



**Hochschule**  
**Augsburg** University of  
Applied Sciences

**Bachelorthesis**

**Fakultät für  
Informatik**

Studienrichtung  
Informatik

**Ziegler Marco**  
**Analyse und praktische Anwendung eines LoRaWAN 1.1 Systems**

Erstprüfer: Prof. Dr. Hubert Högl  
Zweitprüfer: Prof. Dr.-Ing. Thorsten Schöler  
Abgabe der Arbeit am: 20.11.2020

Hochschule für angewandte  
Wissenschaften Augsburg  
University of Applied Sciences

An der Hochschule 1  
D-86161 Augsburg

Telefon +49 821 55 86-0  
Fax +49 821 55 86-3222  
[www.hs-augsburg.de](http://www.hs-augsburg.de)  
[info@hs-augsburg.de](mailto:info@hs-augsburg.de)

Fakultät für Informatik  
Telefon +49 821 5586-3450  
Fax +49 821 5586-3499

Verfasser der Bachelorthesis:  
Ziegler Marco  
Hauptstr. 28  
86551 Aichach  
Telefon +49 1590 4477742  
[marco.ziegler@hs-augsburg.de](mailto:marco.ziegler@hs-augsburg.de)

## Abstract

Das LoRaWAN Protokoll, welches zur energiesparenden Übertragung kleiner Datenmengen über große Distanzen verwendet wird, bietet einen Weg der Datenübertragung für verstreute und zahlreiche IoT Devices. Hierbei sind viele dieser End-Nodes auf jahrelange Laufzeit ausgelegt und somit teils relativ alt und somit von ihrer Softwareimplementation weit hinter dem momentan neuesten Versionen des Protokolls.

Eine umfangreiche Neuerung stellt hierbei LoRaWAN Version 1.1 dar, welches neben neuen Funktionalitäten wie MAC-Commands auch umfassenden Änderungen im generellen Verbindungsaufbau und somit Inkompatibilitäten zwischen unterschiedlich versionierten End-Nodes und deren Zielservern. Dies ist wiederum aufgrund der hohen Anzahl an älteren Devices in Benutzung welche durch dieses Problem keine Möglichkeit hätten, ihre Daten über neuere Server und Netzwerke zu übertragen, wodurch diese eventuell einzeln auf neue Versionen aktualisiert oder gar komplett ersetzt werden müssten.

Diese Inkompatibilität soll nach Spezifikation durch Fallbacks auf ältere Protokollversionen ausgehend von der höher versionierten Komponente behoben werden. Ob dieser spezifizierte Fallback aber in der Praxis ohne Probleme oder Anomalien funktioniert soll hier durch den Aufbau einer passenden LoRaWAN Testumgebung mit End-Nodes und Servern der verschiedenen Versionen überprüft werden.

Hierbei stellte sich heraus, dass der Fallback bei den verwendeten Servern und End-Nodes funktioniert und somit die Kompatibilität zwischen neuer versionierten Servern und End-Nodes mit älteren Devices und Netzwerken gewährleisten kann.

Da durch diese Implementierung auch ältere Komponente mit neuen Servern und End-Nodes kommunizieren können, müssen ältere End-Nodes oder bestehende Serverimplementationen nicht zwingend auf die neuere Version aktualisiert werden. Auch ältere, mit neueren Versionen eventuell inkompatible, End-Nodes können weiterhin verwendet werden und müssen von dem her nicht ausgetauscht werden.

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
<b>2</b>	<b>Grundlagen</b>	<b>3</b>
2.1	LoRaWAN Protokoll . . . . .	3
2.2	LoRaWAN Netzwerkaufbau . . . . .	3
2.2.1	End-Nodes . . . . .	4
2.2.2	Gateways . . . . .	5
2.2.3	Networking Server . . . . .	5
2.2.4	Applikations Server . . . . .	6
2.3	LoRaWAN 1.1 . . . . .	6
<b>3</b>	<b>Versuchsaufbau - physische Komponente</b>	<b>7</b>
3.1	Gateway . . . . .	7
3.1.1	Software . . . . .	7
3.2	End-Nodes . . . . .	8
3.2.1	Pycom LoPy 1.0 . . . . .	8
3.2.1.1	Software . . . . .	9
3.2.2	B-L072Z-LRWAN1 . . . . .	9
3.2.2.1	Software . . . . .	9
3.3	Networking/Application Server . . . . .	10
3.3.1	The Things Network . . . . .	10
3.3.2	Private LoRaWAN Server . . . . .	11

