

SS2018 Embedded Systems

LoRa-Messnetzwerk

Sinan Ayhan, Nick Dirlein, Holger Herbersagen
Marcel Hizli, René Schießwohl

7. September 2018

Einleitung

LoRa ist eine neue innovative Technologie, die sehr energiesparend ist, sodass es perfekt für unsere kommerzielle Idee geeignet ist. Die Arbeit zeigt unser Vorgehen und den Verlauf des Projekts innerhalb des Semesters. Hierbei gehen wir sowohl auf die Konzeptionierung, Umsetzung und auf den Ausblick ein. Zudem weiten wir dies aus, indem wir erste Gedanken zur Kommerzialisierung beschreiben. In dem folgenden Bild erkennt man unseren ersten fertigen Prototypen und die einzelnen Bauteile, die verwendet wurden. Hierbei handelt es sich um die gelötete Platine, die Wägezelle, das LoRa-Modul mit der Antenne und ein LCD-Display. Detaillierte Informationen zu den einzelnen Bauteilen folgen in den nächsten Kapiteln.

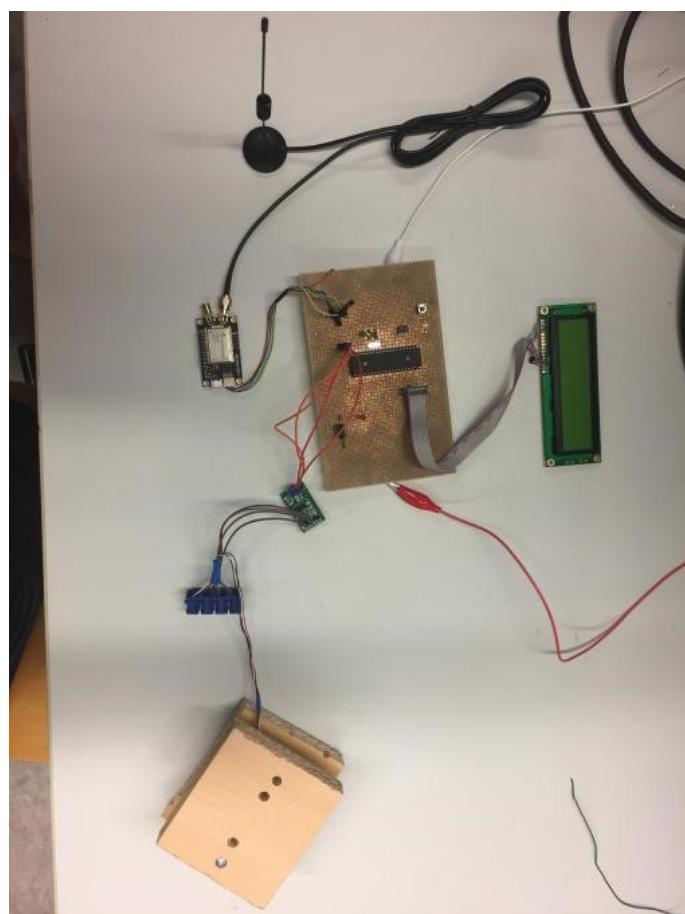


Abbildung 0.1: Überblick über die Einzelteile des Prototyps

Das Ziel dieser Dokumentation ist, dass klar und verständlich erklärt wird, wie der erste Prototyp funktioniert, welche Stärken dieser besitzt und wie man diesen verbessern könnte.

Inhaltsverzeichnis

1 Projektvorstellung	1
2 Ausgangssituation / Beweggründe für die Umsetzung	2
3 Einkaufsliste	4
4 Komponentenmodell des Wägesystems für Kaffeekannen	5
5 Überblick der Bauteile	7
5.1 LoRa-Gateway als Empfänger der Sensordaten	7
5.2 Wägezelle als Gewichtssensor	8
5.3 ADC zur Wandlung und Verstärkung von Signalen	9
5.4 LoRa-Node als Sender der Sensordaten	10
6 Gewichtsmessung mit einer Wägezelle	11
6.1 Funktionsweise der Wägezelle	11
6.2 Versuchsaufbau zur Gewichtsmessung	16
6.3 Programmlogik unseres Microchips	18
6.3.1 Vereinfachte Funktionsweise für momentane Gewichtsmessung .	18
6.3.2 Algorithmus für Rausch- und Driftunterdrückung	19
6.3.3 Algorithmus für Gewichtssendung mit dem LoRa-Modul	19
6.4 Demonstration der Gewichtsmessung	20
6.5 Alternative Use-Cases von “Straight bar”-Wägezellen	20
6.6 Alternative Bauformen von Wägezellen	21
6.6.1 Disk-Wägezellen	21
6.6.2 Wägesensoren	22
6.6.3 S-Typ-Wägezellen	22
6.6.4 Kompressions-Wägezellen	23
6.7 Alternative Gewichtssensoren	24
6.7.1 Force Sensitive Resistor (FSR)	24
6.7.2 Piezoelektrischer Sensor	24
7 Drahtlose Verbindung zwischen den Komponenten mit dem LoRaWAN-Protokoll	25
7.1 Aufsetzen eines Servers (LoRa-Gateway)	25
7.2 Erstellen einer LoRa-Applikation zum Abrufen der Daten	27
7.3 Aufsetzen eines Clients (LoRa-Node)	30
7.4 Spreading Factor	32
7.5 Security / Datensicherheit	33

7.6 Methoden und Funktionen zur Kommunikation mit dem LoRa-Modul (UART & Modul RN2483)	34
7.6.1 RN2483.c	34
7.6.2 uart.c	34
8 Webserver für die Visualisierung	36
8.1 Visualisierung der empfangenen Daten mit Angular CLI	36
8.2 NodeJS Server für die Verarbeitung der Datenpakete	37
8.3 The Things Network	37
8.4 Inaktivität der Behälter	39
8.5 Voraussetzungen und Nutzung des Node-Servers	39
9 Schaltplan und Verlöten der Platine	40
10 Fazit der prototypischen Umsetzung eines Wägesystems für den Gastronomie-Betrieb via LoRaWAN	45
11 Ausblick auf die Kommerzialisierung	46
11.1 LoRaWAN Modul Code verbessern	46
11.2 Front-End Lizenzkosten vermeiden	46
11.3 Erstellen eines wasserfesten Gehäuses	47
11.4 Kommerzialisierung und mögliches Geschäftsmodell	47

Abbildungsverzeichnis

0.1 Überblick über die Einzelteile des Prototyps	
1.1 Use-Case mit den Akteuren Kunde und Gastronomie-Personal und den Komponenten Kaffeekanne und Web-Server	1
2.1 Projektplan	3
4.1 Komponentenmodell, erstellt mit https://www.draw.io/	5
5.1 iC880A-SPI Concentrator Board (links) und Raspberry Pi Model 2B (rechts) bilden zusammen das Gateway, welches die Sensordaten der LoRa-Nodes entgegennimmt und zu einem Web-Server weiterleitet.	7
5.2 Eine 20Kg Wägezelle wird für die Gewichtsmessung zwischen zwei Spanholzplatten befestigt. Während der Gewichtsmessung liegt die untere Spanholzplatte am Boden an und auf der oberen Spanholzplatte wird das zu messene Gewicht aufgelegt.	8
5.3 Ein HX711 24-Bit ADC zur Analog-Digital-Wandlung und Verstärkung des Signals zwischen Gewichtssensor und Mikrocontroller	9
5.4 Data Link LoRa RN2483 als LoRa-Node, das zum Senden der Sensordaten über das LoRaWAN-Protokoll verwendet wird.	10
6.1 Aufbaubeschreibung einer Wägezelle	11
6.2 Funktionsweise einer Wägezelle bei Gewichtsmessung	13
6.3 Verschaltung der Dehnmessstreifen, wobei die Widerstände R1-R4 die jeweiligen Dehnmessstreifen darstellen.	14
6.4 Verschaltung der Dehnmessstreifen mit Spannungsmessung	15
6.5 20Kg Wägezelle zwischen zwei Spanholzplatten befestigt	16
6.6 Verdrahtung der Wägezelle	17
6.7 HX711 ADC-Wandler, Ansicht auf Bauteil	18
6.8 Disk-Wägezelle	21
6.9 Wägesensor	22
6.10 S-Typ-Wägezelle	22
6.11 Kompressions-Wägezelle	23
6.12 Force Sensitive Resistor	24
6.13 Piezoelektrischer Sensor	24
7.1 Übersicht eines Beispiel-LoRa-Netzwerks	25
7.2 Übersicht der registrierten Gateways im TheThingsNetwork	26

7.3 Übersicht der erstellten Applikationen im TheThingsNetwork	27
7.4 Formular zur Erstellung einer Applikation	27
7.5 Übersicht der registrierten Geräte	28
7.6 Formular zur Registrierung eines Geräts	28
7.7 Aktivierungsmethoden: links Over-the-Air Activation (OTAA), rechts Activation By Personalization (ABP)	29
7.8 Beispielcode	29
7.9 Spezifikationen für das UART-Interface des RN2483	30
7.10 RN2483 Boot Time unterteilt in die einzelnen Funktionen und der benötigten Zeit in ms	31
7.11 Tabelle mit verschiedenen Spreading Factors und die zugehörige Bitrate	33
8.1 Übersicht der Füllstände der einzelnen Kaffeemaschinen auf der Webseite	36
8.2 Füllstand-Verlauf einer Kaffeemaschine von 12:00 Uhr bis 15:00 Uhr . .	37
9.1 Schaltplan, erstellt mit Fritzing	41
9.2 Rückseite der verlötzten Platine unseres ersten Prototyps	43
9.3 Vorderseite der verlötzten Platine unseres ersten Prototyps	44

Tabellenverzeichnis

3.1 Einkaufsliste	4
-----------------------------	---

1 Projektvorstellung

Wir haben uns in der Veranstaltung Embedded Systems des Studiengangs Software-Engineering bei Prof. Dr. Jürgen Doneit und Herr Ulrich Straus im Sommersemester 2018 für das Projekt LoRa-Messnetzwerk entschieden. Die Aufgabe bestand darin, ein Messnetzwerk aufzubauen, welches Sensordaten entgegennimmt und diese über ein Long Range Wide Area Network, kurz LoRaWAN, verschickt.

Der daraus resultierende Use-Case:

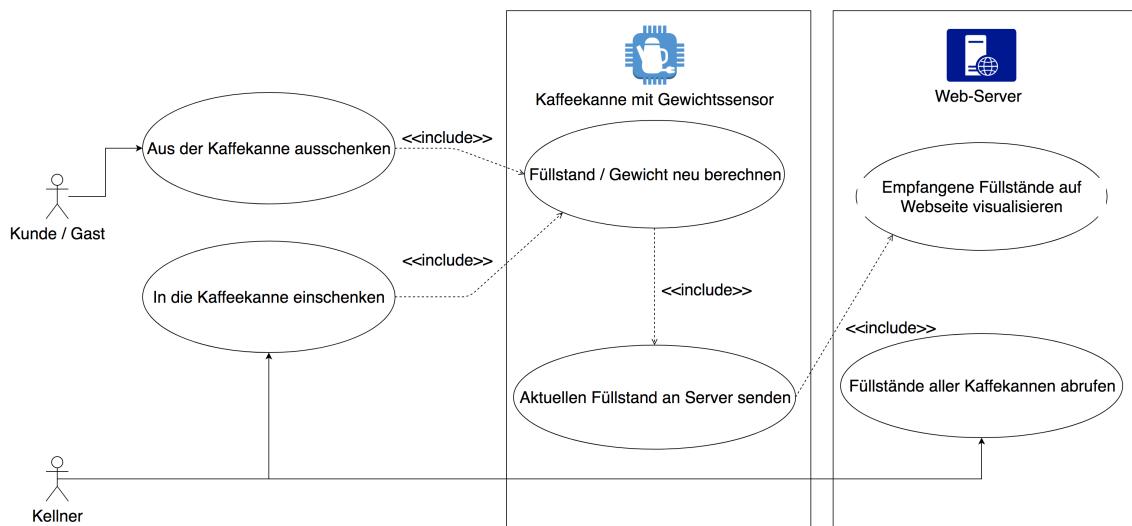


Abbildung 1.1: Use-Case mit den Akteuren Kunde und Gastronomie-Personal und den Komponenten Kaffeekanne und Web-Server

In einem Restaurant wird der Inhalt einer Kaffeekanne durch das Gewicht bestimmt. Um die Gewichtsmessung für alle Kannen durchzuführen, werden sie auf einer Plattform abgestellt, in der ein Wäge-Sensor-System integriert ist. Das Sensor-System, welches über einen Akku betrieben wird, besteht aus einem Wäge-Sensor, einem Mikrocontroller und einem Transmitter (dem LoRa-Node). Die Messdaten werden jeweils ausgelesen und an den angeschlossenen LoRa-Node übertragen. Die Gesamtheit aller LoRa-Nodes kommuniziert mit einem Empfänger, dem LoRa-Gateway. Das LoRa-Gateway, angeschlossen ans Internet, empfängt die Sensorgewichtsdaten und leitet diese auf einen Web-Server weiter. Das Server-System verarbeitet die Daten, um die Füllstände der Kaffeekannen schlussendlich auf einer Webseite zu visualisieren. Der plattformübergreifende Zugriff über diverse Endgeräte ist somit möglich.

