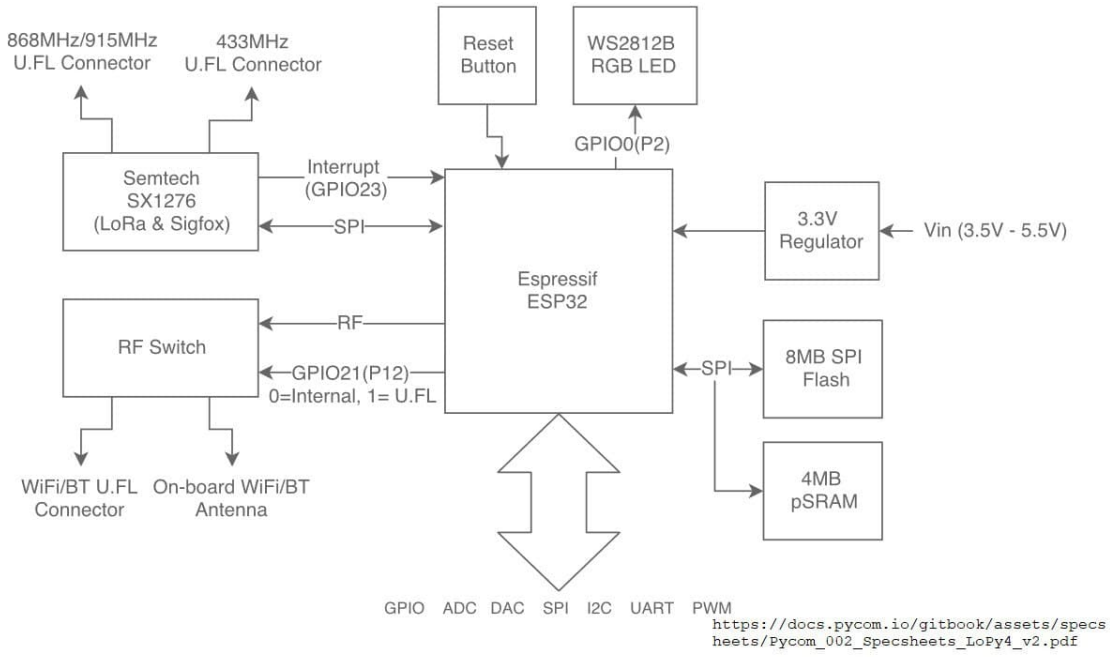

Abbildung 4.1: LoPy4-Blockdiagramm
[Pl22]

Im Kern befindet sich ein Espressif ESP32 mit Xtensa dual-core-32-bit LX6 Mikroprozessor. Als Speicher verwendet es 520KB + 4MB RAM und einen externen 8MB

Flash-Speicher, wohin auch der Code dann geladen wird. Zur Ein-/Ausgabe und der Steuerung von Sensoren und Aktoren, bietet das Board vielseitige Peripherien wie GPIO, ADC, DAC, SPI, I2C, UART und PWM an.

Darüberhinaus verfügt das Entwicklungsboard über WLAN (802.11b/g/n/e/i), Bluetooth/Bluetooth Low Energy (BLE) v4.2, sowie LoRa und Sigfox Protokolle. Es lässt sich leicht mit einem Breadboard benutzen, um damit einfache Schaltungen, ohne zu löten, testen zu können.

Für die WLAN und Bluetooth Anwendung kann man entweder die interne, im Board eingebaute, oder eine externe Antenne verwenden. Bei der Nutzung von LoRa und Sigfox muss man jedoch darauf achten, dass man unbedingt eine externe Antenne verwendet, denn sonst könnte damit der LoRa-Chip beschädigt und unbrauchbar gemacht werden. Dafür gibt es zwei Anschlussmöglichkeiten; einmal für das 868 MHz und einmal für das 433 MHz Frequenzband. Dafür wird der SX1276 LoRa-Transceiver-Chip der Firma Semtech verwendet.

Für die Spannungsversorgung wird lediglich ein Mikro-USB-Kabel, welches an eine USB-Schnittstelle am Computer das Board mit 3.3-5.5V versorgt, benötigt. Neben der Spannungsversorgung kann damit auch das Board direkt programmiert werden.

Für die Programmierung des Boards kann man auf unterschiedliche Entwicklungsumgebungen zurückgreifen. So werden die Entwicklungsumgebungen wie Atom und Visual Studio Code (VSC) von dem Hersteller mit einem sogenannten Pymakr-Plugin für einen reibungslosen Upload und das Flashen des Boards unterstützt. Wenn das Benutzen des Plugins nicht möglich ist, wie unter anderem es teilweise bei uns der Fall war, so kann man auf das von PyCom angebotene File-Transfer-Protocol (FTP) zugreifen. Dafür muss man zunächst einmal einen FTP-Server auf dem Board erzeugen, mithilfe dessen man dann auf das Filesystem des Boards Zugriff hat. Damit der Code im Board läuft, muss man lediglich die jeweiligen Dateien (hauptsächlich die main.py Datei) auf das Board mit FTP-Befehlen in den Flash-Speicher kopieren.

Für die Unterstützung von LoRa bietet Pycom zwei Möglichkeiten an: LoRa-RAW und LoRaWAN. Mit LoRa-RAW kann eine direkte Punkt-zu-Punkt Verbindung für die Kommunikation zwischen zwei LoRa-Knoten hergestellt werden. Mit LoRaWAN kann darüberhinaus der LoRa-Knoten in das TheThingsNetwork (TTN) oder das Chirpstack-Netzwerk eingebunden werden. Die Bandbreite kann zwischen den Werten 125, 250 und 500 kHz variiert werden. Außerdem kann man den Spreizfaktor zwischen 7 und 12 auswählen. Für die Synchronisation kann die Preamble mit der Anzahl der Chirps angepasst werden, wobei der Standardwert bei acht liegt.