<b>4</b>	<b>Versuchsaufbau - Software</b>	<b>12</b>
4.1	Vorbereitung . . . . .	12
4.1.1	LoPy . . . . .	12
4.1.2	B-L072Z-LRWAN1 . . . . .	15
4.1.3	Gateway . . . . .	20
4.1.4	Networking Server . . . . .	23
4.2	Konfiguration . . . . .	24
4.2.1	The Things Network . . . . .	24
4.2.1.1	Gateways . . . . .	24
4.2.1.2	End-Nodes . . . . .	26
4.2.2	Private Server . . . . .	28
4.2.2.1	Gateways . . . . .	28
4.2.3	Gateway . . . . .	33
4.2.4	End-Nodes . . . . .	35
4.2.4.1	LoPy . . . . .	35
4.2.4.2	B-L072Z-LRWAN1 . . . . .	37
<b>5</b>	<b>Verhaltensanalyse</b>	<b>39</b>
<b>6</b>	<b>Zusammenfassung</b>	<b>43</b>

# Kapitel 1

## Einleitung

Die Anzahl von verwendeten IoT-Geräten hat in den letzten Jahren stets starken Zuwachs erlebt. Diese Beliebtheit treibt stets Weiterentwicklungen sowohl in Hardware als auch in Software um IoT-Implementationen mehr und mehr Funktionalitäten zu ermöglichen. Hierzu zählen auch Kommunikationsprotokolle wie Sigfox, NB-IoT oder LoRaWAN.

LoRaWAN steht hierbei für \*Long Range Wide Area Network\* und stellt eine open-source Lösung für IoT Kommunikation über große Distanzen in Verbindung mit geringem Energieverbrauch und einfacher Einbindung neuer Geräte in das Netzwerk dar. Hierbei kommunizieren die Endgeräte über Radio-Transceiver mit einem Gateway welches wiederum über einen Internetanschluss mit Applikationsservern kommunizieren kann. Diese Server können folglich die Daten des Endgerätes auswerten und gegebenenfalls über ein Gateway eine Antwort an das Endgerät senden.

Aufgrund der einheitlichen Implementation des LoRaWAN Protokolls war der Plug-and-Play Ansatz des gesamten Netzwerks stets gut durchsetzbar und selbst Weiterentwicklungen des LoRaWAN Protokolls auf höhere Versionen soll laut Spezifikation ohne Probleme zusammen mit vorherigen Versionen arbeiten können.

Doch ist die Etablierung verschiedener Versionen wirklich komplett ohne Probleme möglich? Wie verhält sich das gesamte Netzwerk wenn verschiedene Endgeräte und Gateways mit verschiedenen Versionen implementiert wurden? Ergeben sich durch die Versionsunterschiede Probleme oder Anomalien welche bei Geräten mit gleicher Version nicht auftreten? Diese Fragen sollen hier näher betrachtet und analysiert werden.

Hierzu wird ein Gateway und zwei Endgeräte mit verschiedenen Versionen eingerichtet auf welchen folglich diverse LoRaWAN Funktionalitäten implementiert

werden. Bei voller Kompatibilität sollten die beiden Geräte gleiche Ergebnisse liefern. Sollten sich Unterschiede feststellen, so ist dies ein Indiz für eventuelle Kompatilitätsprobleme zwischen den Protokollversionen.

# Kapitel 2

## Grundlagen

Dieses Kapitel soll grundlegende Informationen zu LoRaWAN wiedergeben. Fokus wird hierbei auf den Generellen Aufbau des Netzwerks und die generellen Neuerungen in LoRaWAN v1.1x gelegt werden.

### 2.1 LoRaWAN Protokoll

LoRaWAN ist eine Implementierung welche LoRa Technologie verwendet, um ein IoT Netzwerk zu realisieren. LoRa, ein Akronym für "Long Range", ist hierbei eine Technologie bei der Daten über Radiotransceiver über lange Strecken mit geringem Energieaufwand übertragen werden können.

LoRaWAN selbst ist eine Netzwerkarchitektur welche wiederum in vier eigene Schichten aufgeteilt ist. Diese Schichten sind die End-Nodes, Gateways, Networking Server und folglich die Applikations Server . LoRa selbst wird hierbei zwischen End-Nodes und Gateways eingesetzt und ermöglicht es, End-Nodes "plug-and-play" in das gesamte Netzwerk einzusetzen. Kommunikation von den Gateways aufwärts wird über eine normale Internetverbindung umgesetzt.

### 2.2 LoRaWAN Netzwerkaufbau

Die allgemeine Architektur [15] eines LoRaWAN Netzwerks besteht aus 4 Komponenten. Diese Komponente sind:

- End-Nodes welche die eigentlichen IoT-Geräte und Sensoren darstellen.