2 Ausgangssituation / Beweggründe für die Umsetzung

In der Gastronomie üblich, prüft das Personal von Zeit zu Zeit ob in den Behältnissen der Gäste auf den Tischen oder am Buffet noch genügend Inhalt vorhanden ist. Das reicht von der Kaffeekanne auf dem Frühstückstisch, über das Butterbehältnis auf dem Buffet, bis hin zur Weinflasche am Abend im Restaurant. Natürlich gehört dies zum Service eines guten Gastrounternehmens. Auf der anderen Seite kostet es viel Servicekraft die Füllstände zur vollen Zufriedenheit des Kunden und jederzeit im Auge zu behalten.

Unser Ziel ist es bei diesem Prozess eine höhere Effizienz zu erzielen. Es soll zu jeder Zeit und von überall, ohne Sichtprüfung eine klare Aussage über einen nötigen Nachfüllprozess getroffen werden können. Durch das Wegfallen der ständigen Sichtprüfung vor Ort, hat der Kunde eine ruhigere, entspanntere und damit angenehmere Atmosphäre. Er wird nur dann auf den Wunsch des Nachfüllens angesprochen, wenn das Gefäß tatsächlich leer ist. Damit werden deutlich optimalere Geschäftsabläufe gewährleistet:

- Produkte können bedarfsgerechter angerichtet, zubereitet oder hergestellt werden
- Wartezeiten gegenüber dem Kunden lassen sich wahrnehmbar verkürzen, die Zufriedenheit steigt
- und das Personal kann sich auf andere/wichtigere Themen bezüglich des Kunden konzentrieren

Um die Kernfunktionalität im Einsatz zu testen und die Idee zu validieren, beschränken wir uns bei der Entwicklung unseres Prototyps auf die Messung der Füllstände von Kaffeekannen.

Damit das System funktioniert, werden die Kaffeekannen auf einer Plattform abgestellt und durch Wäge-Sensoren gewogen. Die gemessenen Werte und die daraus entstandenen Informationen werden visualisiert und dem Personal über eine Webseite angezeigt. Bei niedrigem Füllstand (oder zu niedriger Temperatur, weil schon zu lange in der Kanne) können sie damit schnell reagieren. Die komplette drahtlose Kommunikation basiert dabei auf LoRa-Nodes, die sehr energiesparend sind. Bei konstanter täglicher Nutzung des Systems ist bei entsprechender Akkugröße eine Laufzeit von einem Jahr möglich. Trotzdem ist zusätzliches Optimierungspotential vorhanden. Denn werden die LoRa-Nodes nach dem Senden der Zeit- und Gewichtsinformationen

in einen Sleep-Mode versetzt, zusätzlich der Sende-Zyklus verlängert und serverseitig ein besserer Abstraktionsalgorithmus implementiert, sind bei gleichem Akku Laufzeiten bis zu drei Jahren denkbar.

Am Anfang des Projektes erstellten wir Bauteildiagramme und diskutierten über Kommunikationswege zwischen den Bauteilen. Danach teilten wir Schwerpunkte beziehungsweise Bauteile auf unsere Teammitglieder auf, welche dann nochmal tiefer vom zugewiesenen Teammitglied recherchiert wurden. Die Zusammensetzung der Bauteile erfolgte zunächst um schnelle Validierungstests und Hardwareänderungen durchführen zu können. Nach erfolgreichen Tests auf unserer zuvor zusammengesetzten Entwicklungsplattform wechselten wir die Hardware auf eine selbst konzeptionierte Platine. Mit dem letzten validen Test und nach der offiziellen Präsentation dokumentierten wir unsere Ergebnisse.

Gefährdet	Aufgabenname	Start Datum	Ende Datum	Zugewiesen	% vollständig
	Projektstart	26.03.18	03.04.18		100%
	Kick-Off	26.03.18	26.03.18	Alle	100%
	Konzeptionierung	26.03.18	26.03.18	Alle	100%
	Arbeitseinteilung	26.03.18	26.03.18	Alle	100%
	Recherche	27.03.18	03.04.18	Rene	100%
	Kraftsensor und Verteilung des gewichts	27.03.18	03.04.18	Holger	100%
	LORA Protokoll	27.03.18	03.04.18	Nick	100%
	LORA Server	27.03.18	03.04.18	Sinan	100%
	Stromversorgung	27.03.18	03.04.18	Marcel	100%
	Grobe Realisierung	03.04.18	17.04.18		100%
	Bestellung	03.04.18	03.04.18	Alle	100%
	Steckboard bestücken	03.04.18	17.04.18	Nick/Holger	100%
	LORA Server	03.04.18	17.04.18	Sinan	100%
	Prototypische Implementierung	16.04.18	25.06.18		100%
	Bestellte Teile einbauen	16.04.18	03.05.18	Alle	100%
	Übertrag auf Steckplatine	07.05.18	15.06.18	René/Marcel	100%
	Schaltplan erstellen	07.05.18	25.06.18	René/Marcel	100%
	Produkt auf Platine umziehen	21.05.18	25.06.18	René/Marcel	100%
	Testen	11.05.18	02.07.18		100%
	Hardwaretest	11.05.18	21.05.18	Alle	100%
	Hardwaretest mit Platine	25.06.18	02.07.18	Alle	100%
	Softwaretest	21.05.18	02.07.18	Alle	100%
	Dokumentation	29.06.18	06.07.18		100%
	Zusammenführen	29.06.18	06.07.18	Alle	100%
	Formatieren	29.06.18	06.07.18	Alle	100%
	Rechtschreibprüfung	29.06.18	06.07.18	Alle	100%

Abbildung 2.1: Projektplan

3 Einkaufsliste

Tabelle 3.1: Einkaufsliste

Komponente	Preis
Web-Server für NodeJS VPS oder Cloud Server mieten oder einen vorhandenen, netzwerkfähigen Computer benutzen	ab 5.00 €/ Monat -
LoRa-Gateway iC880A-SPI + Antenne + Pigtail cable + Raspberry Pi 2 B	167.90 €
LoRa-Node Data Link LoRa RN2483 + Antenne	Ca. 70 €
Wägezelle + HX711	10.71 €
Mikrocontroller ATmega644PA	4.24 €

Für unseren ersten Prototyp benötigten wir das LoRa-Node Modul mit Antenne, um Daten an das Gateway (Server) verschicken zu können. Außerdem haben wir eine Wägezelle mit einem AD-Wandler (HX711) an einen Mikrocontroller (ATmega644PA) angeschlossen, um damit das Gewicht von der Wägezelle messen und übertragen zu können.

Als Vorgriff vor dem Resultat, sei hier an dieser Stelle zu erwähnen, dass während der Validierungstests der Mikrocontroller oder das Steuergerät für die Wägezelle (HX711) als verzichtbar erkannt wurde. Das würde den Preis etwas verringern. Der Grund ist, dass mögliche Sensor-Informationen sich direkt über die Ein- und Ausgangskanäle des LoRa-Moduls (RN2483) ohne Zwischenschritt auf dem Mikrocontroller übertragen lassen. Im weiteren Dokumentationsverlauf wird der Mikrocontroller in Kombination mit dem Steuergerät (HX711) jedoch weiterhin eingesetzt.

4 Komponentenmodell des Wägesystems für Kaffeekannen

Auf dem nachfolgenden Komponentenmodell sind die angedachten Kommunikationswege der einzelnen Module aufgebaut.

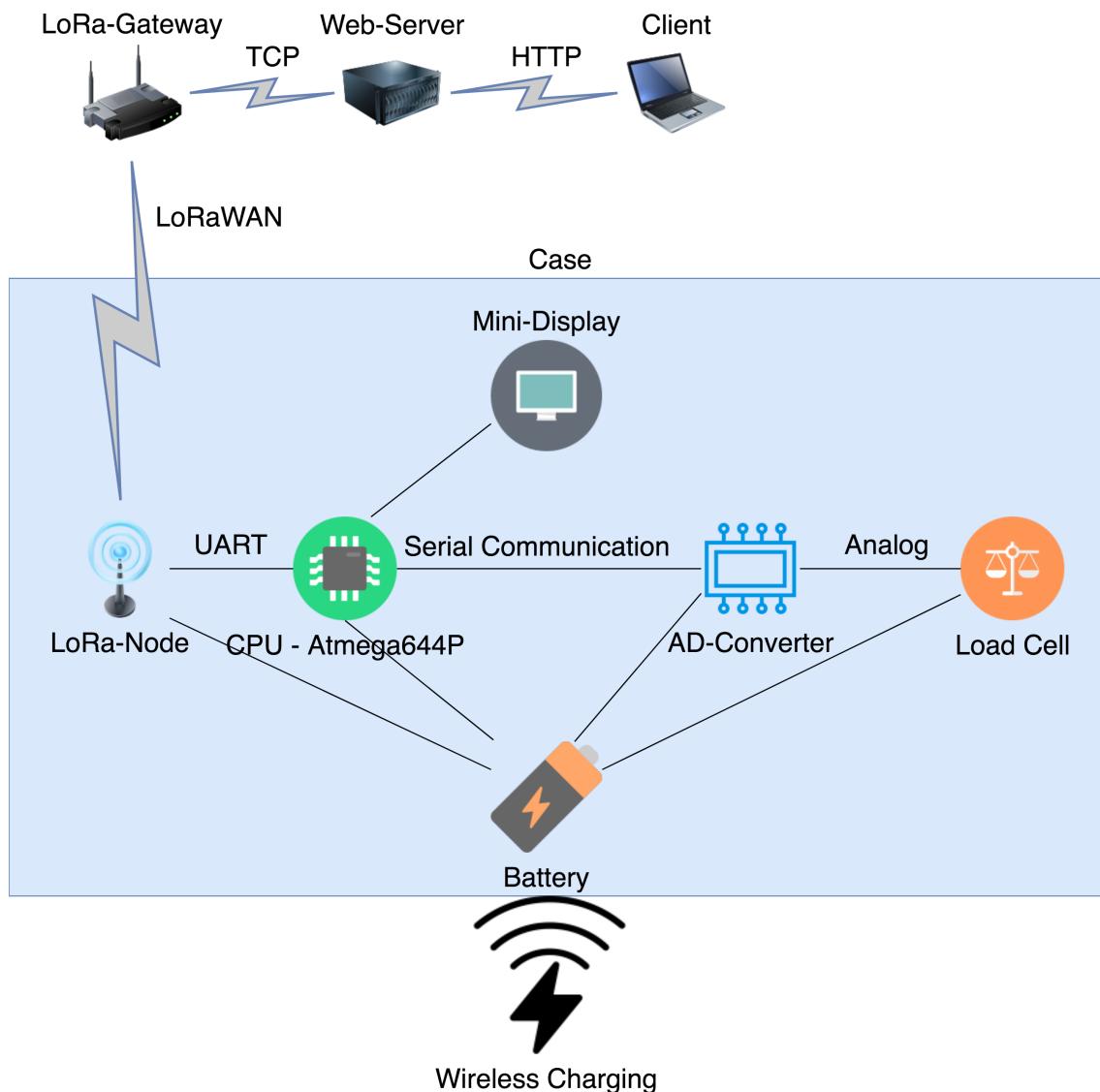


Abbildung 4.1: Komponentenmodell, erstellt mit <https://www.draw.io/>

Die Herausforderung bestand dabei, die verteilten Informationen bis hin zum Client zu liefern. Da in der Gruppe die Arbeitspakete gleichzeitig abgearbeitet wurden, waren wir gezwungen für die einzelnen Testphasen Dummydaten zu erstellen, die das jeweils andere bzw. zuliefernde Modul ersetzt. Das Gehäuse (hier 'Case') beinhaltet alle Komponenten für den Prozess des Wiegens einer Kaffee-Kanne. Der darin enthaltene LoRa-Knoten (hier: LoRa-Node) übermittelt mit Hilfe des LoRaWAN-Protokolls die jeweiligen Sensordaten mit Adresse und Zeitstempel an das LoRa-Gateway, welches alle Knoten in Empfang nimmt.

5 Überblick der Bauteile

5.1 LoRa-Gateway als Empfänger der Sensordaten

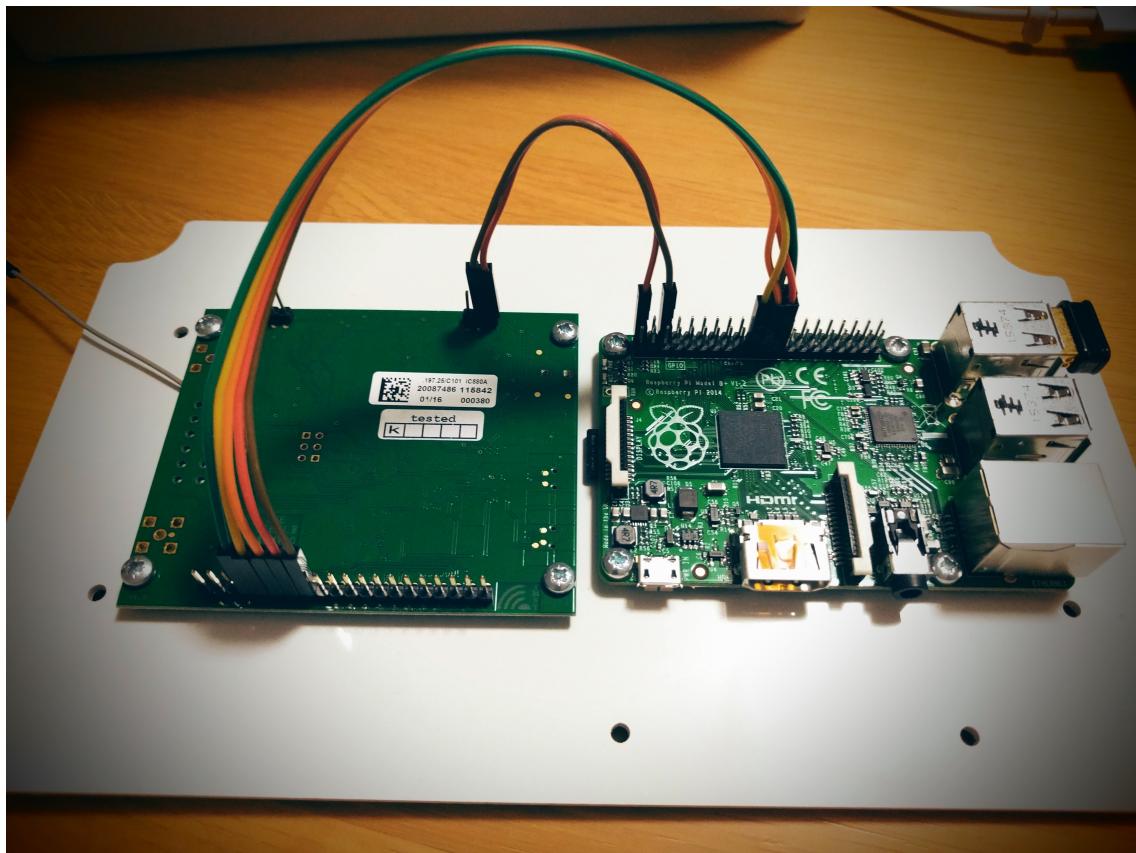


Abbildung 5.1: iC880A-SPI Concentrator Board (links) und Raspberry Pi Model 2B (rechts) bilden zusammen das Gateway, welches die Sensordaten der LoRa-Nodes entgegennimmt und zu einem Web-Server weiterleitet.