Wieviele „Chips” ein Symbol definieren, kann man mit der Coderate einstellen. Darüberhinaus ist zu erwähnen, dass PyCom mit PyMesh, auch die Möglichkeit eines LoRa-Mesh-Netzwerkes auf der MAC-Ebene anbietet. Es ist dabei wichtig, dass alle LoRa-Knoten das gleiche Frequenzband, Spreizfaktor und Bandbreite verwenden. Es verwendet dazu ein hierarchisches Mesh-Netzwerkmodell.

4.1.2 Mögliche Schaltung zum Selbstbauen

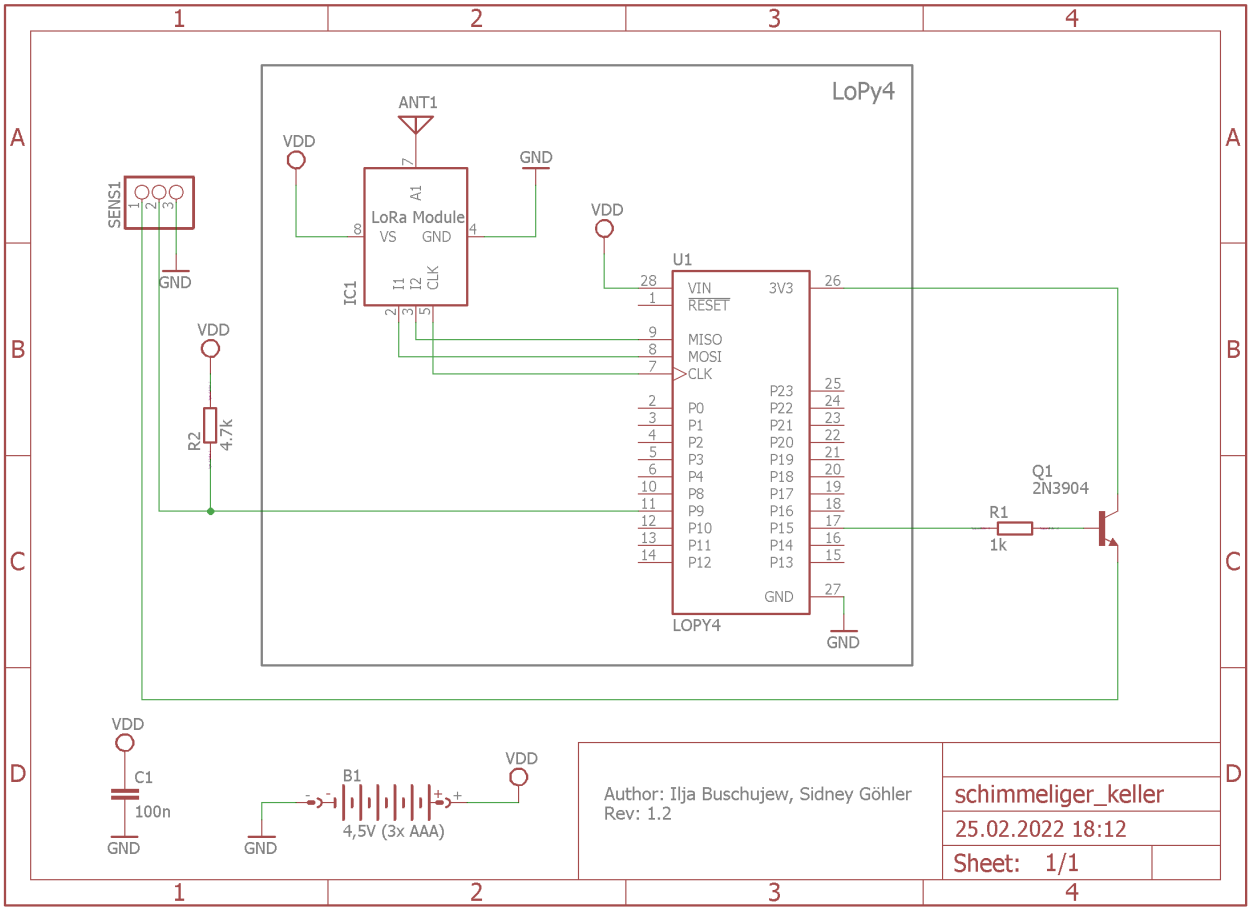

Abbildung 4.2: Möglicher Schaltplan für einen Selbstbau

Der Microcontroller kann ein beliebiger ESP32 basierender Microcontroller sein, wobei der LoPy4 das LoRa Modul bereits integriert hat. Der Transistor kann ein generischern NPN-Transistor sein und der Wiederstand R2 ist theoretisch auch optional, wenn der Microcontroller einen internen Pull-Up Wiederstand dazuschalten kann.

4.1.3 DHT Sensormodul

Bei dem verwendeten Sensor handelt es sich um einen sog. DHT Sensor. Diesen gibt es in zwei verschiedenen Ausführungen, DHT11 und DHT22, wobei sich diese im auswertbaren Messbereich, der Messgenauigkeit und im Preis unterscheiden.