- Gateways die Daten von den End-Nodes empfangen und über eine Internet-Verbindung an die networking Server weitergeben. Die Gateways sind auch in der Lage Daten wieder zurück an die End-Nodes zu senden.
- Networking Server welche die Daten der Gateways sammeln und an den Applikations Server weiter leiten. Diese Server sorgen auch dafür, dass keine Duplikate von mehreren Gateways an die Applikationen übertragen werden.
- Applikations Server auf denen die eigentliche Verarbeitung der End-Node Daten stattfindet. Die Applikation kann der End-Node über die Networking Server und folglich den Gateways auch Antworten an die End-Nodes senden.

### 2.2.1 End-Nodes

End-Nodes sind IoT Sensor oder Steuerelemente welche ihre Daten über das LoRaWAN Netzwerk übertragen können bzw. über dieses angesteuert werden können. Realisiert sind diese oft durch Mikrocontroller wie zum Beispiel einem Py-board mit Erweiterungskarten oder direkt als standalone Device. Die End-Nodes sollten hierbei stets energieeffizient sein, da diese normalerweise nicht an eine feste Stromzufuhr angeschlossen sind und über Batterien betrieben werden.

Dies erlaubt es die End-Nodes an einem beliebigen Ort ohne viel Installationsaufwand zu installieren wodurch es auch einfach ist neue Geräte hinzuzufügen oder nicht mehr benötigte Geräte wieder zu entfernen. Sobald eine End-Node eingeschaltet wird, sollte die Batterie je nach Aufbau im Durchschnitt 1-4 Jahre halten [16], wobei längere Laufzeiten durchaus möglich sind. Datentransfer erfolgt über das LoRaWAN Netzwerk mit Hilfe eines Gateways, dies erfordert natürlich ein funktionsfähiges Gateway in Reichweite der End-Node worauf bei der Installation der End-Node geachtet werden sollte.

End-Nodes selbst sind nochmals in drei verschiedene Klassen [2] aufgeteilt welche für verschiedene Anwendungszwecke geeignet sind.

- End-Nodes der Klasse A empfangen Daten nur in festen Zeitabschnitten folgend auf das Senden von Daten. Generell achtet die End-Node nur für zwei Zeitabschnitte folgend dem Senden von Daten auf eine eventuell eingehende Antwort. Sollte in diesen zwei Abschnitten keine Antwort eintreffen, so gibt es erst nach der nächsten Datensendung ausgehend von der End-Node wieder die Möglichkeit, Daten an die End-Node zu senden.



- End-Nodes der Klasse B bieten neben den Empfangsfenstern welche auch von Klasse A Nodes geboten werden weitere, feste Empfangsfenster welche zu vorher festgelegten Zeiten geöffnet werden.
- End-Nodes der Klasse C stehen permanent für Datenempfang offen, solange der Transceiver nicht gerade mit dem Senden von Daten beschäftigt ist. Hierbei ist zu beachten, dass der Energieverbrauch durch die stetige Empfangsbereitschaft stark zunimmt.

### 2.2.2 Gateways

Gateways [4] sind spezielle Mikrocomputer welche sowohl mit einem LoRaWAN fähigen Transceiver als auch einem Internetanschluss ausgestattet sind. Dies erlaubt es den Gateways als Verbindungsglied zwischen End-Nodes und dem Internet zu dienen.

Sobald eine End-Node eine LoRa-Nachricht über seinen Transceiver aussendet, so nehmen alle aktiven Gateways in Reichweite diese Nachricht entgegen und geben diese über ihre Internetverbindung an die Networking Server weiter. Sollte eine Antwort von einem Server für die End-Node verfügbar sein, so wird diese an ein Gateway in Reichweite dieser End-Node weitergegeben und über den Transceiver weitergereicht.

Gateways sind grundsätzlich stationär da diese viele Daten übertragen und auch stetig empfangen müssen und somit mehr Energie als eine End-Node benötigen. Dies hat zur Folge, dass Stromversorgung über Akkus oder Batterien nicht voll geeignet sind.

### 2.2.3 Networking Server

Die Networking Server dienen dem Filtern von Gateway-Nachrichten und dem Routing von Nachrichten zwischen Applikations Servern und Gateways.

Networking Server nehmen die Nachrichten von Gateways entgegen, welche von einer End-Node ausgesendet wurden. Da mehrere Gateways die Nachricht einer End-Node aufgenommen und folglich weiter gesendet haben könnten, filtern die Networking Server duplizierte Nachrichten heraus. Folglich werden die Daten an den zutreffenden Applikations Server weiter geleitet. Sollte die Applikation eine Antwort für die End-Node haben, so wird diese erst an den Networking Server gesendet welche folglich ein passendes Gateway in Reichweite der End-Node auswählt und die Nachricht über dieses an die richtige End-Node weitergibt.

