HTW-Dresden

Allgemeine Informatik

LoRaWAN Wetterstation SEN-15901

Projektseminar
Wintersemester 21/22

Name:	Reitz	Rezaii-Djafari
Vorname:	Lenny	Raphael
s-Nummer:	s80452	s80455

Tag der Einreichung: 1. März 2022

Professor: Prof. Dr.-Ing. Jörg Vogt

Inhaltsverzeichnis

1 Einleitung

Das Projekt wurde im Laufe des Projektseminars 2021 von Lenny Reitz und Raphael Rezaii-Djafari unter der Führung von Professor Jörg Vogt an der HTW Dresden entwickelt.

1.1 Die Übertragungstechnik: LoRa

LoRa (Abkürzung für Long Range) ist eine patentierte und proprietäre drahtlose Modulationstechnik basierend auf Chirp Spread Spectrum. Seit seiner Veröffentlichung in 2015 durch die Semtech Corporation hat es sich für IoT Geräte bewährt, die außerhalb der konventionellen Reichweiten von WLAN, Bluetooth und ZigBee operieren.

Anders als Bluetooth und WLAN können nur geringe Datenmengen mit niedriger Bitrate gesendet werden, dafür aber über eine erheblich größere Reichweite bei geringem Energieverbrauch. LoRa funkt standardmäßig auf dem freien Sub-1-GHz Band, weshalb die nutzbaren Frequenzen, Fair-Use-Policies und Vorschriften am Einsatzort beachtet werden müssen. Für eine höhere Datenrate und keine Beschränkungen gibt es auch eine Variante mit 2.4GHz auf die wir nicht weiter eingehen.

1.2 Das Protokoll & die Architektur: LoRaWAN

LoRaWAN wird Open Source von der LoRa Alliance als Systemarchitektur und Netzwerkprotokoll auf der Vermittlungsschicht entwickelt.

1.2.1 Systemarchitektur

Vermaschte Netze werden genutzt um eine große Abdeckung kosteneffizient zu erzielen, allerdings steht der Stromverbrauch durch die Komplexität und das Weiterleiten der Pakete durch die Geräte, ab hier Endknoten genannt, im Widerspruch zur geforderten Energieeffizienz der Endknoten. Da LoRa ohnehin eine große Reichweite besitzt, wird eine Stern-Topologie verwendet.

Netzwerkknoten stehen dabei nicht für ein einzelnes, sondern für eine beliebig große Anzahl geographisch naher Gateways. Wenn ein Endknoten Daten sendet, passiert dies durch Broadcasts. Alle Gateways, die das Paket erhalten haben, senden es an den Netzwerk Server, der redundante Pakete

herausfiltert, Sicherheitschecks durchführt und schließlich das Paket an seinen Bestimmungsort weiterleitet.

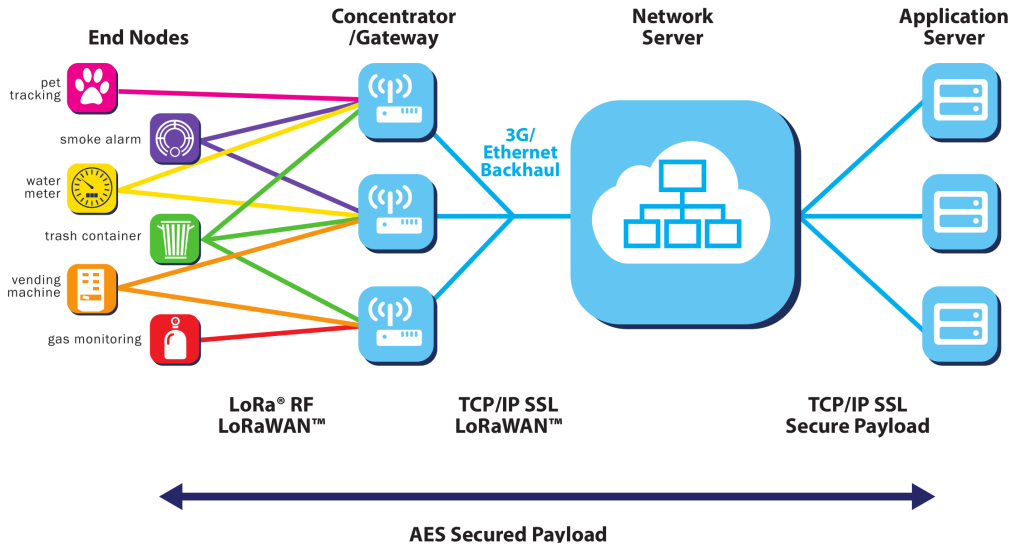


Abbildung 1: LoRaWAN Architektur[1]