	DHT11	**DHT22**
Betriebsspannung	3 . . . 5 V DC	
Stromverbrauch	max. 2,5 mA während der Konvertierung	
Temperaturbereich	-20. . . 60 °C	-40. . . 80 °C
Temperatur Genauigkeit	± 2,0 °C	± 0,5 °C
Feuchtigkeit Messbereich	20%. . . 90% RH	0%. . . 99,9% RH
Feuchtigkeit Genauigkeit	± 5,0% RH	± 2. . . 5%
Abtastrate	1 Hz	0,5 Hz
Preis (bei reichelt elektronik)	1,80 €	6,80 €

Tabelle 4.1: Vergleich DHT11 zu DHT22

Anzumerken ist noch, dass der DHT22 Sensorungefähr das doppelte Volumen des DHT11 Sensors hat. Interessant ist auch, dass der DHT11 Sensor ungefähr doppelt so häufig angesprochen werden kann, was vermutlich auf seiner weniger komplexen Schaltung beruht.
Für die meisten Anwendungsfälle würde wahrscheinlich ein DHT11 Sensor (oder mehrere parallel geschaltete DHT11 Sensoren) ausreichen, um ein weitgehend zuverlässiges Ergebnis zu erhalten.

Kommunikation mit dem DHT Sensor[Tec12]

Die Kommunikation mit dem DHT Sensor erfolgt über eine einzelne Leitung, was zum einen die Kosten reduziert, zum anderen aber auch die Reichweite der störungfreien Übertragung erhöht. Um den Sensor anzusprechen und die Daten anschließend auszulesen, muss der Datenstrom kodiert werden. Dies geschieht über ein Protokoll, welches die Kommunikation in drei Schritte aufteilt:

- Request (die Anfrage)
- Repsonse (die Antwort des Sensors)
- Data (die vom Sensor übertragenen Daten)

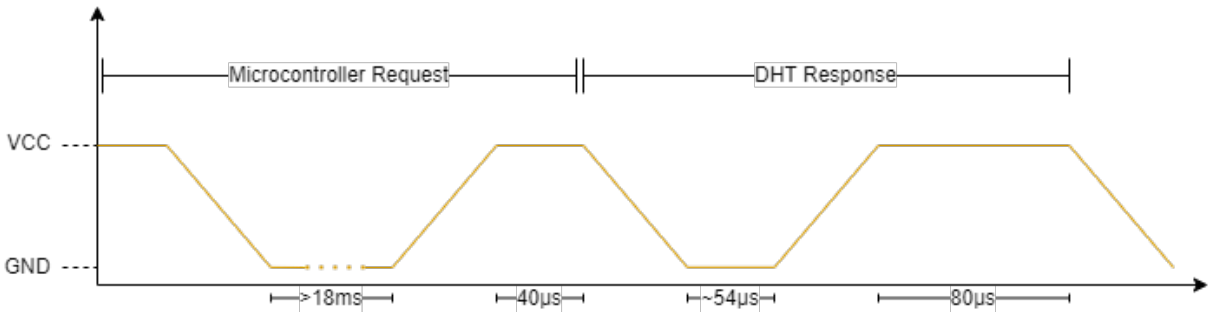

Abbildung 4.3: DHT Kommunikation

Um den DHT Sensor dazu zu bringen, Messewerte zu senden, zieht der Microcontroller den Datenbus für mindestens 18ms auf Masse, um ihn anschließend für 40µs wieder auf Versorgungsspannungsniveau zu ziehen. Dadurch versteht der DHT Sensor, dass er beginnen soll Messwerte zu sammeln.
Der DHT Sensor antwortet zunächst mit einer Sequenz, bestehend aus einem ca. 54µs langen LOW und anschließend 80µs HIGH.

Nachfolgend sendet der DHT Sensor fünf Datenpakete, bestehend aus jeweils 8 Bit, welche die einzelnen Sensor Messwerte mit einer Prüfsumme darstellen. Insgesammt sendet der DHT Sensor somit 40 Bit.

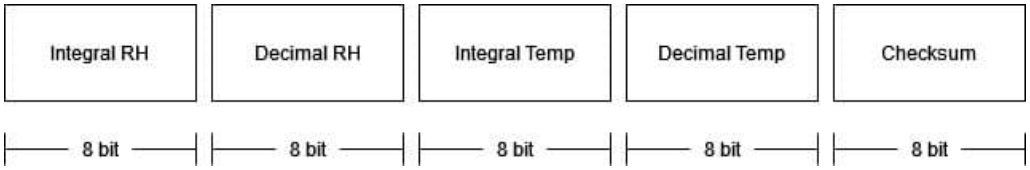

Abbildung 4.4: DHT Paketstruktur

Die einzelnen Sensor Messwerte sind noch in Integral Anteil und Dezimal Anteil unterteilt, wobei die einzelnen Bits sich durch die dauer des HIGH Signals nach einem 54µs langem LOW Singal unterscheiden.