## 2.2.4 Applikations Server

Der Applikations Server hält die eigentliche Applikation, z.B. ein Programm, welcher die Daten einer End-Node auswertet oder diese generell steuert.

Je nach Art der End-Node werden hier eventuell gesammelte Daten gesammelt und ausgewertet. Diese Daten können dann verwendet werden um potentielle Aktionen auszulösen. So kann z.B. bei der Feststellung von hohen Temperaturen eine Klimaanlage eingeschaltet werden.

## 2.3 LoRaWAN 1.1

Neuerungen in LoRaWAN 1.1 [14] beinhalten neben Erweiterungen wie direkten Support für Class B Devices, neuen MAC Commands oder handover Roaming auch Änderungen in der Join-Sequenz und den verwendeten Sicherheitsprotokollen.

Die Erweiterungen sind hierbei für die Kompatibilität eher weniger ausschlagend, aber die abgeänderten Kommunikationsabläufe haben einen direkten Einfluss auf die Funktionsweise von End-Nodes.

So benötigen LoRaWAN 1.1 Devices als Beispiel nun persistenten Speicher um Tokens und Counter, welche für die neue Join-Sequenz benötigt werden, abzuspeichern. Dies bedeutet, dass ältere End-Nodes, welche auf LoRaWAN 1.0 basieren, ohne persistenten Speicher nicht mal über Softwareupdates die neue Join-Sequenz verwenden können.

Sollte ein Networking Server also nur LoRaWAN 1.1 unterstützen, so kann eine ältere End-Node diesen Server nicht verwenden. Sollte eine neuere, 1.1 fähige, End-Node aber einen älteren 1.0 Server ansprechen, so wird die Version der Join-Sequenz auf 1.0 herabgesetzt um Kompatibilität zu ermöglichen.

# Kapitel 3

## Versuchsaufbau - physische Komponente

Die für die Analyse verwendete Versuchsaufbau besteht im gesamten aus einem Gateway und zwei verschiedenen End-Nodes. Folglich werden die physikalischen Geräte und deren zugehörige Softwareimplementationen näher erklärt.

### 3.1 Gateway

Das Gateway besteht aus einem Raspberry Pi welches um ein iC880A SPI LoRaWAN Concentratorboard erweitert und folglich in ein passendes Gehäuse mit Antenne eingebaut wurde.

Vorteil des iC880A Concentratorboards ist vor allem die hohe Performanz in Verbindung mit hoher Reichweite. So ist der Concentrator dazu im Stande bis zu 8 unterschiedliche Signale in einem Umkreis von 1.3km [3] in Stadtgebieten einzufangen und zu bearbeiten. Ein Raspberry Pi als Hostsystem für den iC880A bietet genügen Leistung um mehrere tausend End-Devices über ein Gateway mit dem LoRaWAN Netzwerk zu verbinden.

#### 3.1.1 Software

Als Software für den Betrieb des Gateways wird die ic880a-gateway-Software [10] verwendet. Diese ist speziell für den iC880A Concentrator in Kombination mit einem Raspberry Pi entwickelt und erlaubt einfache Konfiguration über json Files.

Die einfache Konfiguration ist hilfreich beim Wechsel der Ziel-Networking Server. Dies ist wiederum benötigt um das Verhalten der End-Nodes bei Verbindung zu verschiedenen versionierten Servern zu testen.

## 3.2 End-Nodes

Die beiden End-Nodes laufen jeweils auf einer Version des LoRaWAN Protokolls. Hierbei verwendet der Pycom LoPy 1.0 und das B-L072Z-LRWAN1 Version 1.1x um die jeweiligen Tests auf verschiedenen Versionen durchlaufen zu können.

### 3.2.1 Pycom LoPy 1.0

Das erste End-Device besteht aus einem Pycom LoPy 1.0 in Verbindung mit einem Pycom Expansion Board 2.1.

Abbildung 3.1: Pycom LoPy 1.0 auf einem Pycom Expansion Board 2.1



Diese Verbindung bietet ein kleines, energieeffizientes und dennoch leistungsfähiges End-Device für die Nutzung von LoRaWAN. Im Grunde bietet das LoPy ein kompaktes Board welches alle Grundlegenden Voraussetzungen für die Nutzung von LoRa bereitstellt. So bietet das LoPy unter Anderem einen festen Steckplatz für die für LoRa benötigte Antenne, vollen MicroPython Support für die Programmierung der Device-Software in Python mit Support für Multithreading und WiFi für Zugang zur Boardsoftware um Programmänderungen auch auf Distanz vornehmen zu können. Das Expansion Board dient zur Erweiterung des LoPy um Batterie und USB-Power, einer Status-LED für eventuelle Statusanzeigen und einem MicroSD Kartenslot für Speichererweiterung.