Dadurch verlagert sich die Komplexität von den Endknoten und Gateways auf den Netzwerk Server. Außerdem wird die Verfügbarkeit verbessert, wenn ein Gateway ausfällt oder ein Endknoten mobil ist, da kein Handover notwendig ist.[1]

1.2.2 Geräteklassen

Nicht jedes IoT Gerät hat die gleichen Anforderungen. Generell gilt, dass die Kommunikation bidirektional, asynchron und verschlüsselt erfolgt, wobei Endknoten Daten zum Netzwerk senden sobald diese verfügbar sind. Der Unterschied zwischen den Klassen liegt darin, wann ein Endknoten Daten vom Netzwerk empfangen kann.

- **A: batteriebetriebene Sensoren**

  Ein Empfangsfenster wird nur geöffnet nachdem Daten an das Netzwerk übertragen wurden. Das Netzwerk muss Downlink Kommunika-

[1]LoRa Alliance. (2015). A technical overview of LoRa and LoRaWAN [Ebook] (pp. 7-9). https://lora-alliance.org/wp-content/uploads/2020/11/what-is-lorawan.pdf

tion für das Gerät solange bereithalten bis dieses Daten sendet. Klasse A ist am energiesparsamsten und muss von allen Geräten unterstützt werden.

- **B: batteriebetriebenes Steuergerät**

  Zusätzlich zu den Eigenschaften der Klasse A, öffnen Geräte der Klasse B das Empfangsfenster zu festgelegten Zeiten. Eine zeitliche Synchronisation mit dem Gateway ist erforderlich.

- **C: Steuergerät mit fester Stromversorgung**

  Das Empfangsfenster der Geräte der Klasse C ist permanent geöffnet. Es wird nur bei Übertragungen geschlossen.

1.2.3 LoRaWAN Netzbetreiber

Aufgrund der Open Source Natur von LoRaWAN gibt es die verschiedensten Netzbetreiber. Wir werden uns auf das populäre The Things Network / The Things Stack (TTN) beschränken. Es ist frei nutzbar, allerdings gibt es neben den Frequenzrichtlinien, keine Uptime Verpflichtungen und eine Fair-Use-Policy:

- Die Uplink Airtime (Endknoten → TTN) ist auf 30 Sekunden je 24 Stunden pro Endknoten beschränkt.
- Es dürfen maximal 10 Downlink Nachrichten (TTN → Endknoten) je 24 Stunden pro Endknoten gesendet werden.

Für die nachfolgende Problemstellung eignet sich das TTN optimal. Es sei jedoch erwähnt, dass das kommerzielle The Things Industries (TTI) mit weniger Beschränkungen und SLAs existiert.

1.3 Problemstellung

Wir haben eine analoge Wetterstation und wollen diese an das TTN anbinden. Dafür nutzen wir einen Arduino Uno mit LoRa-Shield.

Die Wetterstation muss verbunden und ein Programm geschrieben werden, dass die Daten ausliest und versendet. Die Auflösung und Anzahl der Datensätze, sowie Übertragungsparameter, Authentifizierungsmethoden und Schlüssel sollen einfach konfigurierbar sein.

2 Übersicht

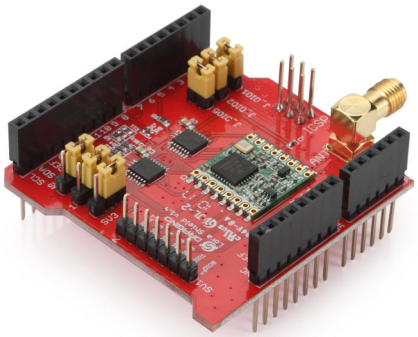
2.1 Hardware

Folgende Hardware ist Voraussetzung für den Aufbau des Projekts. Ein LoRa-Gateway muss dafür in Reichweite des Arduinos sein. Sind bereits Gateways in unmittelbarer Reichweite, ist es nicht unbedingt erforderlich ein extra Gateway zu installieren. Im Sinne der Unabhängigkeit und Effizienz der Kommunikation wird jedoch empfohlen, ein eigenes Gateway einzusetzen.