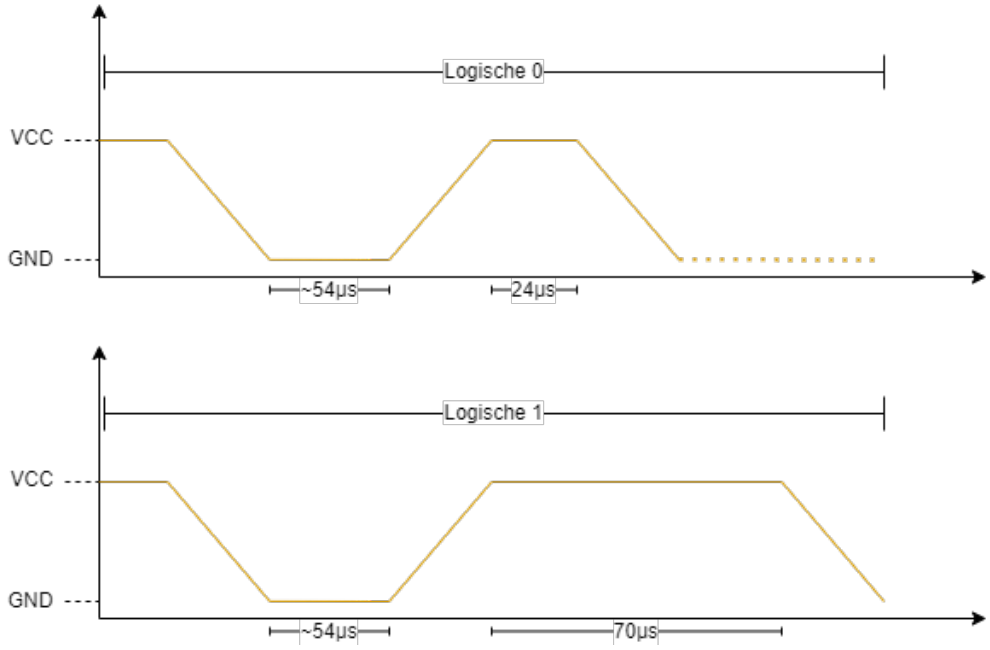

Abbildung 4.5: DHT Bit Identifikation

Abschließend sendet der DHT Sensor ein ca. 54µs langes LOW Signal, wonach der Bus wieder auf Versorgungsspannungsniveau gezogen wird und der Sensor in den Idle Mode geht.

4.1.4 Restliche Hardware

- **Antenne:** Bei der verwendeten Antenne handelt es sich um eine Multiband Antenne, welche für mehrere Freuquenzbänder (unter anderem das LoRa Freuquenzband) genutz werden kann.
- **USB Kabel:** Wir verwenden ein Daten-USB Kabel welches besonders geschirmt ist und eine maximal Länge von ca. 10cm aufweist. Wir hatten teilweise Probleme mit anderen USB Kabeln.

4.2 Beschreibung der Software

Für unser Projekt haben wir drei verschiedene, miteinander interagierende Software Komponenten realisiert, welche über eine Schnittstelle (Interface) miteinander kommunizieren. Der Vorteil einer solchen Architektur ist, dass die einzelnen Komponenten sich unter umständen wiederverwenden lassen und sich im Idealfall so eine Software modular aufbauen lässt. Da wir als Programmiersprache ausschließlich Python bzw. Micropython verwendet haben, könnte man argumentieren, dass unsere Software automatisch Modular ist, da sich in der Theorie alle programmierten Komponenten in Python wiederverwenden lassen. Dies ist aber sehr verallgemeinert gesprochen, da gerade die Programmierung der Mikrocontroller definitiv auch Individualsoftware benötigt, welche sich aber immerhin nicht nur auf einer einzelnen Mikrocontrollerfamilie ausführen funktionieren würde. Anzumerken ist noch, dass für unsere finale Version des Projektes vermutlich nur eine einzelne Softwarekomponente notwendig wäre.
Für die Programmierung der Mikrocontroller verwenden wir die Programmiersprache Micropython, welche eine schlanke und schnelle Implementation der Programmiersprache Python ist, welche für Mikrocontroller optimiert wurde.

4.2.1 1. Komponente: Sensoransteuerung und der Versand der Daten mittels LoRa(WAN)

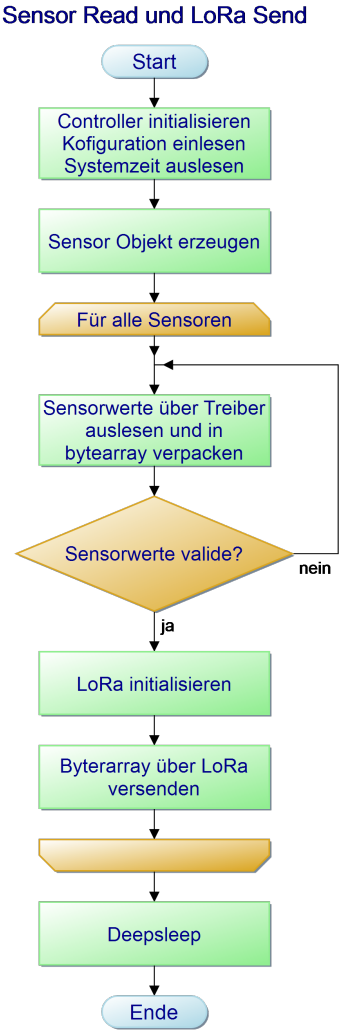

Abbildung 4.6: Programm Ablauf: Komponente 1

Zunächst wird der Microcontroller initialisiert, wobei er standartmäßig mit allen Modulen (WiFi, LoRa, etc.) im aktivierten Zustand startet. Dieser Umstand beruht darauf, dass wir die PyCom Plattform nutzen, welche wenig vorab Konfiguration ermöglicht.
Jegliche Konfigurationsmöglichkeiten unserer Programme haben wir in extern Dateien im JSON Format abgelegt, welche im nächsten Schritt eingelesen werden. In der Konfigurationsdatei können unter Anderem die Dauer, die der Microcontroller im Schlafmodus verbringen soll, sowie die einzelnen Sensoren definiert werden.