### 3.2.1.1 Software

Das LoPy wird direkt über Pythoncode programmiert und bietet direkt alle benötigten Packages um eine End-Device Implementation auf Version 1.0x einzurichten. Es werden also keine spezifischen Implementationen eines LoRaWAN Stacks oder andere externe Driver benötigt.

Simple Beispielimplementationen [13] können der Dokumentation des LoPy entnommen und folglich für den spezifischen Anwendungszweck angepasst werden.

## 3.2.2 B-L072Z-LRWAN1

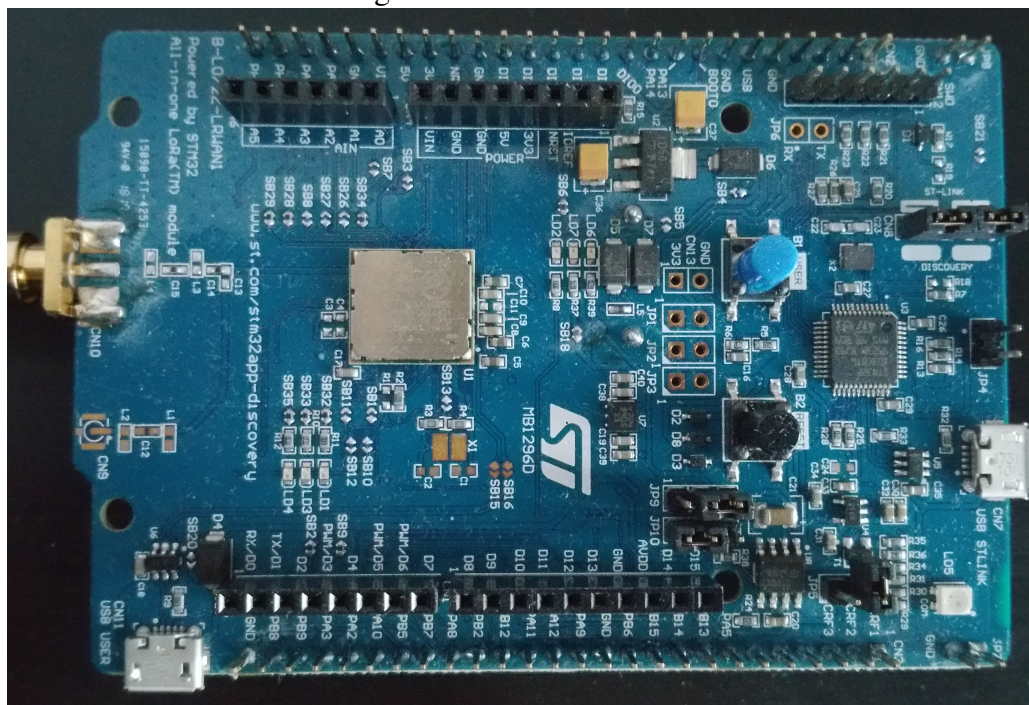
Das zweite End-Device ist das komplette, standalone Board B-L072Z-LRWAN1 von STMicroelectronics.

Dieses Board stellt ein komplettes LoRaWAN 1.1 kompatibles Gesamtpaket bereit welches alle drei Device Klassen für LoRaWAN End-Devices unterstützt. Programmierung erfolgt in C wobei die Programme auf einem Hostsystem kompiliert werden. Die Binaries können folglich über einen MicroUSB Port manuell oder direkt über eine Toolchain auf das B-L072Z-LRWAN1 eingespielt werden. Die aufgespielte Binary wird nach dem Installationsprozess direkt ausgeführt, wodurch kein stetiger Neustart des Boards von Nöten ist. Mehrere LEDs auf dem Board können hierbei für schnelle, visuelle Statusanzeigen verwendet werden.

### 3.2.2.1 Software

Als Basisimplementation der Software für das B-L072Z-LRWAN1 wird der LoRaMac-node End-Device Stack [7] verwendet.

Abbildung 3.2: B-L072Z-LRWAN1 Board



Dieser bietet eine End-Device Implementation auf Version 1.1x für verschiedene Microcontroller. Des weiteren kann der Stack auch Version 1.0x verwenden und bietet somit hohe Flexibilität für eventuelle Kompatibilitätstests.