Quelle: <https://raw.githubusercontent.com/ttn-zh/ic880a-gateway/spi/images/mounted-boards.jpg>

5.2 Wägezelle als Gewichtssensor



Abbildung 5.2: Eine 20Kg Wägezelle wird für die Gewichtsmessung zwischen zwei Spanholzplatten befestigt. Während der Gewichtsmessung liegt die untere Spanholzplatte am Boden an und auf der oberen Spanholzplatte wird das zu messene Gewicht aufgelegt.

5.3 ADC zur Wandlung und Verstärkung von Signalen

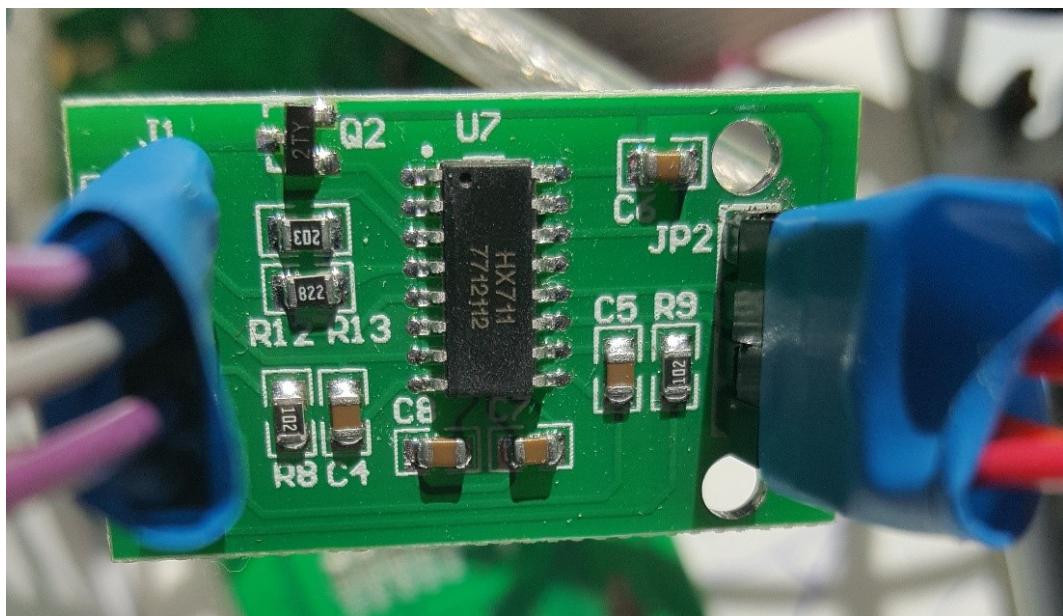


Abbildung 5.3: Ein HX711 24-Bit ADC zur Analog-Digital-Wandlung und Verstärkung des Signals zwischen Gewichtssensor und Mikrocontroller

5.4 LoRa-Node als Sender der Sensordaten

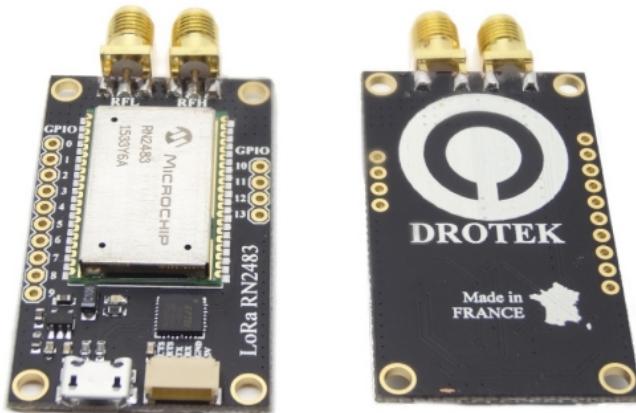


Abbildung 5.4: Data Link LoRa RN2483 als LoRa-Node, das zum Senden der Sensordaten über das LoRaWAN-Protokoll verwendet wird.

Quelle:

https://drotek.com/shop/2643-thickbox_default/data-link-lora-rn2483.jpg

6 Gewichtsmessung mit einer Wägezelle

Um die Kaffeekanne und deren Füllstand zu messen, entschied sich das Team nach Sondierung mehrerer Alternativen (Drucksensoren, visuelle Sensoren) für die Gewichtsmessung. Dazu soll die Technik einer Wägezelle zum Einsatz kommen. In Personen- oder Küchenwagen sind sie in millionenfacher Ausführung verbaut, was beweist, dass sie valide genutzt werden können. Weitere Alternativen, nicht nur zum Einsatz, sondern auch in der Bauform sind in Kapitel 6.5 ff. zu finden.

6.1 Funktionsweise der Wägezelle

Wägezellen sind Sensoren, die das Gewicht über Verformung ihres Materials mit Hilfe von Dehnungsmessstreifen messen können. Es sind vier Dehnungsmessstreifen, wie im Bild unten zu sehen, an der Wägezelle angebracht.

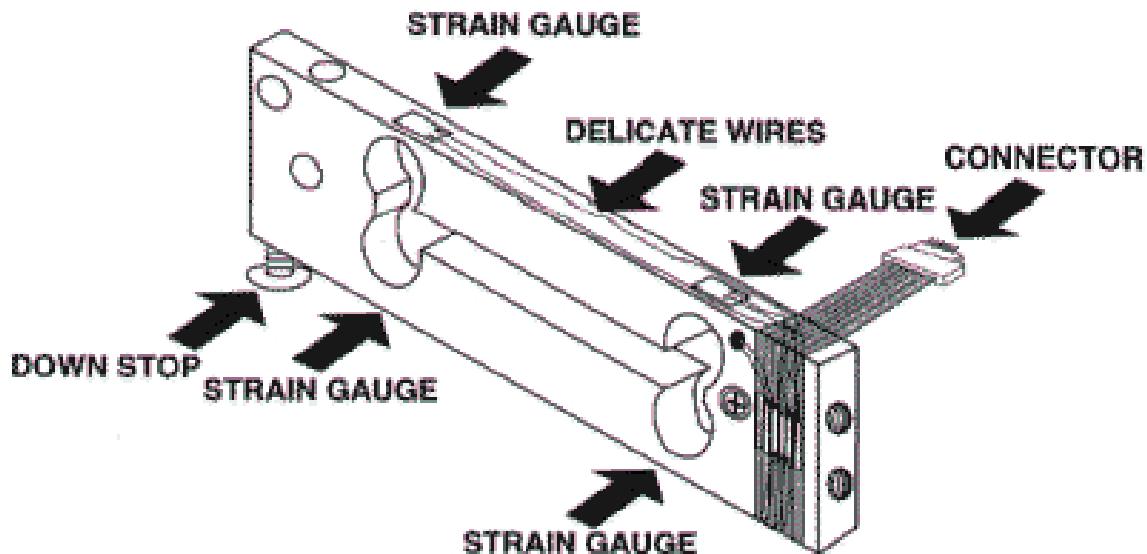


Abbildung 6.1: Aufbaubeschreibung einer Wägezelle

Quelle: <https://learn.sparkfun.com/tutorials/getting-started-with-load-cells>

Bei Auflage von Gewicht bzw. Verformung der Wägezelle messen zwei Dehnungsmessstreifen die Kompression und die anderen zwei die Spannung. Bei Kompression des Dehnungsmessstreifens werden die Leiter dicker (höherer Leiterquerschnitt) und kürzer

(geringere Leiterlänge). Laut der Formel für den Leiterwiderstand

$$R = p \cdot \frac{I}{A} \quad (6.1)$$

$$\text{Leiterwiderstand} = \text{spezifischer Widerstand} \cdot \frac{\text{Leiterlänge}}{\text{Leiterquerschnitt}} \quad (6.2)$$

verringert sich der Widerstand des Dehnungsmessstreifens. Bei Spannung des Dehnungsmessstreifens werden die Leiter dünner und länger, somit erhöht sich auch der Widerstand des Dehnungsmessstreifens.

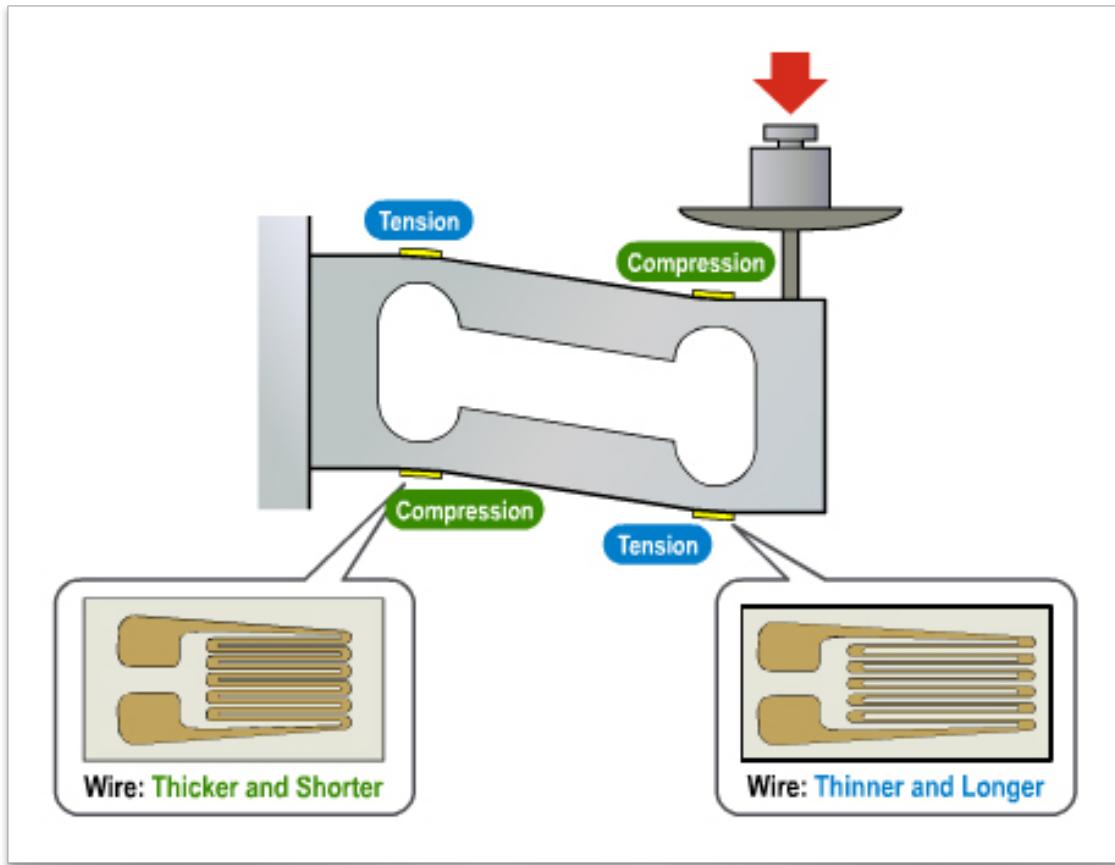


Abbildung 6.2: Funktionsweise einer Wägezelle bei Gewichtsmessung

Quelle: <https://learn.sparkfun.com/tutorials/getting-started-with-load-cells>

Um die Widerstandsveränderungen der Dehnungsmessstreifen messen zu können, sind die Dehnungsmessstreifen innerhalb der Wägezelle in einer Wheatstone-Brücken-Formation geschalten. Die Wägezelle wird mit vier Dehnungsmessstreifen ausgestattet. Diese sind in ihrer Form, in der Länge und der geometrischen Anordnung genau gleich. [vgl. Al-Mutlaq, Sarah: Getting Started with Load Cells, URL: <https://learn.sparkfun.com/tutorials/getting-started-with-load-cells>, (Stand 06.07.2018)]

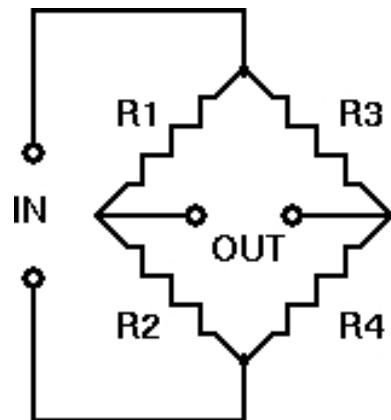


Abbildung 6.3: Verschaltung der Dehnmessstreifen, wobei die Widerstände R1-R4 die jeweiligen Dehnmessstreifen darstellen.