Um das anbinden mehrere Sensonren zu ermöglichen haben wir uns dazu entschieden, eine Klasse zu programmieren, welche eben dieses Sensorobjekt darstellen soll. Der Treiber wurde von einem Benutzer auf github veröffentlicht[Jur17], funktioniert aber im Prinzip nach dem oben genannten Verfahren, was bedeutet, dass zunächst eine gewisse Zeit gewartet wird, anschließend das Bus auf GND Niveau gezogen wird, um den Sensor mitzuteilen, dass er doch bitte anfängt Messwerte zu sammeln. Nachdem der Sensor eine Antwort gegeben hat, werden die empfangenen Bits zunächst alle gesammelt und anschließend in 5 einzelne Bytepakete decodiert. Abschließend prüft der Treiber mithilfer der Prüfsumme, ob die empfangenen Daten Sinn machen.
Je nach Sensortyp (DHT11/22) werden die empfangenen Daten nochmals in den richtigen Wertebereich "verschoben".
Anschließend werden für alle initialisierten Sensoren, über den Treiber die aktuellen Messwerte eingelesen und in ein bytearray verpackt, wobei hier schon das erste mal geschaut wird, ob die gemessen Sensorwerte überhaupt Sinn ergeben bzw. in dem vom Hersteller angebenen Bereich fallen, da es selbst bei der korrekten Dekodierung immernoch zu unsinnigen Werten kommen kann. Wurde dieser Test erfolgreich bestanden wird im Microcontroller über das LoRa Modul über eine Bibliothek initialisiert und die ermittelten Werte werden versendet.
Ein weiterer Bestandteil des bytearrays ist die aktuelle Systemzeit, welche verwendet wird, um im späteren Verlauf ermitteln zu können, ob es sich wirklich um ein neues empfangenes Datenpaket handelt, oder ob das Paket in irgendeinem Puffer zunächst verloren gegangen ist und zufällig wieder ins Tageslicht gerückt ist.
Zu guter Letzt wird der Microcontroller in den Schlafmodus versetzt, wobei die Zeit, die er im stromsparenden Modus verbringt benutzerdefiniert ist. Anzumerken ist hier noch, dass der Sensor über einen Transistor ausgeschaltet wird um noch etwas mehr Strom zu sparen und bei der Neuinitialisierung wieder angeschaltet wird. Die Systemzeit läuft auch im Schlafmodus weiter, solange der Microcontroller mit Strom versorgt wird, nur das LoRa-Modul muss neu initialisiert werden.

4.2.2 2. Komponente: Emfangen der Daten und Versand ins Internet

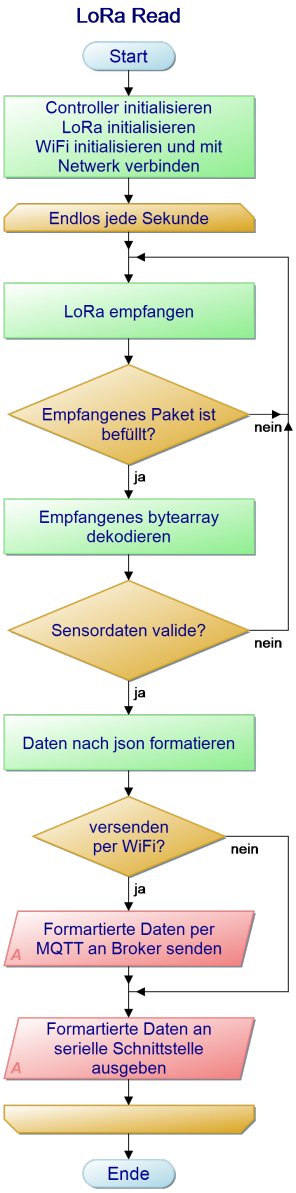

Abbildung 4.7: Programm Ablauf: Komponente 2

Diese Softwarekomponente läuft schlussendlich auf dem LoRa Gateway, bzw. in unserem Fall dem zweiten Microcontroller, welcher in unserem Fall den Empfänger bei der P2P Verbindung darstellt. Wie bei der ersten Komponente, wird zunächst der Microcontroller initialisiert. Des Weiteren werden Konfigurationsdaten eingelesen, welche für die optionale WiFi Verbindung und das senden an die MQTT Broker

benötigt werden. Möchte man die Sensordaten einfach nur über eine serielle Schnittstelle auslesen, können diese Felder entsprechend leer gelassen werden. Außerdem wird in dieser Komponente das LoRa Modul direkt initialisiert, da dies nur einmalig geschehen muss.
Es beginnt eine Endlosschleife, welche in bestimmten Abständen schaut, ob an dem Socket neue, per LoRa emfpangene, Daten aufgetaucht sind. Sollte dies der fall sein, werden die empfangenens bytes decodiert und auf ihre plausibilität gebprüft. Machen die empfangenen Daten Sinn, werden sie ins JSON Format gebracht und über die Serielle Schnitstelle ausgegeben, sollte der Nutzer der Microcontroller mit dem Internet verbunden haben, werden die Daten per MQTT an die entsprechenden Broker gesendet.
In der Konfigurationsdatei kann der Nutzer bestimmte Schwellwerte definieren, bei denen die Daten an unterschiedliche Broker gesendet werden. Somit können auf dem einen Feed zunächst alle Daten gesammelt werden, auf einem anderen wiederum, nur Daten die einen wirklich interessieren.

4.2.3 3. Komponente: Manuelles abrufen und versenden der veröffentlichten Daten

Möchte der Nutzer seine Daten zunächst nur lokal verwalten, hat er dennoch die Möglichkeit mithilfe der beiden folgende Skripte die Daten Nachträglich per MQTT an einen Broker zu schicken, bzw. den entsprechenden MQTT Broker zu subscriben, um seine Daten an einer anderen Stelle für die weitere Verabeitung zu sammeln.
Dafür haben wir eine Pythonklasse bereitgestellt, welche es dem Nutzer ermöglicht, entweder ein MQTT Publisher und/oder Subscriber Objekt zu erzeugen.
Im Falle des Publishers hat man die Möglichkeit, mehrere Schwellwerte in einer Konfigurationsdatei abzulegen, sodass je nach Bedarf ein oder mehrere Feeds mit Daten gefüttert werden können.

Im Prinzip laufen beide Programme so, dass innerhalb einer Endlosschleife, entweder an der Seriellen Schnittstelle oder am MQTT Broker angefragt wird, ob neue Datensätze zur verfügung stehen. Beim Publisher gibt es noch die besonderheit, dass dieser die Daten erneut auf ihre Plausibilität prüft, wofür unter anderen ein Zeitstempel hilfreich ist, um zu erkennen, dass es sich um einen neuen Datensatz handelt.
Ist dies der Fall, werden die Daten an den Broker gesendet.