### 3.3 Networking/Application Server

Zum Testen des Verhaltens der beiden End-Devices bei Verbindung zu verschiedenen LoRaWAN Versionen werden zwei verschiedene Networking Server benötigt. Hierbei bieten sich die Server des The Things Network (TTN) als Implementation von Version 1.1 Servern, und eine private Installation des Private LoRaWAN Server für einen 1.0x Servers an.

#### 3.3.1 The Things Network

TTN [19] ist ein öffentlich zugängliches LoRaWAN-Netzwerk bei welchem man Gateways und End-Devices anmelden und folglich über die TTN Server verwenden kann. Hierbei sind die Gateways anderer Benutzer für alle frei zugänglich wodurch eine grosse Fläche mit LoRaWAN Connectivity abgedeckt werden kann.

Da die TTN Networking Server die Version 1.1 mit 1.0 Fallback unterstützen und die Verwendung dieser ohne weitere Anforderungen möglich ist, bietet sich TTN perfekt für Tests welche eine volle 1.1 Implementation voraussetzen.

### **3.3.2 Private LoRaWAN Server**

Der Private LoRaWAN Server von Petr Gotthard auf Github [6] ist eine Version 1.0.3 Implementation einer LoRaWAN Networking und Application Server Kombination.

Dies erlaubt es einen lokalen LoRaWAN Server auf Version 1.0x zu betreiben welcher sich damit gut als Testumgebung für das Verhalten der End-Devices in Verbindung mit einem 1.0x Networking Servers zu testen. Installation und Betrieb des Servers erfolgt hierbei auf einem lokalen PC welcher Debian 10 verwendet.

# Kapitel 4

## Versuchsaufbau - Software

Dieses Kapitel soll einen Einblick in den Installations und Aufbauprozess eines vollwertigen LoRaWAN Netzwerks mit End-Nodes, Gateway und Servern verschaffen. Hierbei werden sowohl die öffentliche Variation über TTN als auch die private Implementierung über einen privaten Netzwerkservers betrachtet. Alle Installationen werden auf einem Debian 10 System durchgeführt.

### 4.1 Vorbereitung

Als Vorbereitung auf das Arbeiten mit dem aufzubauendem Netzwerk soll hier die grundlegende Installation von passenden IDEs bzw. Aufsetzen der einzelnen Geräte angesprochen werden. Ziel ist es hierbei, alles auf einen einfachen und funktionsfähiges Stand zu bringen auf welchem folglich weiter aufgebaut werden kann.

#### 4.1.1 LoPy

Das Arbeiten mit dem LoPy erfordert direkten Zugang zum Device entweder über WiFi oder USB. In diesem Beispiel wird die USB Variante aufgrund der simpleren Arbeitsweise gewählt.

Als IDE/Editor zum programmieren des Codes und folglichem Upload auf das LoPy existieren fertige Packages für Atom [1] und Visual Studio Code (VSC) [21]. Der einzige Unterschied der beiden Optionen liegt in der Präferenz des Entwicklers und folgliches Beispiel zeigt den Prozess für den Atom Editor wobei die Installation für VSC analog abläuft.



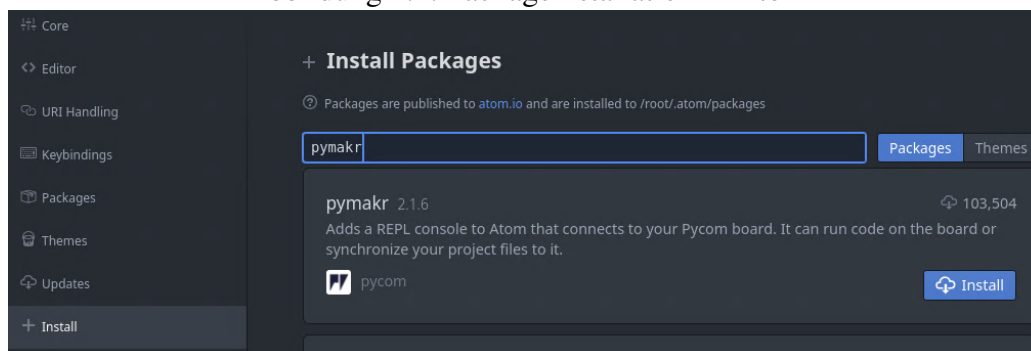
Zuerst muss Atom selbst installiert werden. Dazu kann man entweder das Package Repository von Atom zu seinem apt hinzufügen, die deb Packages manuell installieren oder den source Code selbst kompilieren. Für eine Installation ohne Hinzufügen neuer Repositories wird hier die manuelle Variante gewählt. Die deb File für die Installation kann über die da Git von Atom [12] heruntergeladen werden.