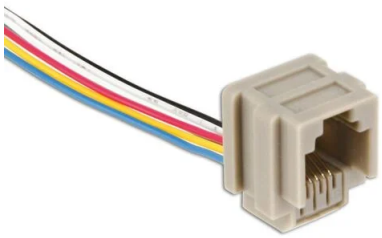
- Wettermessgerät-Kit SEN-15901 (enthält Regenmesser, Anemometer, Windfahne)
- Arduino Uno
- Dragino Lora Shield v1.4 (für Arduino Uno)
- LoRa Gateway (z.B. Dragino LPS8)
- 2x Modular-Einbaubuchsen mit Anschlusslitzen, 6P6C
- Breadboard
- Jumper-Kabel
- 1x Widerstand (10 kOhm)

(a) Wetterstation SEN-15901

(b) Dragino LoRa Gateway LPS8



(c) Dragino LoRa Shield 1.4 für Arduino



(d) RJ11 (6P6C) Einbaubuchse mit Anschlusslitzen

Abbildung 2: Auswahl der benötigten Hardware [2]

[2](a) Berrybase, Wettermessgerät-Kit für Windgeschwindigkeit, Windrichtung und Niederschlag. [image] berrybase.de/sensoren-module/feuchtigkeit/wettermessger-228-t-kit-f-252-r-windgeschwindigkeit-windrichtung-und-niederschlag. (b) Dragino, LPS8 Indoor LoRaWAN Gateway. [image] dragino.com/products/lora-lorawan-gateway/item/148-lps8.html. (c) Dragino Wiki, Dragino LoRa Shield v1.4. [image] wiki.dragino.com/index.php?title=Lora_Shield. (d) ManoMano, Modular-Einbaubuchse mit Anschlusslitzen, 6P6C. [image] manomano.de/p/modular-einbaubuchse-mit-anschlusslitzen-6p6c-13397080.

2.2 Projektaufbau

2.2.1 Wetterstation

Die Wetterstation hat 3 Sensoren (Regenmesser, Anemometer, Windfahne) und besitzt zwei RJ11-Stecker, über die diese ausgelesen werden können. Die RJ11-Stecker werden mit den beiden Einbaubuchsen verbunden, um die einzelnen Anschlüsse zuordnen zu können. Dann ergibt sich anhand Kabelreihenfolge bzw. -farbe folgende Belegung:

Abbildung 3: Belegung der beiden RJ11-Stecker der Wetterstation

Ein Stecker ist demnach zum Auslesen des Regenmessers und der andere für Windfahne und Windgeschwindigkeit.

2.2.2 Arduino Uno (mit LoRa-Shield)

Der Arduino Uno bietet 6 analoge und 14 digital belegbare Pins. Durch den aufgesetzten LoRa-Shield werden die digitalen Pins 2 und 9 bereits belegt (siehe LoRa Shield Pin Mapping). Diese beiden Pins können also zum Anschließen der Sensoren nicht mehr genutzt werden. Ein beispielhafter Aufbau könnte demnach so aussehen:

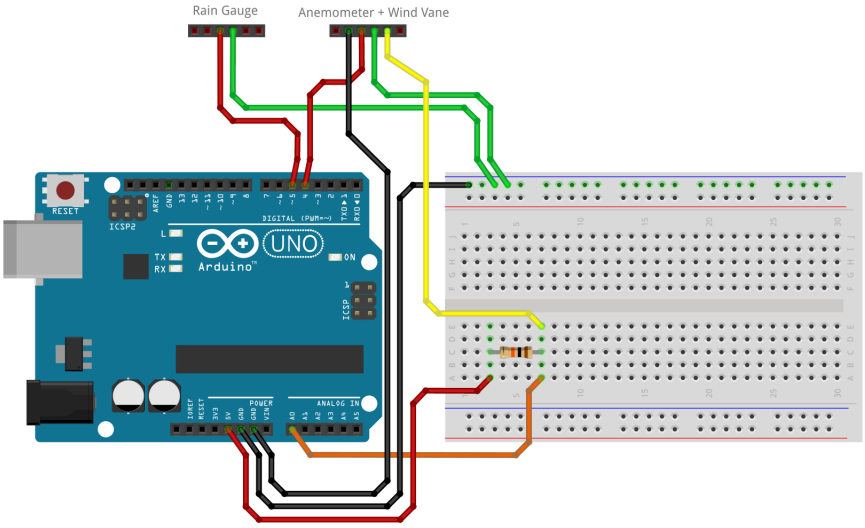


Abbildung 4: Beispielhafter Aufbau, um die Wetterstation an den Arduino anzuschließen. (Das aufgesteckte LoRa-Shield fehlt in dieser Abbildung)

Der Regenmesser und das Anemometer werden über die digitalen Pins 4 und 5 verbunden. Die Windfahne wird an den analogen Pin A0 über einen Widerstand (10 kOhm) angeschlossen. Einzelheiten zu der Funktionsweise der Sensoren werden in Abschnitt 3.1 erläutert.