Der Subscriber gibt neue Datensätze in der Konsole/Serielle Schnittstelle aus, sodass diese ggf. weiter verarbeitet werden können.

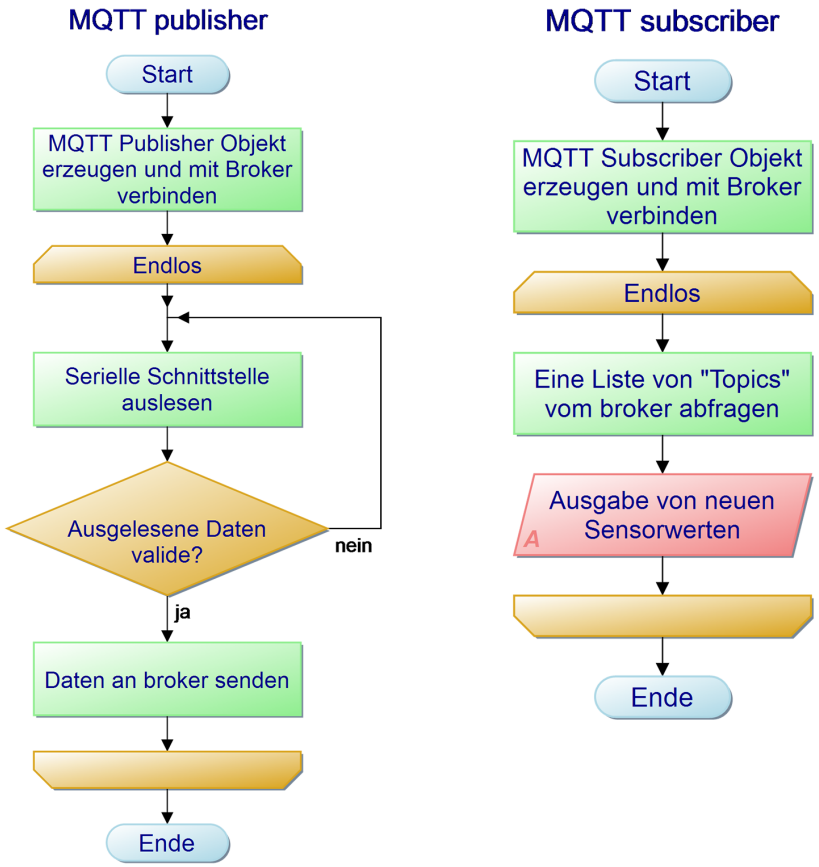

Abbildung 4.8: Programm Ablauf: Komponente 3

4.3 Visualisierung der Sensordaten

Für die Visualisierung unserer Sensordaten haben wir auf *adafruit.io* einige Feeds erstellt welche zum Einen die Rohdaten, also alle Sensordaten, zum Anderen nur mit Daten, welche einen bestimmten Schwellwert überschritten haben, gefüttert werden. Die Feeds sind frei zugänglich und können von jedem gefüttert und betrachtet werden. Zur Visualisierung haben wir ein Dashboard erstellt, welches alle Feed in einem Liniendiagramm darstellt.

4.4 Berechnung der Laufzeit im Batteriebetrieb

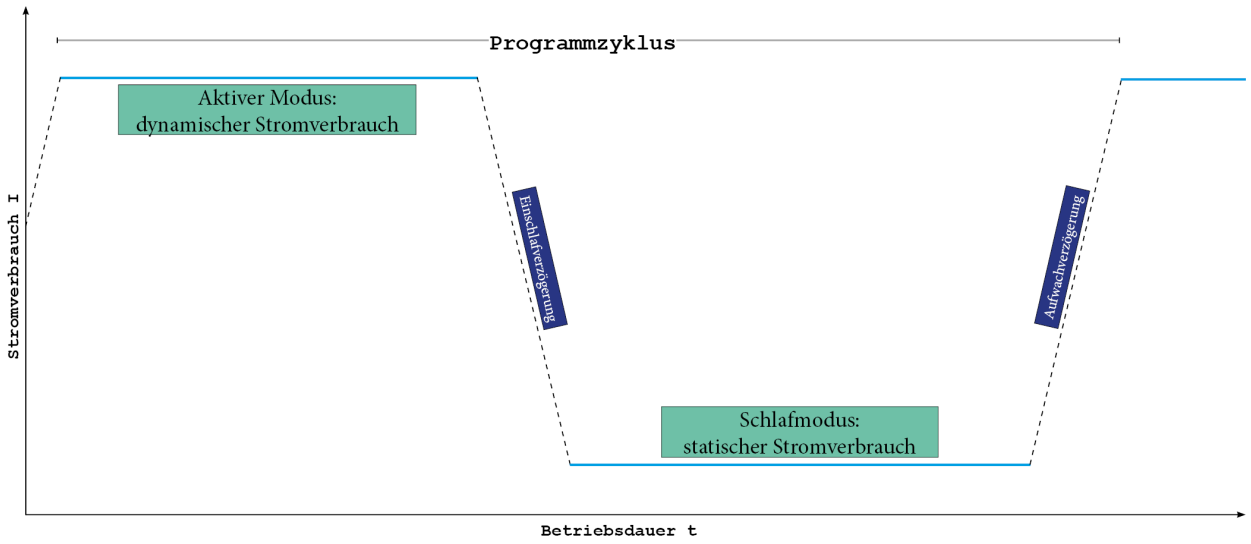

Abbildung 4.9: Stromverbrauch während eines Programmzyklus

Der durchschnittliche Stromverbrauch je Programmzyklus ergibt sich somit aus dem Tastverhältnis zwischen aktivem und passivem Modus (duty cycle).

$$I_{avg} = \frac{I_{aktiv} \cdot t_{aktiv} + I_{passiv} \cdot t_{passiv}}{t_{aktiv} + t_{passiv}}$$

Anzumerken ist hier jedoch noch, dass der Microcontroller eine gewisse Zeit und einen gewissen Strom benötigt, um den Modus zu wechseln. Diese Kennzahlen sollten für ein präzises Modell aus dem Datenblatt entnommen werden, was wir für unser Projekt erstmal vernachlässigt haben.