Das dpkg Command installiert das vorherig heruntergeladene Package. Sollten Dependencies für dieses fehlen, so können diese über apt-get nachgeladen werden.

```
1 user@Hostsystem:~# sudo dpkg -i atom-amd64.deb
2 user@Hostsystem:~# sudo apt-get -f install
```

Nun kann Atom gestartet werden. In Atom selbst muss nun das Pymakr Package installiert werden. Hierzu kann man einfach über “Edit” und folglich “Preferences” in das Settingsmenü von Atom gelangen. In diesem Menü kann man durch “Install” Packages suchen und folglich installieren.

Abbildung 4.1: Packageinstallation in Atom



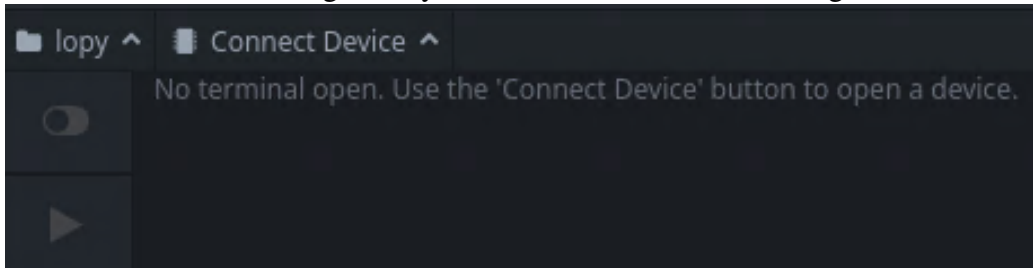
Alternativ zur Installation über den Packagemanager in Atom selbst, kann Pymakr auch über die Konsole mit dem atom package manager (apm) installiert werden. In diesem Fall ist pymark der Name des Packages und @2.1.6 gibt die gewünschte Version an, wodurch auch ältere Versionen installiert werden können.

```
1 user@Hostsystem:~# apm install pymakr@2.1.6
```

Nach der Installation empfiehlt es sich, Atom neu zu starten.

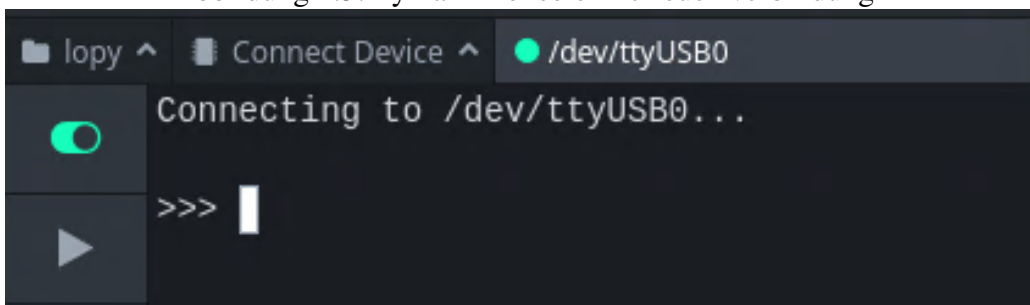
Nun sollte sich im unteren Teil von Atom eine Pymakr Konsole befinden, welche noch keine Verbindung zu einem Device aufgebaut hat.

Abbildung 4.2: Pymakr-Konsole ohne Verbindung



Um eine Verbindung zum LoPy herzustellen, muss dieses über USB an das Hostsystem angeschlossen werden. Folglich sollte Atom und Pymakr automatisch eine Verbindung zu diesem aufbauen.

Abbildung 4.3: Pymakr-Konsole mit neuer Verbindung



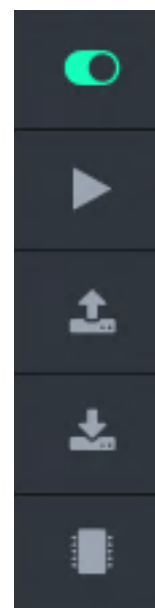
Die folgende Python-Konsole ist eine direkte Verbindung zum LoPy und kann zum Ausführen von normalem Python-Code verwendet werden. Die Konsole selbst bietet auch weitere, wichtige Funktionalitäten.

Der erste Button gibt den Status der Verbindung zum Device an. Sollte man die Verbindung unterbrechen bzw. aufbauen wollen, kann man dies über diesen Button.

Der zweite Button führt die momentan im Editor gewählte File auf dem Device aus.