3 Funktionsweise

3.1 Wetterstation

Die drei Sensoren der Wetterstation sind passive Bauelemente. Zum Messen muss also eine Spannung von außen (vom Arduino) angelegt werden.

3.1.1 Regenmesser

Der Regenmesser besteht aus einem Auffangbehälter und einem darin befindlichen einfachen Kippschalter. Wenn sich genügend Wasser im Behälter gesammelt hat, kippt der Schalter um und das Wasser läuft wieder aus dem Behälter heraus. Ein Kippvorgang entspricht einer Wassermenge von 0,2794

mm. Die folgende Umrechnung ergibt die Wassermenge in Milliliter bei einer Auffangfläche von 55 cm^2 (entspricht der Auffangfläche des Regenmessers):

$$1\,mm = 1\,l/m^2 \quad \text{(Niederschlag)}$$
$$\Rightarrow 0.2794\,mm = 0.2794\,l/m^2 = 0.02794\,ml/cm^2$$
$$\Rightarrow 0.02794\,ml/cm^2 * 55\,cm^2 \approx 1.54\,ml \quad (1)$$

Ein Kippvorgang beträgt also im Idealfall 1,54 ml. Mehrere Testläufe haben aber gezeigt, dass der Regenmesser mit einer relativ großen Ungenauigkeit schaltet. Je nach dem, wie schnell das Wasser einfließt, variiert die Messgenauigkeit stark.

3.1.2 Anemometer

Das Schalenanemometer nimmt Windgeschwindigkeiten auf, indem ein Reed-Schalter durch einen Magneten geschaltet wird. Je schneller sich das Anemometer dreht, umso größer wird die Schaltfrequenz. Eine Windgeschwindigkeit von 2.4 km/h schließt den Schalter mit einer Frequenz von 1 Hz.

Eine Umdrehung des Anemometers entsprechen 3 Schaltvorgängen.

3.1.3 Windfahne

Die Windfahne besitzt 8 Schalter, welche jeweils mit unterschiedlich großen Widerständen verbunden sind. Die Schalter können einzeln oder in Paaren umgelegt werden, wodurch sich 16 mögliche Widerstandswerte und damit Himmelsrichtungen bestimmen lassen.

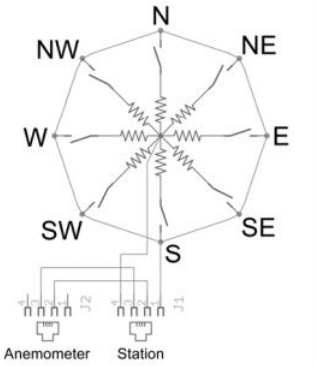


Abbildung 5: Aufbau der Windfahne mit den 8 verschiedenen Widerständen[3]

Folgende Tabelle zeigt, wie die einzelnen Windrichtungen bei einer Betriebsspannung von 5 V und einem Widerstand von 10 kOhm aus der resultierenden Spannung abgelesen werden:

Windrichtung	Richtung in Grad	Spannung in V
N	0.0	3.84
NNE	22.5	1.98
NE	45.0	2.25
ENE	67.5	0.41
E	90.0	0.45
ESE	112.5	0.32
SE	135.0	0.9
SSE	157.5	0.62
S	180.0	1.4
SSW	202.5	1.19
SW	225.0	3.08
WSW	247.5	2.93
W	270.0	4.62
WNW	292.5	4.04
NW	315.0	4.33
NNW	337.5	3.43

[3]SparkFun, The eight switches and their respective resistors internally on the wind vane. [image] https://learn.sparkfun.com/tutorials/weather-meter-hookup-guide.

3.2 Arduino

Der Arduino Uno empfängt die Signale der Sensoren der Wetterstation über digitale und analoge Pins. Diese Signale müssen entsprechend verarbeitet und interpretiert werden und im Anschluss mithilfe des LoRa-Shields an das TTN übergeben werden.

3.2.1 Erfassung der Messwerte

Über Interrupts (siehe Code-Ausschnitt), die beim Zustandswechsel der digitalen Pins ausgelöst werden, kann die Anzahl der Schaltvorgänge in einfachen Variablen (Counter) inkrementell erfasst werden. Der Arduino berechnet jede Sekunde einen neuen Messwert, daher werden diese Counter für Regenmesser und Anemometer jede Sekunde zurückgesetzt. Beide Sensoren werden außerdem zeitlich entprellt, sodass nur einmal pro Schaltvorgang inkrementiert wird. Das Zeitfenster, in dem die Sensoren entprellt werden, ist auf die technischen Eigenschaften der Sensoren abgestimmt, d.h. die Entprellung ist nicht so stark, dass im Zweifel Messwerte verloren gehen könnten.