Berechnung der durchschnittlichen Laufzeit im Bettriebetrieb

Die Laufzeit im Betteriebetrieb lässt sich über die Kapazität der Batterie(n) und des durchschnittlichen Stromverbrauchs ermitteln

$$t_{Laufzeit} = \frac{C_{Batterie}}{I_{avg}}$$

Nachfolgend wird illustriert, wie sich die Batterielaufzeit mit der zunahme der Programmzyklusdauer erhöht. Die farbliche Markierung gibt an, ob der Sensor die gesammt Zeit angeschalten bleibt, oder über einen Schalter bzw. Transistor nur im Moment der Ermittlung der Messwerte eingeschaltet wird.

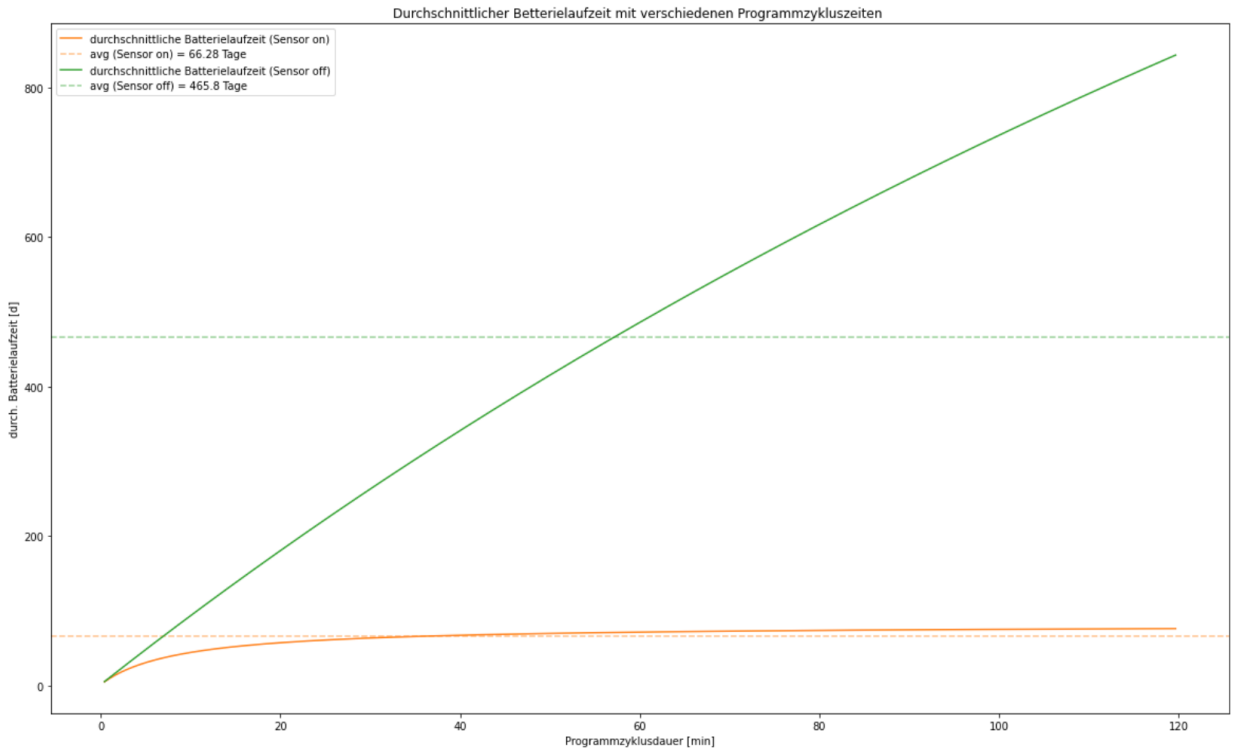

Abbildung 4.10: Potentiell mögliche Batterielaufzeiten in Abhängigkeit zur Programmzykluszeit

Zu sehen ist, dass die Batterielaufzeit in beiden Fällen mit der Zunahme der Programmzyklusdauert ansteigt, ab einer bestimmten Dauer, aber nach und nach weniger zunimmt.

Eine ausführlichere Simulation findet man im beiliegenden Jupyter Notebook.[Bus22]

Kapitel 5: Fazit

5.1 Projektauswertung

Sidney

Insgesammt ist es ein sehr interessantes Projekt gewesen, für mich war es besonder interessant etwas mehr darüber zu erfahren, dass es scheinbar doch eine Menge Initiativen und Projekte gibt, die ihren Fokus auf das Commoning bzw. auf das bereitstellen von freien Inhalten legt.
Für mich persönlich ging es in der gesammten Veranstaltung zu sehr um ein Produkt, als um eine Vision, aber das ist ja bekanntlich geschmackssache.

Ilja

5.2 Probleme und Herausforderungen

Sidney

Neben kleineren Problemen wie z.B. einem falschen USB-Kabel, war es wie so häufig eher das Problem, den Fokus auf die wichtigen Dinge zu legen. Dabei hat der Projektmanagement-Anteil aberdefinitiv geholfen.