Quelle: <https://learn.sparkfun.com/tutorials/getting-started-with-load-cells>

Wenn für die Verschaltung der Dehnungsmessstreifen gilt:

$$\frac{R1}{R2} = \frac{R3}{R4} \quad (6.3)$$

dann misst man 0V an Out (siehe Abb. 6.3). Falls sich ein Widerstand verändern sollte, zum Beispiel durch Verformung eines Dehnungsmessstreifens (d.h. stellt man eine Kaffeekanne auf die Wägezelle), lässt sich eine Spannung an Out laut dieser Formel messen:

$$V_{out} = \left(\frac{R3}{(R3 + R4)} - \frac{R2}{(R1 + R2)} \right) \cdot V_{in} \quad (6.4)$$

In Abbildung 6.4 sind die verschalteten Dehnungsmessstreifen unter Wirkungseinfluss zu sehen und damit die Auslenkung der jeweiligen Streifen.

Full-bridge strain gauge circuit

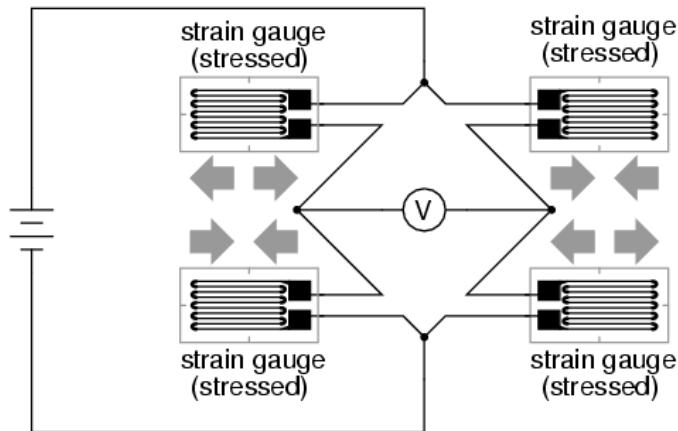


Abbildung 6.4: Verschaltung der Dehnmessstreifen mit Spannungsmessung

Quelle: <https://learn.sparkfun.com/tutorials/getting-started-with-load-cells>

Die nun an *Out* (siehe Abb. 6.3) oder an *V* (siehe Abb. 6.4) gemessene analoge Spannung ist sehr klein. Für eine Abbildung der analogen Spannung auf digitale Werte ist ein Analog-Digital-Wandler (AD-Wandler) nötig, welcher nicht nur die Signale verstärkt, sondern auch die digitalen Werte repräsentiert. Der zu unserer Wägezelle angebotene integrierte Schaltkreis (englisch *integrated circuit*, kurz IC, siehe Wikipedia https://de.wikipedia.org/wiki/Integrierter_Schaltkreis) war der HX711, der die gemessenen Spannungen bis zu 128fach verstärkt und mit einer Präzession von 24 Bit [0 bis 16777217] wandelt.

6.2 Versuchsaufbau zur Gewichtsmessung

Die Wägezelle wurde an zwei gegenüberliegenden Punkten an zwei Holzplatten geschraubt, damit sich das Gewicht auf alle Dehnungsmesstreifen auswirkt.



Abbildung 6.5: 20Kg Wägezelle zwischen zwei Spanholzplatten befestigt

Die Wägezelle wurde wie folgt an das HX711-Modul angeschlossen:

- Rot an E+ (Excitation+/Vin+)
- Schwarz an E- (Excitation-/Vin-)
- Weiß an A+ (Output+/Vout+)
- Grün an A- (Output-/Vout-)

Veranschaulichung der Verdrahtung (Achtung! Gelbe Verbindung ist bei uns weiß):

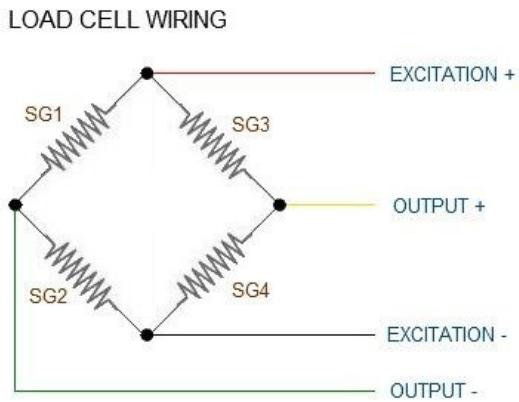


Abbildung 6.6: Verdrahtung der Wägezelle

Quelle: [https://learn.sparkfun.com/tutorials/
load-cell-amplifier-hx711-breakout-hookup-guide](https://learn.sparkfun.com/tutorials/load-cell-amplifier-hx711-breakout-hookup-guide)

Die A-Anschlüsse und B-Anschlüsse des HX711 (siehe Abbildung 6.7 HX711 linke Kontaktseite) sind für verschiedene Verstärkungskanäle. Der A-Kanal hat einen programmierbaren Verstärkungsfaktor von 128 oder 64 und der B-Kanal hat einen festen Verstärkungsfaktor von 32.

Die rechte Seite des HX711 wird wie folgt angeschlossen:

- GND an Ground
- VCC an Versorgungsspannung in unserem Fall 5V
- SCK an PD5 unseres Microcontrollers/ATmega644PA
- DT (Data) an PD6 unseres Microcontrollers/ATmega644PA

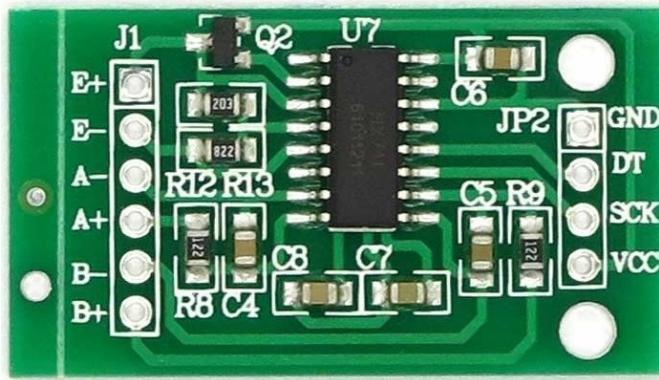


Abbildung 6.7: HX711 ADC-Wandler, Ansicht auf Bauteil

Quelle: <https://tinyurl.com/y8grwwdb>

6.3 Programmlogik unseres Microchips

Nach dem Aufbau unseres Systems geht es nun an die Gewichtsmessung. Das Programm zur Gewichtsmessung mittels dem HX711 basiert auf getsiddd's HX711 AVR Bibliothek (<https://github.com/getsiddd/HX711>).

Die wichtigsten Erweiterungen von uns sind:

- die Möglichkeit, negative Gewichtsveränderungen wahrnehmen zu können, siehe Funktion weightStuff() in RN2483.c, Zeile 246-254
- auf Tastendruck eine Tar-Funktion auszuführen, siehe Funktion weightStuff() in RN2483.c, Zeile 312-316
- Gewichts-Sende-Algorithmus in Verbindung mit dem LoRa-Modul, siehe Funktion ISR(TIMER0_COMPA_vect) in RN2483.c, Zeile 146-194

6.3.1 Vereinfachte Funktionsweise für momentane Gewichtsmessung

Nachdem die Bibliothek für den Baustein HX711 angepasst ist, also das Gewicht gemessen und versendet werden kann, geht es darum einen visualisierbaren Wert zu

erhalten. Jedoch sind nicht alle Wägezellen perfekt gleich und ohne Rücksicht auf diese Imperfektionen würde jede Wägezelle einen anderen Messwert ergeben. Um dieses Problem zu lösen, ist eine Kalibrationsvariable im Code implementiert. In unserem Fall, ist die Kalibrationsvariable ein fest codierter Parameter, der durch "Trial und Error" ermittelt wird, indem wir ein bekanntes Gewicht auf die Wägezelle legen und den Kalibrationswert in Richtung des gesuchten Gewichtes anpassen. Mit dem Ergebnis aus der Kalibrierung ist die Gewichtsmessung möglich.

Dazu ein kurzer Programmablauf der Funktionsweise:

- das Gewicht wird zunächst gelesen (Bsp.: 8563)
- das gefälschte Gewicht wird davon abgezogen ($8563 - 6546 = 2017$)
- anschließend wird durch das Ergebnis der Kalibrierung geteilt: ($2017 / 2000 = 1,0085$)
- um am Ende einen lesbaren Kilogrammwert zu bekommen: 1,0085 Kg

6.3.2 Algorithmus für Rausch- und Driftunterdrückung

Das Differenzgewicht von der letzten und der momentanen Gewichtsmessung wird mit einer vorher gewählten Veränderungsrate verglichen. Wenn das Differenzgewicht höher als die gewählte Veränderungsrate ist, wird der momentan gemessene Wert verwendet und angezeigt, im anderen Fall wird der Wert verworfen, weil er vermutlich durch Rauschen oder Drift verursacht wurde.

Der Veränderungswert gibt auch die kleinstmögliche messbare Veränderung an. Am Beispiel unseres Projektes könnte bei einem Veränderungswert von ca. 5g, jemand mit einem Strohhalm Flüssigkeit aus dem Behältnis saugen, ohne dass das Programm die Gewichtsveränderung wahrnimmt.

6.3.3 Algorithmus für Gewichtssendung mit dem LoRa-Modul

Die gemessenen Gewichtsdaten müssen nun an die Sammelstelle (den Gateway) gesendet werden. Der Sendevorgang basiert auf dem LoRa-Protokoll, welches vom LoRa-Modul in Empfang und in regelmäßigen Abständen zum LoRa-Gateway gesendet wird (vgl. 4.1 Komponentenmodell). Mit Hilfe eines Interrupts, wird jede Sekunde eine Gewichtsmessung durchgeführt. Beim Ablauf des Sendeintervalls (in unserem Projekt momentan eingestellt auf fünfzehn Sekunden), wird in der Menge der Gewichtsmessungen (in unserem Fall fünfzehn Messungen, jede Sekunde eine), nach den letzten fünf Gewichtsmessungen gesucht, die den gleichen Wert haben. Wenn die Messungen diese Bedingung nicht erfüllen, wird kein Wert gesendet.

6.4 Demonstration der Gewichtsmessung

Hier eine Demo zur Gewichtsmessung, die wir auch in der Zwischenpräsentation gezeigt haben:

<https://streamable.com/ycxf3>

Alternativ: <https://www.youtube.com/watch?v=2tk0ydRsXgg>

6.5 Alternative Use-Cases von “Straight bar”-Wägezellen

Die Art, der von uns verwendeten Wägezellen, wird unter anderem in diesen Bereichen verwendet:

- Küchenwaagen
- Industrielle Gewichtsmessung, die an einem Punkt erfolgt
- Personenwaagen
- (geeichte) Handelswaagen für Obst, Fleisch, Käse, Gemüse

6.6 Alternative Bauformen von Wägezellen

Es gibt überhaupt viele Möglichkeiten zu wiegen. Neben den Wägezellen sind auch andere technische Lösungen möglich, auf die wir in Kapitel 6.7 kurz eingehen.

Wägezellen gibt es in verschiedensten Bauformen. Die folgende Aufzählung zeigt die meisten davon. Einige haben wir auch auf unsere Anforderungen hin untersucht, um für unser Projekt und entsprechende Folgeprojekte die richtigen auswählen zu können.

6.6.1 Disk-Wägezellen

Disk-Wägezellen haben eine runde Form und sind kompakter gebaut.

Mögliche Einsatzgebiete sind:

- Industrie bei sehr großen Gewichtsmessungen 1 - 50 Tonnen, verteilte Aufnahme der Gewichtseinwirkung
- Umgebungen mit wenig Platz, da sehr platzsparend einsetzbar, jedoch mit sehr hoher Ungenauigkeit. In diesem Fall liegt die Nicht-Wiederholpräzision bei über 0,5%



Abbildung 6.8: Disk-Wägezelle

Quelle: http://www.forsentek.com/prodetail_13.html

6.6.2 Wägesensoren

Wägesensoren besitzen nur einen Dehnmessstreifen anstatt herkömmliche, die vier besitzen. Man kann vier Wägesensoren in einer Wheatstone-Bridge-Formation zusammen schalten und wie eine Straight-Bar-Wägezelle betreiben. Dabei kann man die einzelnen Wägesensoren auf einer größeren Fläche betreiben und sie als Personenwaage, Fahrzeugwaage oder allgemein als Waage für große Objekte verwenden.



Abbildung 6.9: Wägesensor

Quelle:

<https://www.botshop.co.za/product/load-cell-sensor-resistance-strain-50kg/>

6.6.3 S-Typ-Wägezellen

Diese Wägezellen lassen sich, dank ihrer S-Form, in Spannungs- und Kompressionsbetrieb betreiben.



Abbildung 6.10: S-Typ-Wägezelle

Quelle: <https://www.coventryscales.co.uk/scale-type/load-cells/tension-s-type-load-cells/s-type-load-cell/>

6.6.4 Kompressions-Wägezellen

Diese Art von Wägezellen sind nur in Kompressionsumgebungen einsetzbar und gehören auch zur Familie der Disk-Wägezellen. Ihr Einsatzgebiet ist in hocheffizienten Industrieanwendungen, mit meist deutlich über 1Kg als Minimalgewicht.