Für die Windfahne muss keine Anzahl von Schaltvorgängen ermittelt werden, da hier zu jeder Zeit der aktuelle Messwert (in Form der resultierenden Spannung) vorliegt. Es genügt also einmal pro Sekunde die Spannung am analogen Eingang zu messen, um die aktuelle Windrichtung bestimmen zu können.

3.2.2 Verarbeitung der Messwerte

Im nächsten Schritt werden die Messwerte für die weitere Verarbeitung unterschiedlich interpretiert. Allgemein wird versucht, die Datenmenge pro Messung so gering wie möglich zu halten, deshalb werden auf dem Arduino direkt keine formatierten für den Menschen lesbare Werte ausgegeben. Es geht alleine darum, die Messwerte für den Transport vorzubereiten.

Die Werte für Anemometer und Regenmesser können ohne weiteres abgeschickt werden, während der Wert für die Himmelsrichtung (die abgelesene Spannung) noch komprimiert werden kann. Da jede Himmelsrichtung eindeutig einer Spannung zugeordnet werden kann, wird ein einfaches Array der möglichen Spannungswerte aus der Tabelle 3.1.3 erstellt, von dem am Ende nur die Position der Spannung im Array weitergegeben wird. Bei Ausgabe

der Daten im TTN kann dann letztendlich mit derselben Indizierung, die zugehörige Himmelsrichtung aus einer Tabelle bestimmt werden.

Um die Zuordnung der Spannungswerte immer gewährleisten zu können, verwendet man ein sortiertes Array der Spannungswerte (beginnend mit dem kleinsten Wert). Mit einem ungenau gemessenem Spannungswert kann somit einfach durch die sortierte Liste iteriert werden und an genau dem Wert im Array gestoppt werden, der dem aktuell gemessenen am nächsten kommt. (siehe Code-Ausschnitt)

3.2.3 Wetterdaten im Cache

LoRaWAN ist nicht darauf ausgerichtet, jede Sekunde neue Nachrichten zu kommunizieren. Es sollten immer so wenig Daten wie nötig versendet werden. Daher wäre es falsch jede Sekunde neue Wetterdaten zu senden. Der Arduino kann mehrere Messwerte zu einem Datensatz zusammenfassen und mehrere solcher Datensätze sammeln, bevor diese als Gesamtpaket versendet werden. Der Nutzer kann dabei selbst wählen, wie viele Messwerte zusammengefasst werden sollen (wie hoch die Auflösung sein soll) und wie viele dieser zusammengefassten Messwerte als ein Paket versendet werden sollen. Welche Parameter hier entscheidend sind, ist im Abschnitt 4 genauer beschrieben.

Wie sich der Aufbau der Daten im tatsächlich versendeten Paket zusammensetzt, wird im folgenden erläutert. Wie im oberen Abschnitt 3.2.2 beschrieben, liegen nach jeder Sekunde komprimierte Werte für die drei Sensoren vor. Jeder Messwert wird nun so verrechnet, dass die gewünschte zeitliche Auflösung der Werte erreicht wird. Folgende Werte sind dann in einem zusammengefassten Datensatz enthalten:

1. der Index der am meisten gemessenen **Windrichtung** [WV]
2. die summierte Anzahl der Schaltvorgänge des **Regenmessers** [R]
3. die durchschnittliche Anzahl an Schaltvorgängen des **Anemometers** [AN]
4. die im betrachteten Zeitraum höchste Anzahl an Schaltvorgängen des **Anemometers** [M_AN]

Diese Werte eines Datensatzes werden im Anschluss auf 3 Byte verteilt (siehe Code-Ausschnitt):

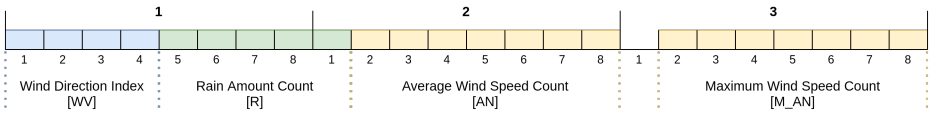


Abbildung 6: Aufbau eines Datensatzes

Die verschiedenen Wertebereiche lassen sich folgendermaßen ableiten:

1. **WV**: 4 Bit

   Die Windfahne misst 16 verschiedene Himmelsrichtungen. Übergeben wird nur der Index des sortierten Arrays der Spannungswerte (siehe 3.2.2), daher kommen hier nur Werte von 0 bis 15 zustande. Diese 16 Werte lassen sich exakt auf 4 Bit verteilen.