Ilja

5.3 Ausblick

Erweiterung von LoRa-LoRa zu LoRaWAN

In unserem Projekt haben wir bisher leider kein richtiges Sensornetzwerk aufgebaut, sondern zwei Microcontroller über eine P2P LoRa Verbindung Kommunizieren lassen. Dies hat zum einen den Grund, dass wir selber kein LoRa Gateway aufbauen wollten, zum anderen aber auch kein umliegendes Gateway in erreichbarer Nähe haben.

Benutzung von mehreren Sensorknoten an einem LoRa-Gateway

Wenn wir es hinbekommen sollten ein eigenes Gateway bereitzustellen, welches auch unsere Publisher-Software implementiert hat, würden wir gerne mehrere Sensorknoten mit diesem Gateway verbinden, um so möglicherweise Unterschiede in den Messdaten zwischen den Microcontrollern zu entdecken.

Erweiterung der Software zur Einbindung von mehreren Sensorkomponenten

Derzeit haben bieten wir nur die Möglichkeit einen oder mehrere DHT Sensoren einzubinden. Wir würden gerne eine universelle Schnittstelle bereitstellen, die es ermöglicht eine Vielzahl von Sensoren anzubinden und auszuwerten.

Batteriebetrieb in Hardware umsetzen

Den Batteriebetrieb haben wir bisher nur Simuliert. Da wir keine geeigneten Konstruktion bauen konnten, die einen Batteriehalter ordnungsgemäß halten könnte. Zu einem Serienreifen Produkt gehört somit auch ein Gehäuse mit Battriehalter.

Alternative Dashboard Möglichkeiten

Wir würden gerne weitere Dashboardmöglichkeiten evalueiren, da die kostenlose Version der adafruit.io Dashboards doch recht limiert ist. Ggf. ist es auch Sinnvoll eine eigene Dashboard Plattfrom bereitzustellen, da wir weiterhin den Ansatz der freien Inhalte verfolgen wollen.

5.4 Abschlusswort

Peace

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[ada20] adafruit.io. *Adafruit Dashboard example*. https://cdn-blog.adafruit.com/uploads/2020/11/2020-11-09-dark-dashboard.png. Letzter Zugriff: 25.02.22. 2020.

[AG22] Paessler AG. *IT Explained - MQTT*. https://www.paessler.com/de/it-explained/mqtt. Letzter Zugriff: 24.02.22. 2022.

[BH19] David Bollier und Silke Helfrich. *Frei, fair und lebendig: die Macht der Commons*. Sozialtheorie. Bielefeld: transcript Verlag, 2019, S. 400. ISBN: 978-3-8394-4530-3. DOI: https://doi.org/10.14361/9783839445303.

[Bus22] Sidney Göhler & Ilja Buschujew. *GitHub Projekt: Schimmeliger Keller*. https://github.com/j4yj03/schimmeliger_keller. Letzter Zugriff: 25.02.22. 2022.

[Gho17] Sakshama Ghoslya. *All about LoRa and LoRaWAN - LoRa Decoding*. http://www.sghoslya.com/p/lora_9.html. Letzter Zugriff: 24.02.2022. 2017.

[Gol18] Isabell Gollmer. *Agiles arbeiten mit Scrum*. https://www.etventure.de/blog/digitallearning-2-agiles-arbeiten-mit-scrum/. Letzter Zugriff: 18.02.22. 2018.

[Hab15] F. Habermann. „Commonsbasierte Zukunft. Wie ein altes Konzept eine bessere Zukunft ermöglicht“. In: *Aus Politik und Zeitgeschichte* (2015), 35–37.

[Jur17] JurassicPork. *DHT_PyCom*. https://github.com/JurassicPork/DHT_PyCom/blob/pulses_get/dth.py. Letzter Zugriff: 25.02.22. 2017.

[kom22] elektronik kompedium. *LPWAN - Low Power Wide Area Network*. https://www.elektronik-kompendium.de/sites/kom/2207181.htm. Letzter Zugriff: 22.02.2022. 2022.

[Lie22] Robert Lie. *LoRa*. https://lora.readthedocs.io/en/latest/. Letzter Zugriff: 22.02.2022. 2022.

[MB19] Marcel Mangel und Sebastian Bicchi. *Praktische Einführung in Hardware Hacking - Sicherheitsanalyse und Penetration Testing für IoT-Geräte und Embedded Devices*. 1. Aufl. Heidelberg: MITP-Verlags GmbH & Co. KG, 2019. ISBN: 978-3-958-45818-5.

[Pl22] Pycom-ltd. *Lopy4-Datasheet*. `https://docs.pycom.io/gitbook/assets/specsheets/Pycom_002_Specsheets_LoPy4_v2.pdf`. Letzter Zugriff: 24.02.22. 2022.

[Sla20] Laurens Slats. *A Brief History of LoRa: Three Inventors Share Their Personal Story at The Things Conference*. `https://blog.semtech.com/a-brief-history-of-lora-three-inventors-share-their-personal-story-at-the-things-conference`. Letzter Zugriff: 23.02.2022. 2020.

[Sta20] Kamil Staniec. *Radio Interfaces in the Internet of Things Systems - Performance studies*. 1. Aufl. Singapore: Springer Nature, 2020. ISBN: 978-3-030-44846-2. DOI: `https://doi.org/10.1007/978-3-030-44846-2`.

[Tec12] Sunrom Technologies. *DHT11 - Humidity and Temperature Sensor*. `https://www.robocraft.ru/files/datasheet/DHT11.pdf`. Letzter Zugriff: 18.02.22. 2012.

[The22] TheThingsNetwork. *Spreading Factors*. `https://www.thethingsnetwork.org/docs/lorawan/spreading-factors/`. Letzter Zugriff: 23.02.2022. 2022.

Eigenständigkeitserklärung

Hiermit versichern wir, dass wir das vorliegende Projektabschlussbericht selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst haben.

Berlin, den 25.02.2022

Sidney Göhler und Ilja Buschujew