Abbildung 6.11: Kompressions-Wägezelle

Quelle: [http://www.eilersen.com/compression-load-cell/product/
atex-compression-load-cell-dla/](http://www.eilersen.com/compression-load-cell/product/atex-compression-load-cell-dla/)

6.7 Alternative Gewichtssensoren

6.7.1 Force Sensitive Resistor (FSR)

Bei Kraftzunahme beziehungsweise Kompression des Sensors verringert sich der Abstand zwischen den Sensorfolien, die voneinander mit einer speziellen Tinte getrennt sind, und somit auch ihr Widerstand. Diese Sensoren sind sehr platzsparend, aber auch ungenauer als Wägezellen, weshalb sie nicht sehr gut für die Gewichtsmessung geeignet sind.



Abbildung 6.12: Force Sensitive Resistor

Quelle: <https://solarbotics.com/product/50803/>

6.7.2 Piezoelektrischer Sensor

Bei Krafteinwirkung produzieren diese Sensoren eine elektrische Ladung, die gemessen werden kann. Sie sind eher für dynamische Anwendungen geeignet, da Signale nur bei Kraftänderungen gemessen werden können, beziehungsweise die Ladung sehr schnell wieder gegen Null geht. Es gibt spezielle Varianten, die ihre Ladung bis zu einer Minute halten können, diese weisen jedoch einen hohen Drift auf.

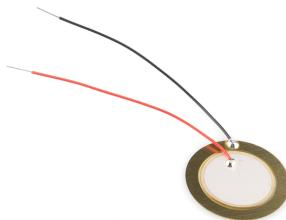


Abbildung 6.13: Piezoelektrischer Sensor

Quelle: <https://www.sparkfun.com/products/10293>

7 Drahtlose Verbindung zwischen den Komponenten mit dem LoRaWAN-Protokoll

7.1 Aufsetzen eines Servers (LoRa-Gateway)

Unser LoRa-Gateway hat im Prinzip die Funktion eines WLAN-Routers. Alle Clients (LoRa-Nodes) melden sich an dem Gateway an und kommunizieren mit dem Gateway. Wie der Begriff „Gateway“ schon andeutet, ist dies ein Tor, das zum Internet führt (siehe folgende Abbildung 7.1).

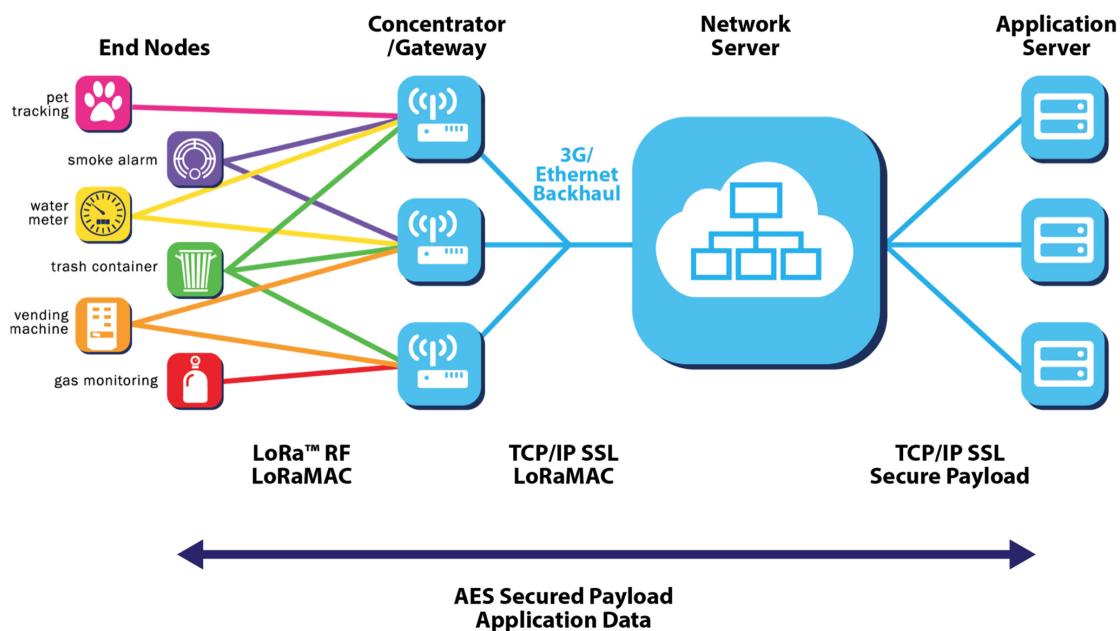


Abbildung 7.1: Übersicht eines Beispiel-LoRa-Netzwerks

Quelle: <http://www.mstanley.co.uk/blog/wp-content/uploads/2015/11/Enabling-world-wide-mobility-for-the-IoT-image-2.jpg>

Realisiert wird unser Gateway durch ein „iC880A-SPI Concentrator Board“, das mit Hilfe einer Antenne mit den LoRa-Nodes Daten austauscht. Das Gateway, also das Concentrator Board, ist über ein Serial Peripheral Interface (SPI) an einen Raspberry Pi angeschlossen. Dieser Einplatinen-Computer versendet nun die Daten der LoRa-Nodes, empfangen über das Gateway, ins Internet.

Damit nun die gesendeten Daten nicht nur lose in der großen weiten Welt des Internets ziellos umher irren, benötigt das Gateway eine Adresse, über die Datenpakete identifiziert werden können. Doch vor der Adress-Konfiguration der Hardware, sollten die Komponenten beschafft und lokal richtig installiert werden damit sie miteinander funktionieren.

Auf dem Wiki der GitHub-Seite von The Things Network Zurich (<https://github.com/ttn-zh/ic880a-gateway/wiki>) kann man nachlesen welche Bauteile man benötigt, wie man die Pins verbinden soll, das Betriebssystem aufsetzt und anschließend das Gateway im TheThingsNetwork registriert.

Nachdem man die Anweisungen zum Konfigurieren befolgt hat, sollte es auf der Webseite für Gateways von TheThingsNetwork zu sehen sein.

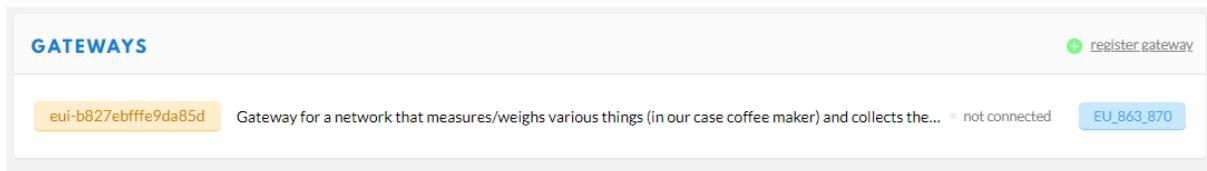


Abbildung 7.2: Übersicht der registrierten Gateways im TheThingsNetwork

Quelle: <https://console.thethingsnetwork.org/gateways>

7.2 Erstellen einer LoRa-Applikation zum Abrufen der Daten

Damit wir jetzt auch mit den verschlüsselten Daten, die wir empfangen, etwas anfangen können, müssen wir erstmal eine Applikation in der Konsole von TheThingsNetwork erstellen.

The screenshot shows the 'APPLICATIONS' section of the TheThingsNetwork console. It lists two applications: 'lora_measure_network' (Network that measures/weights various things (in our case coffee maker) and collects their status (f...)) and 'ttn-handler-eu' (70 B3 D5 7E D0 00 C5 AE). A red box highlights the 'add application' button in the top right corner.

Abbildung 7.3: Übersicht der erstellten Applikationen im TheThingsNetwork

Quelle: <https://console.thethingsnetwork.org/applications>

The screenshot shows the 'ADD APPLICATION' form. It includes fields for 'Application ID' (test2414414), 'Description' (Just a test application.), 'Application EUI' (EUI issued by The Things Network), and 'Handler registration' (ttn-handler-eu). The 'Add application' button at the bottom right is highlighted with a red box.

Abbildung 7.4: Formular zur Erstellung einer Applikation

Quelle: <https://console.thethingsnetwork.org/applications>

Application ID: Muss einzigartig und eindeutig sein.

Description: Eine kurze, aber aussagekräftige Beschreibung der Applikation.

Handler registration: Wurde passend zum Standort gewählt.

Nachdem man nun eine Applikation erstellt hat, fügt man nun auch ein Gerät zu dieser Applikation hinzu.

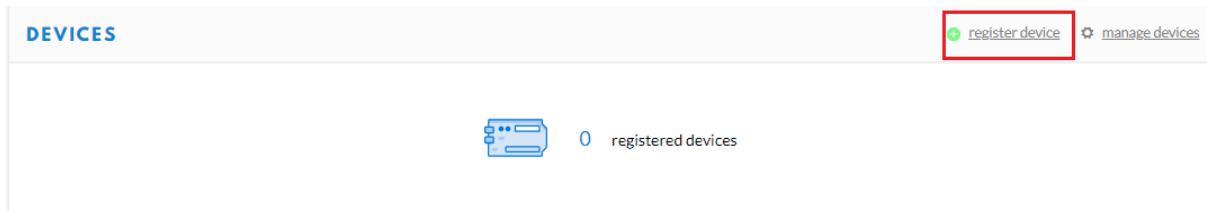


Abbildung 7.5: Übersicht der registrierten Geräte

A screenshot of a 'REGISTER DEVICE' form. The fields are as follows: 'Device ID' is set to 'test1'; 'Device EUI' is redacted; 'App Key' is redacted; and 'App EUI' is set to '70 B3 D5 7E D0 01 05 17'. At the bottom right, there are 'Cancel' and 'Register' buttons, with the 'Register' button being highlighted by a red box.

Abbildung 7.6: Formular zur Registrierung eines Geräts

Quelle: <https://console.thethingsnetwork.org/applications>

Device ID: Muss einzigartig sein und am besten aussagekräftig, damit man diese ID später einem Gerät zuordnen kann.

Device EUI: Kann man selber bestimmen oder generieren lassen, indem man das Zufällig-Icon bzw. den Stift-Icon anklickt, um es umzuschalten. Dies wird später benötigt, um das Gerät richtig zu konfigurieren!

App Key: Kann man selber bestimmen oder generieren lassen, indem man das Zufällig-Icon bzw. den Stift-Icon anklickt um es umzuschalten. Dies wird später benötigt um das Gerät richtig zu konfigurieren!

App EUI: Wird automatisch generiert von der Webseite.

Nachdem wir ein Gerät erstellt haben, sollten wir die Aktivierungsmethode ändern: Dazu gehen wir in *Application* → *application_id* → *devices* → *device_id* → *Settings*. Darin kann man die Activation Method von OTAA zu ABP ändern.



Abbildung 7.7: Aktivierungsmethoden: links Over-the-Air Activation (OTAA), rechts Activation By Personalization (ABP)

Quelle: <https://console.thethingsnetwork.org/applications>

Wenn wir die Änderungen speichern, kommen wir wieder auf die Seite des Geräts. Scrollen wir ganz nach unten, dann sehen wir ein Codebeispiel.



```
EXAMPLE CODE

1 const char *devAddr = "26011C0D";
2 const char *nwkSKey = "FEDC44AB0EED5686F0A43523612ECE23";
3 const char *appSKey = "DAF11CBE2CE9E1345BA1715CE3407D25";
```

Abbildung 7.8: Beispielcode

Quelle: <https://console.thethingsnetwork.org/applications>

Diesen Codeausschnitt speichern wir oder wir merken uns, dass er auf der Seite des Geräts ist, da wir ihn später zum Konfigurieren des Nodes wieder brauchen werden.

7.3 Aufsetzen eines Clients (LoRa-Node)

Bevor wir die LoRa-Nodes mit Strom versorgen, befestigen wir erstmal eine Antenne an den RFH (Radio Frequency High Band – PIN 23) Anschluss.

Unser LoRa-Node ist über die serielle Schnittstelle UART (PIN 6 + 7 am RN2483) an den Mikrocontroller ATmega644PA (PD0 bzw. PIN 14 + PD1 bzw. PIN 15) angeschlossen und kommuniziert so mit dem Mikrochip. Die Baudrate für die UART Verbindung beträgt standardmäßig 57600 bps, kann aber auch mit einer gewissen Character Folge geändert werden. (Siehe folgende Abbildung 7.9)

1.4 UART INTERFACE

All of the RN2483 module's settings and commands are transmitted over UART using the ASCII interface.

All commands need to be terminated with <CR><LF> and any replies they generate will also be terminated by the same sequence.

The default settings for the UART interface are 57600 bps, 8 bits, no parity, 1 Stop bit, no flow control. The baud rate can be changed by triggering the auto-baud detection sequence of the module. To do this, the host system needs to transmit to the module a break condition followed by a 0x55 character at the new baud rate. The auto-baud detection mechanism can also be triggered during Sleep to wake the module up before the predetermined time has expired.

Note: A break condition is signaled to the module by keeping the UART_RX pin low for longer than the time to transmit a complete character. For example, at the default baud rate of 57600 bps keeping the UART_RX pin low for 938 µs is a valid break condition, whereas at 9600 bps this would be interpreted as a 0x00 character. Thus, the break condition needs to be long enough to still be interpreted as such at the baud rate that is currently in use.