2. **R**: 5 Bit

   Mit 5 Bit stehen dem Regenmesser maximal 32 Schaltvorgänge pro Zeitraum zur Verfügung. Durch mehrere Tests hat sich ergeben, dass der Regenmesser innerhalb von 10 Sekunden maximal 27 Schaltvorgänge auslöst. Das entspricht etwa einer Regenmenge von 50 mm. Bei einer gewählten Auflösung größer als 10 Sekunden und stärkerem Regen kann mitunter kein exakter Wert mehr übermittelt werden. Die vorgesehene Größe von 5 Bit reicht dazu nicht aus. Mögliche Lösungsvorschläge für größere Zeiträume werden im Abschnitt **??** beleuchtet.

3. **AN**: 7 Bit

   Die durchschnittliche Windgeschwindigkeit im betrachteten Zeitraum darf maximal 128 Schaltvorgänge auslösen. Das entspricht einer Windgeschwindigkeit von etwa 300 km/h. Solche Geschwindigkeiten werden höchstens in Ausnahmefällen erreicht und sind zudem für die durchschnittliche Geschwindigkeit äußerst unwahrscheinlich.

4. **M_AN**: 7 Bit

   Die maximale Windgeschwindigkeit im betrachteten Zeitraum darf maximal 128 Schaltvorgänge auslösen. Das entspricht ebenfalls einer Windgeschwindigkeit von etwa 300 km/h.

Je nach dem, wie viele Datensätze in einem Paket versendet werden sollen, werden die Datensätze in einem Cache zwischengespeichert. Ist der Cache

voll, werden die Daten als ein Paket versendet. Der Cache wird im Anschluss geleert und kann nun wieder neu befüllt werden (siehe Code-Ausschnitt).

3.2.4 Senden der Daten ans TTN

Ist der Cache voll, gibt der Arduino dem LoRa-Shield über die verwendete LMIC-Library die Anweisung alle Daten im Cache zu versenden. Die Art und Weise, wie die Daten über LoRa versendet werden sollen, kann der Nutzer selbst bestimmen und wird im Abschnitt 4 erklärt.

3.3 TTN

Wie schon in 1.2.3 LoRaWAN Netzbetreiber genannt, nutzen wir das The Things Network. Zur Verwaltung der Applications (Verbund von Endknoten) und Gateways gibt es die Console.

In dieser lassen sich Applications und Gateways erstellen, aktuelle Daten anzeigen und Einstellungen konfigurieren. Besonders für Endknoten interessant sind der Payload Formatter, Integrations und ADR.

3.3.1 Payload Formatter

In diesem lassen sich Uplink und Downlink Nachrichten umwandeln. Bei der Übertragung über LoRa sollte die Payload stets so klein wie möglich sein, weshalb jedes versendete Bit genutzt werden sollte (siehe dazu auch Working with Bytes). Damit TTN die Payload (definiert in 3.2.3) interpretieren kann, müssen wir einen Payload Formatter setzen, wie in 4.4 beschrieben.

3.3.2 Integrations

Hier gibt es die Möglichkeit Trigger zu erstellen, die Daten verarbeiten und weiterleiten (bspw. Webhooks oder Storage Integrations für Up- und Downlink Nachrichten). Für eine Auflistung siehe Applications & Integrations.

3.3.3 Adaptive Data Rate - ADR

ADR ist ein Mechanismus für Endknoten der Spreading Factor, Datenübertragungsrate und Sendeleistung optimiert. Dafür werden die letzten 20 Uplink Nachrichten analysiert. Es ist standardmäßig für jeden Endknoten

aktiviert, sollte für mobile Endknoten jedoch deaktiviert werden (siehe 4.3 ADR deaktivieren).

3.4 Authentifizierungsmethoden

Um einen Endknoten im TTN zu authentifizieren, gibt es zwei Möglichkeiten. Beim Anlegen des Endknotens in der TTN Console muss entweder OTAA oder ABP gewählt werden, wobei ersteres vorzuziehen ist.

Einen wichtigen Aspekt spielt dabei im Bezug auf Sicherheit der Frame Counter. Mit jedem gesendeten Paket wird dieser inkrementiert und vom Netzwerk verifiziert, um Replay-Angriffe zu verhindern.

3.4.1 Over-the-Air Activation - OTAA

Bei der bevorzugten Variante OTAA wird ein Join-Verfahren beim Einschalten des Endknotens initialisiert, wobei Identifikationsinformationen (inklusive Frame Counter) und Schlüssel ausgetauscht werden. Ebenso werden die verfügbaren Frequenzen und der optimale Spreading Factor ermittelt.