Abbildung 7.9: Spezifikationen für das UART-Interface des RN2483

Quelle: <http://ww1.microchip.com/downloads/en/DeviceDoc/40001784F.pdf>

Kommen wir nun zur Initialisierung des RN2483, wenn dieser nun am Microchip hängt und soweit ansprechbar ist:

Folgende Methoden haben wir von der Drotek RN2483 Library (<https://github.com/drotek/RN2483>) für Arduino genommen und für den AVR angepasst und umgeschrieben. Außerdem haben wir einen Teil von der UART-Library von unserem Betreuer Ulrich Straus übernommen.

Zu aller erst setzen wir den RN2483 zurück, indem wir per UART die Befehle „**sys factoryRESET**“ und „**sys reset**“ senden und jeweils mit einem Break- und New-Line-Character bestätigen. (ASCII: Break = 0x0D; New-Line = 0x0A).

Als nächstes füttern wir den RN2483 mit den Daten, die wir beim Erstellen des Geräts im Applikations-Server erhalten haben. Dazu werden folgende Befehle benötigt:

- **mac set devaddr <address>**
- **mac set nwkskey <nwksesskey>**
- **mac set appskey <appSesskey>**

Also mit dem Beispielcode von Abbildung 7.8 wäre dann:

<address> = 26011C0D;

<nwksesskey> = FEDC44AB0EED5686F0A43523612ECE23;

<appSesskey> = DAF11CBE2CE9E1345BA1715CE3407D25

Nun müssen wir noch den *Spreading Factor* (siehe Kapitel 7.4) festlegen und anschließend die Konfiguration speichern und dem Netzwerk beitreten.

Dies wird wie folgt gemacht:

- **mac set dr 5: wobei 5 = SF7/125 kHz**
- **mac save**
- **mac join abp**

Nach jedem Befehl benutzen wir derzeit noch eine Delay-Funktion um dem RN2483 Chip Zeit zum Bearbeiten und Antworten zu lassen. Die aufgerundeten Delay-Zeiten bzw. Berechnungszeiten sieht man im folgenden Diagramm.

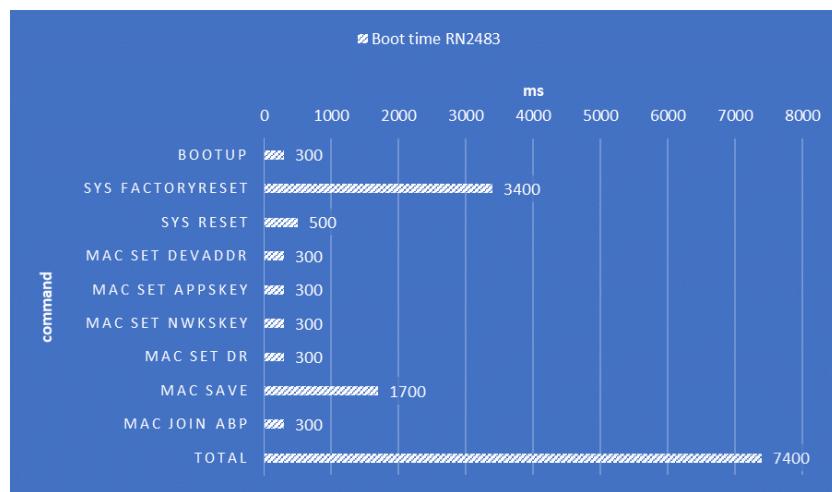


Abbildung 7.10: RN2483 Boot Time unterteilt in die einzelnen Funktionen und der benötigten Zeit in ms

Jetzt ist der RN2483 fertig konfiguriert und wir können mit dem Befehl „**mac tx <type> <portno> <data>**“ Nachrichten versenden. Dabei ist:

<type> = cnf (confirmed) oder uncnf (unconfirmed);

<portno> = Portnummer zwischen einschließlich 1 und 223;

<data> = Die Nachricht als Hexadezimal Wert.

Die vollständige Kommandoliste mit Syntax und Rückgabewerten findet man im RN2483 LoRa Technology Module Command Reference User's Guide: <http://ww1.microchip.com/downloads/en/DeviceDoc/40001784F.pdf>

Wir benötigen aus dieser Liste nur die System Commands aus Kapitel 2.3 und die MAC Commands aus Kapitel 2.4

7.4 Spreading Factor

Der LoRa-Knoten (LoRa-Node) ist nun bereit die Gewichtsinformationen über die Kaffeekanne entgegenzunehmen und als Nachricht verpackt zu senden. Nun geht es um die Feinabstimmung der Konfiguration, um die Reichweite und die Bandbreite des LoRa-Knotens zu bestimmen.

Spreading Factor (kurz: SF) ist, wie der Begriff teilweise schon verrät, ein Faktor, der angibt wie weit sich eine versendete Nachricht ausbreitet, also was für eine Reichweite sie hat. Umso höher dieser Faktor, desto höher die Reichweite. Jedoch leidet die Geschwindigkeit der Übertragung darunter. Man kann sich merken:

- kleiner SF = hohe Bitrate, nicht so hohe Reichweite;
- hoher SF = niedrige Bitrate, hohe Reichweite.

In der folgenden Tabelle ist dies sehr gut erkennbar.

DataRate	Configuration	Indicative physical bit rate [bit/s]
0	LoRa: SF12 / 125 kHz	250
1	LoRa: SF11 / 125 kHz	440
2	LoRa: SF10 / 125 kHz	980
3	LoRa: SF9 / 125 kHz	1760
4	LoRa: SF8 / 125 kHz	3125
5	LoRa: SF7 / 125 kHz	5470
6	LoRa: SF7 / 250 kHz	11000
7	FSK: 50 kbps	50000
8..15	RFU	

Abbildung 7.11: Tabelle mit verschiedenen Spreading Factors und die zugehörige Bitrate

Quelle: <https://witekio.com/wp-content/uploads/2018/01/lora-wan-spreading-factor.png>

Derzeit sind unsere Payloads (Nachrichten) durchschnittlich nur 2 Bytes lang, da wir nur den Füllstand übermitteln. Beim SF7 könnten wir 647 Nachrichten am Tag senden, wenn wir unter dem 30 Sekunden Sendelimit bleiben wollen.

Die genaue Berechnung und weitere Infos zum Spreading Factor findet man auf folgender Google Docs Tabelle: <https://docs.google.com/spreadsheets/d/1QvcKsGeTTPr9icj4XkKXq4r2zTc2j0gsHLrnplzM3I/edit>

7.5 Security / Datensicherheit

Jedes gute System sollte vor Angreifern geschützt sein. Und das ist auch bei LoRaWAN von TheThingsNetwork der Fall. Die Nachrichten werden vor dem Versenden verschlüsselt und beim Empfänger mit Hilfe des App-Session-Key entschlüsselt. Nachrichten von LoRa-Knoten werden als lesbare digitale Daten über ein Frequenzband übermittelt. Da die Frequenzen bekannt sind, kann jeder die Nachrichten abfangen und mitlesen. Die Mitleser sind aber durch die Verschlüsselung nicht in der Lage den Inhalt zu verstehen. Über einen Replay-Angriff (siehe Wikipedia <https://de.wikipedia.org/wiki/Replay-Angriff>) könnte der Interessent sich beim Empfänger aber als jemand anderen ausgeben und so die Entschlüsselung erfahren. Als Maßnahme, um gegen diese Art von Angriff zu schützen, wird ein Frame Counter eingesetzt der jedes Mal, wenn eine Nachricht versendet wird sich um 1

erhöht. So werden alle alten Nachrichten ignoriert und man ist vor Replay-Angriffen geschützt.

Weitere Infos hierzu gibt es auf der TheThingsNetwork Seite zu Security: <https://www.thethingsnetwork.org/docs/lorawan/security.html>

7.6 Methoden und Funktionen zur Kommunikation mit dem LoRa-Modul (UART & Modul RN2483)

Nachdem die LoRa-Kommunikation nun sicher ist, wurde der reibungslose Ablauf im LoRa-Knoten, also zwischen dem LoRa-Modul (RN2483) und dem Mikrocontroller (ATmega644PA), sichergestellt. Die serielle Schnittstelle (UART), die zwischen beiden Bauteilen besteht, implementiert die Methoden des RN2483 für die getaktete Übertragung zum Gateway.

7.6.1 RN2483.c

- **RN2483_init()**: Initialisiert / Konfiguriert das RN2483 Modul wie im Codebeispiel 8.1 zu sehen ist.
- **RN2483_sendData(char *s)**: Konvertiert mit sendHex2(s) das gewünschte Char-Array in Hexadezimal ASCII Code und sendet es mit der UART-Schnittstelle an das Modul, welches anschließend die Nachricht über LoRaWAN an das Gateway schickt.
- **RN2483_sendCmd(char *cmd)**: Sendet ein Kommando über die UART-Schnittstelle an das Modul.
- **RN2483_prepareMessage(int fillRatio)**: Konvertiert ein Integer Wert in Hexadezimal ASCII Code und sendet es über die UART-Schnittstelle an das Modul, welches anschließend die Nachricht über LoRaWAN an das Gateway schickt.
- **delayFunction(int number)**: Führt einen Delay aus, da bei uns maximal nur 850ms am Stück mit der Delay Funktion gewartet werden kann. Bei Parameter-Wert 1 wird 3400ms gewartet, bei 2 wird 500ms gewartet, bei 3 wird 300ms gewartet und bei 4 wird 1700ms gewartet.

7.6.2 uart.c

- **init_uart()**: Initialisiert die UART-Schnittstelle, d.h. setzt die Baudrate, aktiviert die RX und TX Pins und setzt den asynchronen UART Modus mit 8 Data bits, no parity, 1 stop bit.
- **uart_putc(unsigned char c)**: Sendet einen Buchstaben über UART, wenn die Schnittstelle bereit ist.

- **sendString(char tempStringChar[]):** Sendet ein Char-Array über UART, indem es für jeden Char die uart_putc() Methode aufruft.
- **sendHex2(char *s):** Konvertiert das gegebene Char-Array in Hexadezimal ASCII Code und sendet es über die sendString() Methode.

8 Webserver für die Visualisierung

8.1 Visualisierung der empfangenen Daten mit Angular CLI

Mit dem aufgesetzten Gateway, welches die Gewichtsdaten/Füllstände über verschlüsselte Kanäle ins Internet überträgt, fehlt noch die Komponente der Visualisierung. Der Webserver, der den Clients die Weboberfläche (das Front-End) zur Verfügung stellt, übernimmt die Aufgabe der Aufbereitung und Darstellung der übermittelten Daten. Das Front-End ist die Ansicht, auf welche der Kunde blickt, um die Füllstände aller Behälter zu sehen. Sie ist minimalistisch gehalten, sodass keine Überladung entsteht und das menschliche Auge die relevante Information schneller erkennt.

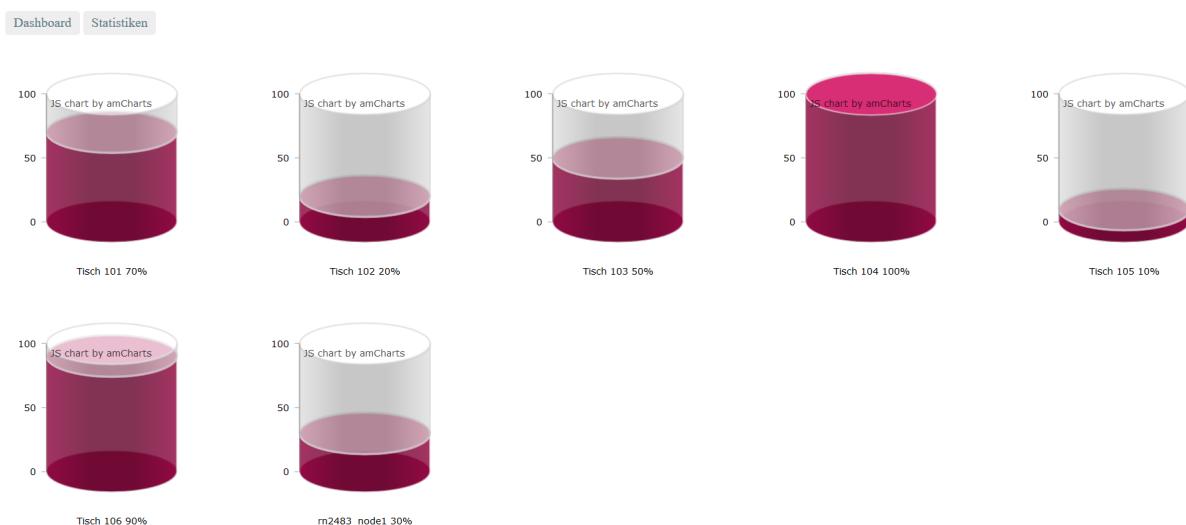


Abbildung 8.1: Übersicht der Füllstände der einzelnen Kaffeemaschinen auf der Webseite

Die Webseite (siehe Abbildung 8.1) zeigt links oben zwei Buttons, die zwischen zwei Anzeigen hin- und herschalten lassen. Die eigentliche Haupt- bzw. Startseite (Dashboard - siehe Abbildung 8.1) mit den Zylindern und deren Inhalten - jeder aktive Node wird hier mit seinem aktuellen Füllstand angezeigt. Und die Statistik-Seite (siehe Abbildung 8.2), die auf einem größeren Diagramm die historischen Werte bis zum aktuellen Zeitpunkt anzeigt.