Anmerkung: Um Replay-Angriffe des Join-Verfahrens zu verhindern, wird eine DevNonce vom Endknoten erstellt. Im Optimalfall speichert der Endknoten diese persistent und sendet jedesmal eine noch nicht genutzte Nonce. Der Arduino besitzt jdedoch keinen persistenten Speicher und kann die bereits genutzten DevNonce nicht einsehen. In neueren MAC Versionen muss eine Nonce jeweils größer als die vorige sein, siehe DevNonce is too small. Daher nutzen wir eine ältere MAC Version bei der der Server nur prüft ob die DevNonce bereits verwendet wurde.

Dennoch kann es im unwahrscheinlichen Fall vorkommen, dass der Arduino zweimal die gleiche DevNonce erstellt. In diesem Fall verweigert das TTN die Verbindung, ein Neustarten des Arduino sollte das Problem beheben.

3.4.2 Activation by Personalization - ABP

Bei ABP werden die vordefinierten Identifikationsinformationen, Spreading Factor und Schlüssel auf dem Gerät gespeichert. Falls ADR deaktiviert ist, bleibt der Spreading Factor konstant. Da es keine Join-Verfahren gibt, muss der Frame Counter, persistent abgespeichert werden. Der Arduino Uno besitzt aber keinen persistenten Speicher. Wenn dieser nun ausgeschaltet wird,

geht der Frame Counter verloren und beim erneuten Einschalten werden somit alle Pakete vom TTN verworfen.

Anmerkung: Der Arduino Uno besitzt ein EEPROM, welchen man nutzen könnte, um den Frame Counter zu speichern. Dieser hat jedoch eine begrenzte Lebensdauer von ungefähr 100000 Schreibzyklen.[2] Der Versuch den Frame Counter im RAM zu behalten und nur beim Ausschalten zu speichern funktioniert nicht, da bei einem Stromausfall die Operation nicht durchgeführt wird. Um das zu umgehen, kann mit jedem gesendeten Paket der EEPROM beschrieben werden. Dann ist dieser nach spätestens 100000 Paketen nicht mehr beschreibbar, was abhängig vom Sendeinterval schnell sein kann.

Daher empfehlen wir unbedingt OTAA zu verwenden. Soll dennoch ABP genutzt werden, so lässt sich in der TTN Console die Frame Counter Überprüfung deaktivieren, wie gezeigt in 4.2 ABP - Konfiguration, wodurch Replay-Angriffe nicht mehr verhindert werden.

Weiterhin gibt es noch ein Problem mit den nutzbaren Frequenzen, siehe 5.1 ABP - Frequenzbereich, was aber OTAA nicht betrifft.

4 Konfiguration & Verbindung der Wetterstation

Anmerkung: Es gibt auch einen Quelltext der die Daten lokal auswertet, ohne jegliche LoRa Implementationen (siehe `local_weather_station.ino`).

Weiterhin wird in dieser Dokumentation nicht darauf eingegangen, wie ein Gateway eingerichtet wird. Konsultieren Sie dafür die Betriebsanleitung des Herstellers.

Vorbedingung:

1. Installieren Sie die Arduino IDE und die notwendige Library nach `Installation.md`
2. Erstellen Sie einen TTN Account unter www.thethingsnetwork.org
3. Navigieren Sie zur Console

[2] Egger, Alexander. Arduino und der EEPROM. Retrieved 10 February 2022, from https://www.aeq-web.com/arduino-atmega-eeprom-read-write-details/

4. Wählen Sie <Go to applications>
5. Fügen Sie durch <Add application> eine Application hinzu
6. Wählen Sie <Add end device> und wechseln in den <Manually> Tab
7. Frequency Plan: Europe 863-870 MHz (SF9 for RX2 - recommended)
8. LoRaWan version: MAC V1.0.2
9. Regional Parameters version: PHY V1.0.2 REV A

Wie bereits in 3.4 Authentifizierungsmethoden erläutert, empfehlen wir OTAA gegenüber ABP.

4.1 OTAA - Konfiguration

10. DevEUI: Generate
11. AppEUI: Fill with zeros
12. AppKey: Generate
13. <Register end device >
14. Wechseln Sie zu `secret.template.h` und geben Sie die generierten Daten ein. Achten Sie ggf. auf das little-endian Format.
15. Benennen Sie `secret.template.h` in `secret.h` um
16. Öffnen Sie `config.h` und aktivieren Sie den OTAA Modus
17. Wählen Sie Ihre gewünschten Datenauflösung und die Größe der Payload. Betrachten Sie dazu das Beispiel und verifizieren Sie die Größe der Payload mit dem LoRaWAN airtime calculator.
18. Prüfen Sie, dass die Fair-Use-Policy nicht verletzt wird
19. Beachten Sie, dass bei OTAA der Spreading Factor automatisch gesetzt wird. Wählen Sie am besten Werte die mit SF12 noch unter die Fair-Use-Policy fallen.
20. Falls gewünscht, können Sie ADR deaktivieren.
21. Installieren Sie das Programm auf dem Arduino Uno.
22. Machen Sie weiter mit Payload Formatter setzen.