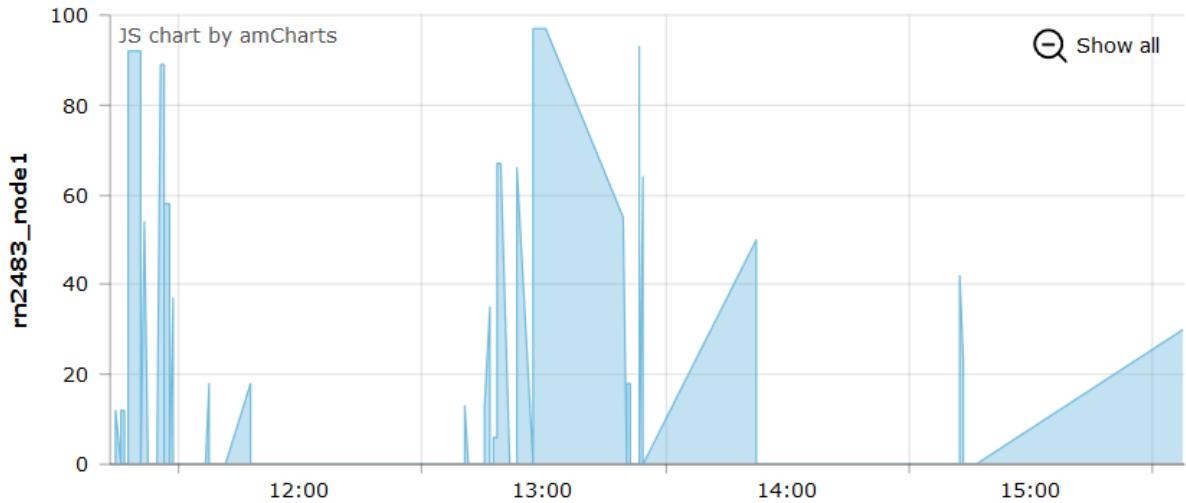


Abbildung 8.2: Füllstand-Verlauf einer Kaffeemaschine von 12:00 Uhr bis 15:00 Uhr

Die Visualisierung ist mithilfe des Front-End-Frameworks Angular CLI geschrieben worden. Die Programmiersprache von Angular ist TypeScript, eine Weiterentwicklung von JavaScript, mit einigen Verbesserungen wie die Typisierung von Variablen. Weiterhin wurden die Zylinder-Behälter mit der „AmCharts“ Angular-Bibliothek implementiert. Diese Bibliothek bietet die Möglichkeit diese Diagramme mit Hilfe einer „create“ Funktion zu erstellen und einer „update“ Funktion mit neuen Daten zu versorgen und damit das Diagramm in Echtzeit zu verändern. Darüber hinaus wurden die Statistikdiagramme mit Hilfe der gleichen Bibliothek erstellt. Diese Diagramme bieten die gleichen Funktionen wie die Zylinderdiagramme.

8.2 NodeJS Server für die Verarbeitung der Datenpakete

Der Server ist in der Programmiersprache NodeJS geschrieben. Darin wurde ein Express Server implementiert. Dieser Express Server nutzt einen Websocket, um die Verbindung zwischen Server und Client herzustellen. Diese Verbindung wird von dem NPM (Node Package Manager) Modul „ttn“ genutzt. Hierbei ist ttn eine Bibliothek des Networkservers „The Things Network“. Durch das ttn Modul bekommt der Server die Nachrichten vom Network Server und damit kann das Paket verarbeitet und per Websocket an den Client weitergeleitet werden. Während der Weiterverarbeitung erfolgt auch die Speicherung einkommender Daten.

8.3 The Things Network

Das TTN Modul (SDK) ist die Schnittstelle zwischen dem Express Server und dem Network Server. Die Verbindung wird automatisch hergestellt. Damit jedoch eine Verbindung hergestellt werden kann, müssen im Voraus zwei Werte vorhanden sein. Der

erste ist die Application ID. Diese ID beschreibt das Projekt, welches vorher bei der TheThingsNetwork-Webseite festgelegt wurde. In diesem Fall ist die Application ID: lora_measure_network. Der zweite Wert ist der accessKey. Dieser ist für die Authentifizierung des SDK's mit dem Network Server erforderlich. Der Key muss vorher über die Webseite generiert werden. Die Nachrichten, zwischen Network Server und SDK, sind in einem bestimmten Format:

```
{  
    app_id: 'lora_measure_network',  
    dev_id: 'rn2483_node1',  
    hardware_serial: '0004A30B001C6A30',  
    port: 1,  
    counter: 0,  
    payload_raw: <Buffer 33 30>,  
    metadata: { time: '2018-07-05T18:04:13.482082016Z' },  
    message: '30'  
}
```

Listing 8.1: Payload

- **app_id:** Application ID
- **dev_id:** Das Gerät, welches die Nachricht gesendet hat
- **hardware_serial:** Hardwarenummer des Nodes
- **counter:** Messagecounter der bei jeder Nachricht um 1 inkrementiert
- **payload_raw:** Die Nachricht wird vom Network Server als Hexadezimal
- **metadata:** In diesem Objekt ist ein Zeitstempel enthalten. Dieser Zeitstempel ist von dem Zeitpunkt, als die Nachricht im Network Server angekommen ist.
- **message:** Das ist die Payload, welche vom payload_raw geparsst wurde

8.4 Inaktivität der Behälter

Sobald ein Gerät inaktiv wird (d.h. eine Stunde ohne Nachrichtenempfang), ist es für die Nutzer in der UI nicht mehr möglich, das Gerät zu sehen. Diese Inaktivität würde bei einem Thermobehälter die Abkühlung bedeuten. Das Limit basiert auf dem Resultat einer Gesprächsrunde innerhalb der Entwickler. Sobald jedoch das Gerät wieder sendet, wird es in der UI eingeblendet. Dieses Limit hat den Hintergrund, dass ein Gerät unerwartet vom Netz genommen wird. Wenn dieser Fall eintreten würde, könnte man in der Visualisierung nicht erkennen, ob das Gerät überhaupt noch existiert.

8.5 Voraussetzungen und Nutzung des Node-Servers

Unterstützte Betriebssystem

- Windows
- Linux
- Unix

Benötigte Software

- NodeJS (inkl. NPM)
- Angular

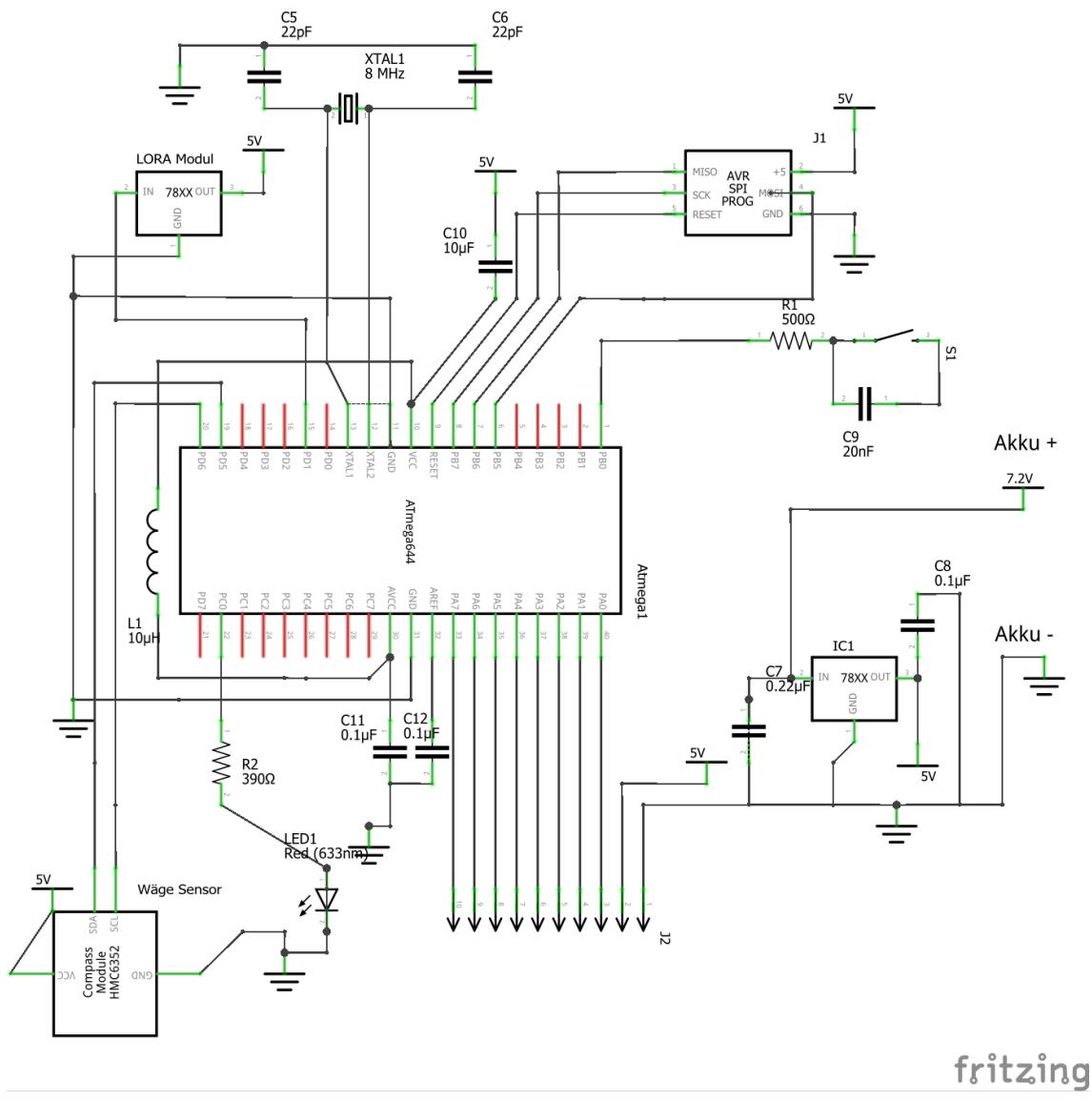
Nutzung

- Starte zwei Kommandozeilenprogramme
 - Konsole 1 im Ordner „server“: node server.js
 - Konsole 2 im Ordner „client“: ng serve -open

9 Schaltplan und Verlöten der Platine

Nachdem alle Komponenten nun miteinander interagieren, wurde zum Abschluss der LoRa-Knoten, der bisher aus Entwicklungsboards und Modulen zusammengesteckt war, in eine eigene Platine verwandelt. Ein designter Schaltplan half die Einzelteile auf einer Lochrasterplatine zu verlöten.

Der Schaltplan wurde mit dem Programm Fritzing (<http://fritzing.org/home>) erstellt.



fritzing

Abbildung 9.1: Schaltplan, erstellt mit Fritzing

Als Mikrocontroller haben wir im Schaltplan einen ATmega644 statt einem ATmega644PA verwendet, da diese fast gleich sind. Ausgehend vom Mikrocontroller werden an den Pins XTAL1 und XTAL2 der Quarz angeschlossen. Die Pins PA0 bis PA7 sind für ein LCD Display reserviert, auf dem Gewichtsinformationen, aber auch Debug-Meldungen ausgegeben werden können. Der Wägesensor ist mit den Pins PD5 und PD6 verbunden. Wobei hier noch die notwendigen Pullup-Widerstände für die I²C-(Two-Wire-) Schnittstelle fehlen. Für das Wägesensor-Bauteil (im Schaltplan fälschlicherweise als Compass, HMC betitelt) wurde ein Platzhalter-Bauteil mit vier Pins verwendet, da kein passendes Layout für unseren Wägesensor gefunden wurde. Das LoRa-Modul wird an PD1 angeschlossen und es wurde ebenso eine Platzhalterkomponente mit drei Pins verwendet. Zudem haben wir einen Taster auf PD0 für die direkte Kommunikation mit dem Wäge-Modul und an PD5 – PD7 einen ISP-Anschluss für die Programmierung angelötet. Die Platine und dessen Bauteile werden mit zwei in Reihe geschalteten Akkus versorgt die jeweils eine Spannung von ca. 3,6 Volt haben und damit eine Gesamtspannung von 7,2 Volt liefern. Dazwischen geschaltet ist ein DC-DC Wandler, der den Stromkreis auf konstante 5V versorgt.

Die Rückseite der gelöteten Platine:

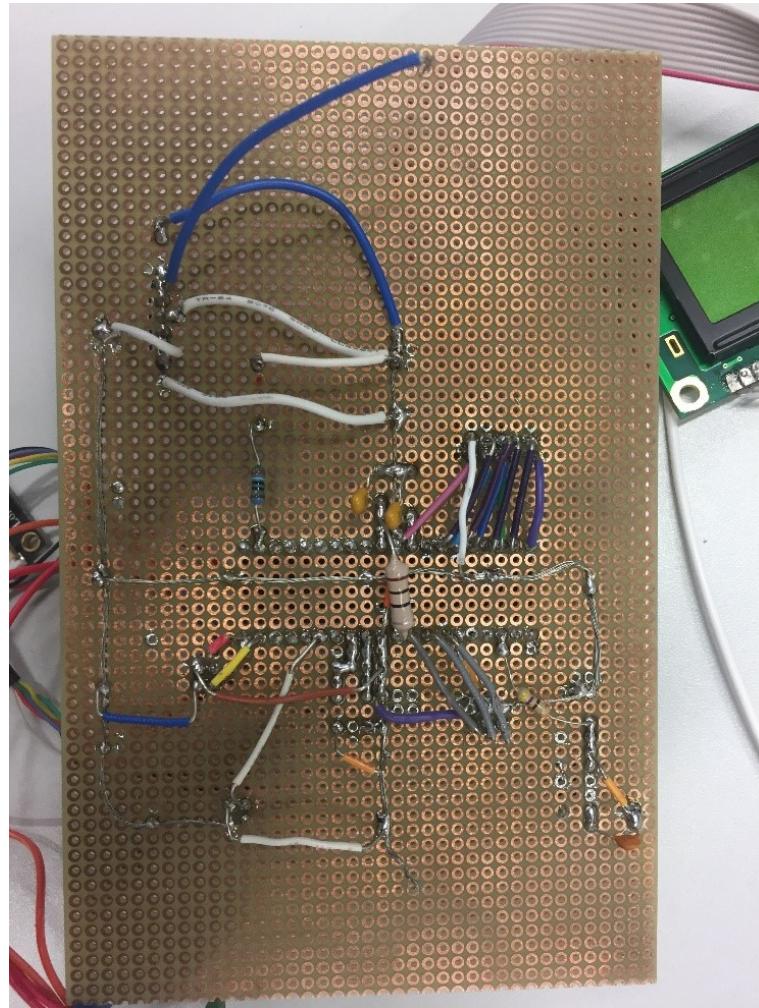


Abbildung 9.2: Rückseite der verlötzten Platine unseres ersten Prototyps

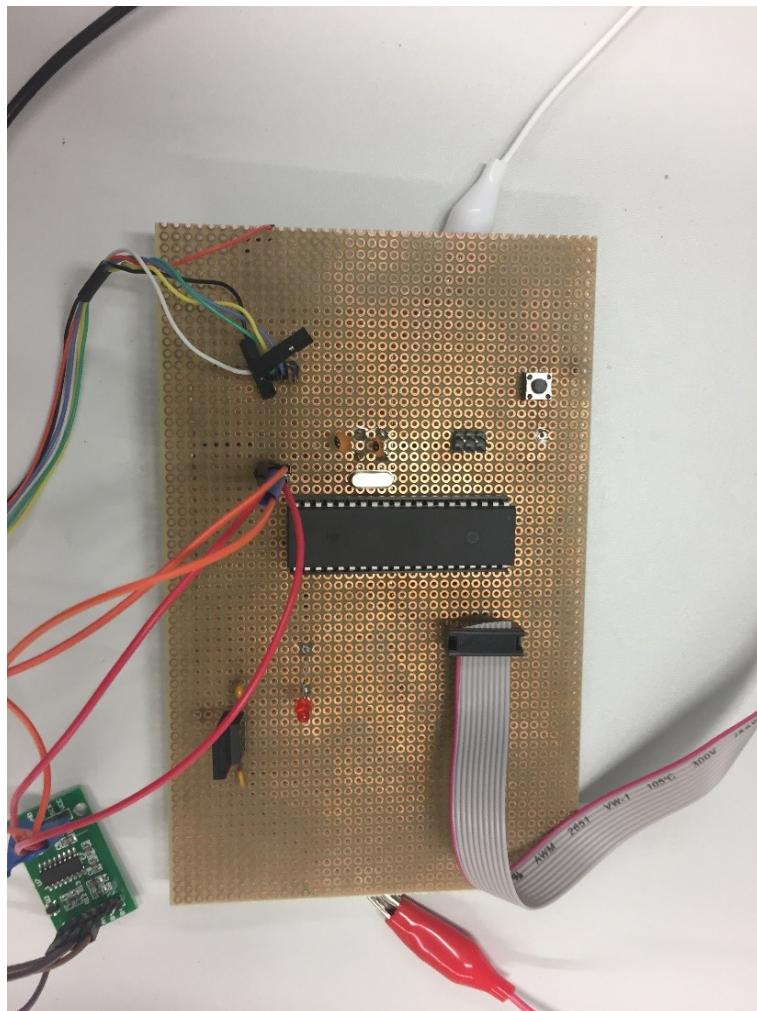


Abbildung 9.3: Vorderseite der verlötenen Platine unseres ersten Prototyps

Dem Taster wurde eine Tar-Funktion zugewiesen, mit der das Gewicht genullt werden kann. Softwareseitig ist es auch möglich dem Taster andere Funktionen zuzuweisen, wie beispielsweise das An- und Ausschalten des Gerätes oder mit Verbund eines Displays zwischen verschiedenen Anzeigemodi zu wechseln.

10 Fazit der prototypischen Umsetzung eines Wägesystems für den Gastronomie-Betrieb via LoRaWAN

Zusammenfassend sei zu sagen, dass der Aufbau, die Funktion und der Nutzen der angenommenen Vorteile vollumfänglich realisiert werden konnte.

Alle Komponenten arbeiten nach Konfiguration und Implementierung exakt so zusammen, das der reibungslose Ablauf des Prozesses der ständigen Visualisierung von Füllstand und dem Nachfüllen der Behältnisse gewährleistet ist.

LoRa bietet über verschiedene Kommunikationswege eine sichere, energieeffiziente Übertragung und kann alle nötigen Daten zur Visualisierung übermitteln. Die Zyklen, die zur Übermittlung zu Verfügung stehen, sind ausreichend kurz damit die nutzbaren Informationen, also Füllstände von Kaffeebehältnissen oder anderen Behältnissen, clientseitig in Echtzeit zu sehen sind. Wobei der Energieverbrauch im Einzelnen nicht untersucht wurde, konnten Beobachtungen an den verwendeten Akkupacks Rückschlüsse auf die enorm niedrigen Energiekosten zulassen.

Für eine weiträumige Nutzung, konnten Tests anderer Gruppen zeigen, dass selbst in riesigen Gastro-Anlagen die nötigen Reichweiten erzielt werden können, die für eine Füllstands-Anwendung nötig sind.

Zudem läuft das System plattformunabhängig, sodass man das System auf vielen verschiedenen Geräten verwenden kann. Die Visualisierung wird im Browser, beispielsweise Internet Explorer oder Google Chrome angezeigt, sodass keine weiteren Installationen nötig sind. Der Server läuft auch auf diversen Betriebssystemen wie Windows oder MacOS, sodass hier keine großen Umstellungen nötig sind. Außerdem ist es auch möglich die Visualisierung auf einem Smartphone aufzurufen, beispielsweise auf einem iPhone, da diese auch einen Browser installiert haben, sodass man die Füllstände auch zwischendurch und schnell überprüfen kann.

Dadurch, dass das System plattformunabhängig implementierbar ist, kann man es unkompliziert und flexibel in ein bestehendes System des Hotels oder Gastronomie integrieren. Eine mögliche Option ist, dass man den Raspberry Pi an einen PC anschließt und eine Installation startet, damit die ganzen Systeme installiert und hochgefahren werden.

Abschließend muss erwähnt werden, dass ein großes Verbesserungspotential vorhanden ist, sodass weitere nützliche Funktionen eingebaut werden können, damit die Prozesse in der Gastronomie und in Hotels verbessert werden.

11 Ausblick auf die Kommerzialisierung

11.1 LoRaWAN Modul Code verbessern

- Firmware vom **RN2483** updaten
- Gebrauch von der sleep Funktion „sys sleep <length>“ machen um Strom zu sparen
- Beim Ausschalten den Frame-Counter mit „mac get upctr“ auslesen und in den EEPROM vom ATmega644PA speichern. Beim Einschalten den Frame Counter aus dem EEPROM auslesen und mit „mac set upctr <fCntUp>“ wieder neu setzen, damit man nicht jedes Mal den Frame Counter in der Konsole von The-ThingsNetwork zurücksetzen muss.
- Empfang von Nachrichten oder Befehlen vom Gateway ermöglichen und durch UART-Receive diese Nachrichten bzw. Befehle im Mikrocontroller auslesen.
- Delays durch Interrupts ersetzen, so dass das Programm weiterarbeiten kann, während es auf ein Ereignis wartet.
- Verschiedene Profile oder Modis passend zu den *Spreading Factors* (siehe Kapitel 7.4) programmieren z.B.: niedrige Reichweite + hohe Bandbreite, hohe Reichweite + niedrige Bandbreite, etc.

11.2 Front-End Lizenzkosten vermeiden

Das Frontend hat das Problem, dass es die *amCharts-Bibliothek* nutzt. Die Bibliothek ist für Open-Source- und nicht-kommerzielle Projekte frei verfügbar. Die Lizenz (Stand 06.07.2018) kostet für ein Jahr 4590 €. Darin sind die Diagramme in den Seiten Dashboard und Statistik enthalten. In Zukunft sollen selbstentwickelte Diagramme verwendet werden, um Lizenzkosten zu vermeiden.

11.3 Erstellen eines wasserfesten Gehäuses

In naher Zukunft ist es außerdem noch wichtig, dass sich die Platine und die nötige Hardware in einem Gehäuse befinden, da es sonst nicht im kommerziellen Umfeld nutzbar wäre. Zudem sollte das Gehäuse wasserfest sein, da sonst Kaffee oder ähnliches hindurchkommen und etwas beschädigen könnte.

Dadurch wird unser Produkt deutlich kompakter und nutzvoller für Hotels. Unser erster Ansatz wäre ein Prototyp mit einem 3D-Drucker zu erstellen.

11.4 Kommerzialisierung und mögliches Geschäftsmodell

Die nächsten nötigen Schritte zur Kommerzialisierung sind:

1. Kundengespräche:

Wir müssen Gespräche mit unseren potenziellen Kunden, beispielsweise Hotels oder Restaurants führen, damit wir abschätzen können, wie hoch die Nachfrage sein könnte. Hierbei bekommen wir direktes Feedback zu unseren Prototypen, sodass darauf reagiert werden kann. Außerdem müssen wir Kundengruppen erschließen, damit wir eingrenzen können, in welchen Branchen das Produkt nutzbar ist.

2. Optimierung des Prototyps:

Wir müssen ein wasserdichtes Gehäuse erstellen, damit die Technik verpackt werden kann. Zudem muss das Design des Gehäuses angepasst werden, sodass es in der Gastronomie verwendbar ist. Des Weiteren werden wir den Prototypen verbessern, indem wir einen Sleep-Mode einführen, damit die Laufzeit verlängert wird, den Taster verbessern und unnötige Komponenten entfernen.

3. Erstellen von Business-Plan, Strategie und Dokumente für Unternehmensgründung:

Für eine erfolgreiche Start-Up-Gründung benötigen wir einen Business-Plan, damit wir einen seriösen roten Faden haben und wir damit Investoren überzeugen können. Zudem müssen wir uns Strategien überlegen und die nächsten Schritte planen. Dazu benötigen wir noch weitere Dokumente und Berechnungen für die Unternehmensgründung.

Mögliche Geschäftsmodelle:

Um den Aufwand für Gastronomiebetreiber möglichst klein zu halten, sollten Geräte- und Softwarewartung möglichst outsourct sein. Gastronomiebetreiber bezahlen monatlich/jährlich für die Nutzung des Systems eine festgelegte Summe. Darin sind Installationen und Wartungsarbeiten enthalten. Für Erweiterungen und/oder Updates fallen weitere Einmalgebühren und eventuell Aufschläge im jährlichen Turnus an. Der

Server wird im cloudbasierten Betrieb realisiert, was Wartung und Updates der lokalen Installationen vereinfacht. Lediglich der Austausch und die initiale Installation vor Ort macht einen Besuch nötig.

Literatur

- Electronics, Drotek. *DataLink LoRa RN2483*. <https://drotek.com/shop/en/lora/763-data-link-lora-rn2483.html>.
- Electronics, Mouser. „Analog-to-Digital Converter“. https://www.mouser.com/ds/2/813/hx711_english-1022875.pdf.
- Inc., Microchip Technology. 2015-2017a. *Low-Power Long Range LoRa Technology Transceiver Module*. <http://ww1.microchip.com/downloads/en/DeviceDoc/50002346C.pdf>.
- . 2015-2017b. *RN2483 LoRa Technology Module Command Reference User's Guide*. <http://ww1.microchip.com/downloads/en/DeviceDoc/40001784F.pdf>.
- Kaiser, Reto. *ic880a Gateway*. <https://github.com/ttn-zh/ic880a-gateway/wiki>.
- LoRa(WAN) airtime calculator*. <https://docs.google.com/spreadsheets/d/1QvcKsGeTPPr9icj4XkKXq4r2zTc2j0gsHLrnplzM3I/edit#gid=0>.
- mstanley. *LoRa WAN*. <http://www.mstanley.co.uk/blog/wp-content/uploads/2015/11/Enabling-world-wide-mobility-for-the-IoT-image-2.jpg>.
- Network, The Things. *LoRaWAN Security*. <https://www.thethingsnetwork.org/docs lorawan/security.html>.
- Sparkfun. *Gewichtsmessung*. <https://learn.sparkfun.com/tutorials/getting-started-with-load-cells>.
- Wikipedia contributors. 2018a. *Force-sensing resistor – Wikipedia, The Free Encyclopedia*. [Online; accessed 13-August-2018]. https://en.wikipedia.org/w/index.php?title=Force-sensing_resistor&oldid=846591724.
- . 2018b. *Load cell – Wikipedia, The Free Encyclopedia*. [Online; accessed 13-August-2018]. https://en.wikipedia.org/w/index.php?title=Load_cell&oldid=851413016.
- . 2018c. *Piezoelectric sensor – Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Piezoelectric_sensor&oldid=839026087. [Online; accessed 13-August-2018].
- Witekio. *LoRa WAN spreading factor*. [https://witekio.com/wp-content/uploads/2018/01/lora-wan-spreading-factor.png](http://witekio.com/wp-content/uploads/2018/01/lora-wan-spreading-factor.png).