4.2 ABP - Konfiguration

10. <Show advanced activation, LoRaWAN class and cluster settings >
11. Activation mode: ABP
12. DevEUI: Generate
13. Device address: Generate
14. AppSKey: Generate
15. NwkSKey: Generate
16. <Register end device >
17. Nachfolgend deaktivieren Sie den Frame Counter, siehe Problematik in 3.4 Authentifizierungsmethoden.
18. Wechseln Sie in den <General settings> Tab
19. Wählen Sie bei Network layer <Expand>
20. Wählen Sie unten <Advanced MAC settings>
21. Setzen Sie das Häkchen in <Resets frame counters>
22. Falls gewünscht, entfernen Sie das Häkchen bei <Use adaptive data rate (ADR)>
23. <Save changes>
24. Wechseln Sie zu `secret.template.h` und geben Sie die generierten Daten ein.
25. Benennen Sie `secret.template.h` in `secret.h` um
26. Öffnen Sie `config.h` und aktivieren Sie den ABP Modus
27. Wählen Sie Ihre gewünschten Datenauflösung und die Größe der Payload. Betrachten Sie dazu das Beispiel und verifizieren Sie die Größe der Payload mit dem LoRaWAN airtime calculator.
28. Prüfen Sie, dass die Fair-Use-Policy nicht verletzt wird.
29. Falls gewünscht, können Sie ADR deaktivieren.
30. Installieren Sie das Programm auf dem Arduino Uno.

31. Machen Sie weiter mit Payload Formatter setzen.

4.3 ADR deaktivieren

1. Wechseln Sie in den <General settings> Tab im TTN
2. Wählen Sie bei Network layer <Expand>
3. Wählen Sie unten <Advanced MAC settings>
4. Entfernen Sie das Häkchen in <Use adaptive data rate (ADR)>
5. <Save changes>

4.4 Payload Formatter setzen

1. Wählen Sie den <Payload formatters> Tab
2. Uplink ist standardmäßig ausgewählt
3. Formatter type: Javascript
4. Kopieren Sie den Javascript Code von `payload_formatter.js` in das vorgesehene Feld
5. <Save changes>
6. Testen Sie den Formatter mit der Payload X
7. Starten Sie den Arduino. Falls ein Gateway in Reichweite ist, sollten die Daten nach einiger Zeit im Tab <Live data> erscheinen

5 Limitierungen

5.1 ABP - Frequenzbereich

In Europa ist der Downlink Frequenzbereich 868.1 - 868.8 MHz. Die Anwendung wechselt dabei automatisch zwischen den Kanälen, um eine gleichmäßige Auslastung zu erzielen.

Bei OTAA werden die nutzbaren Frequenzen beim Join-Verfahren vom Netzwerk übertragen. Weil ABP dieses nicht nutzt, müssen diese Frequenzen jedoch vordefiniert werden.

Wenn das Programm im ABP Modus läuft, funktionieren nur die Frequenzen: 868.1, 868.3, 868.5MHz. Wahrscheinlich setzen wir die zusätzlichen Frequenzen nicht korrekt. Im Codebeispiel der Library wurden diese leider deaktiviert.

Wenn wir die zusätzlichen Frequenzen hier freischalten, sendet das Gateway bei Paketen die nicht auf den 3 oben genannten Frequenzen gesendet wurden, keine Downlink Nachricht und quittiert dem Endknoten das Paket nicht.

6 Fazit

- für beide sehr interessant
- noch nie mit Arduino gearbeitet und wenig Mikrocontroller Erfahrung
- auch noch nie LoRa, ins Kalte Wasser geschmissen
- schwierige Programmierung mit Geräten die nicht häufig in der Community zumindest TTN genutzt werden
- Änderung auf TTNv3 hat es auch nicht besser gemacht
- viele Libraries, die aber alle nicht funktioniert haben, erst über Tipp der anderen Gruppe von angepasster Dragin Library erfahren
- Debugging ist schwierig bis unmöglich ohne eigenes Gateway, wahrscheinlich am Sinnvollsten einen RaspberryPI dafür zu nutzen
-