Die folgenden zwei Buttons sind jeweils für Datei Up- und Download auf bzw. von dem Device. Dies erlaubt es Dateien auf das Device hochzuladen oder den momentan dort befindlichen Code auf das Hostsystem herunterzuladen.

Der letzte Button gibt Informationen wie Devicetyp, MAC oder Memory des momentan verbundenen Devices aus.



### 4.1.2 B-L072Z-LRWAN1

Programmierung des B-L072Z-LRWAN1 erfolgt über USB und direktem Flashen des Speichers mit einem vorkompilierten Programm. Hierzu kann als Entwicklungsumgebung Visual Studio Code in Verbindung mit einer passenden Toolchain verwendet werden.

Als Software-Stack wird die LoRaMac-node[7] Implementierung von Lora-net verwendet. Diese benötigt zum kompilieren CMake mit passendem gcc-arm Compiler welche beide über apt installiert werden können, sofern nicht bereits vorhanden. Des weiteren wird für das Flashen des Boards noch openocd[17] in Verbindung mit GDB [5] empfohlen. Hierbei ist zu beachten, dass eine ARM Version des GDB (arm-none-eabi-gdb, gdb-multiarch) verwendet werden muss. Alternative kann auch stlink-org[9] verwendet werden. Installationsanleitungen für Windows und iOS können auf der GitHub Seite von stlink-org gefunden werden.

```
1 user@Hostsystem:~# sudo apt-get install cmake
2 user@Hostsystem:~# sudo apt-get install gcc-arm-none-
  ↳ eabi
3 user@Hostsystem:~# sudo apt-get install openocd
```

Die Installation von VSC erfolgt hierbei ähnlich wie bei Atom. Zuerst muss die deb File ueber die VSC-Downloadpage [22] heruntergeladen und folglich ueber dpkg installiert werden. Alternativ kann VSC auch über snap direkt heruntergeladen werden, sofern der Snap Store[23] installiert ist.

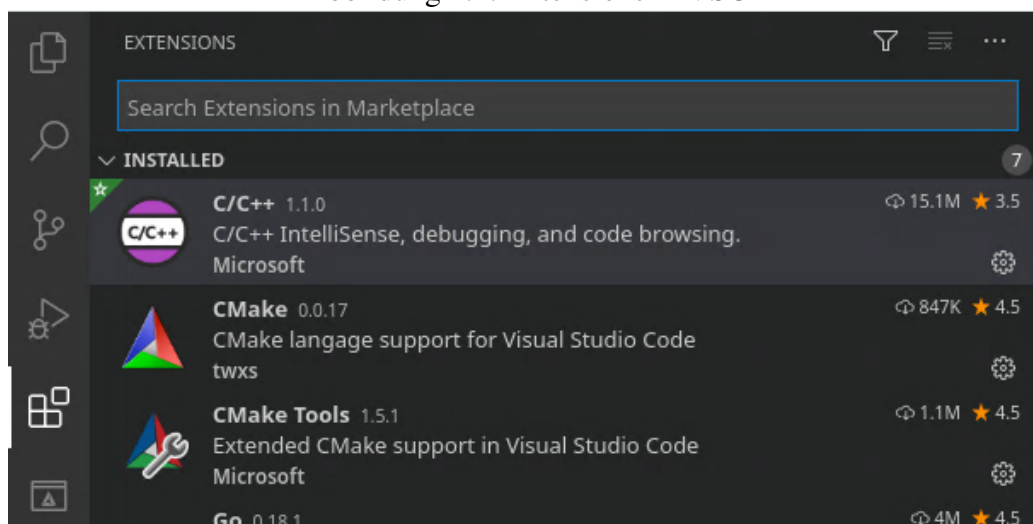
```

1 user@Hostsystem:~# sudo dpkg -i code_1.51.deb
2 user@Hostsystem:~# sudo apt-get -f install
3 #Oder über snap
4 user@Hostsystem:~# sudo snap install --classic code

```

Da der für die Programmierung verwendete Software-Stack in C geschrieben ist, sollten passende Packages für das Arbeit mit C und CMake installiert werden. Hierzu links in das Extensions Menü wechseln und neben der C/C++ Extension für die Verwendung von CMake noch CMake und Cmake Tools zu VSC hinzufügen.

Abbildung 4.4: Extensions in VSC



Die Einrichtung der Entwicklungsumgebung ist somit abgeschlossen und der LoRaMac-node[7] Stack kann nun über das zugehörige Git Repository heruntergeladen werden.

Zur Initialisierung von CMake muss im Root-Verzeichnis des heruntergeladenen Verzeichnis ein "build" Ordner erstellt, und in diesem die Toolchain für make definiert werden.

```

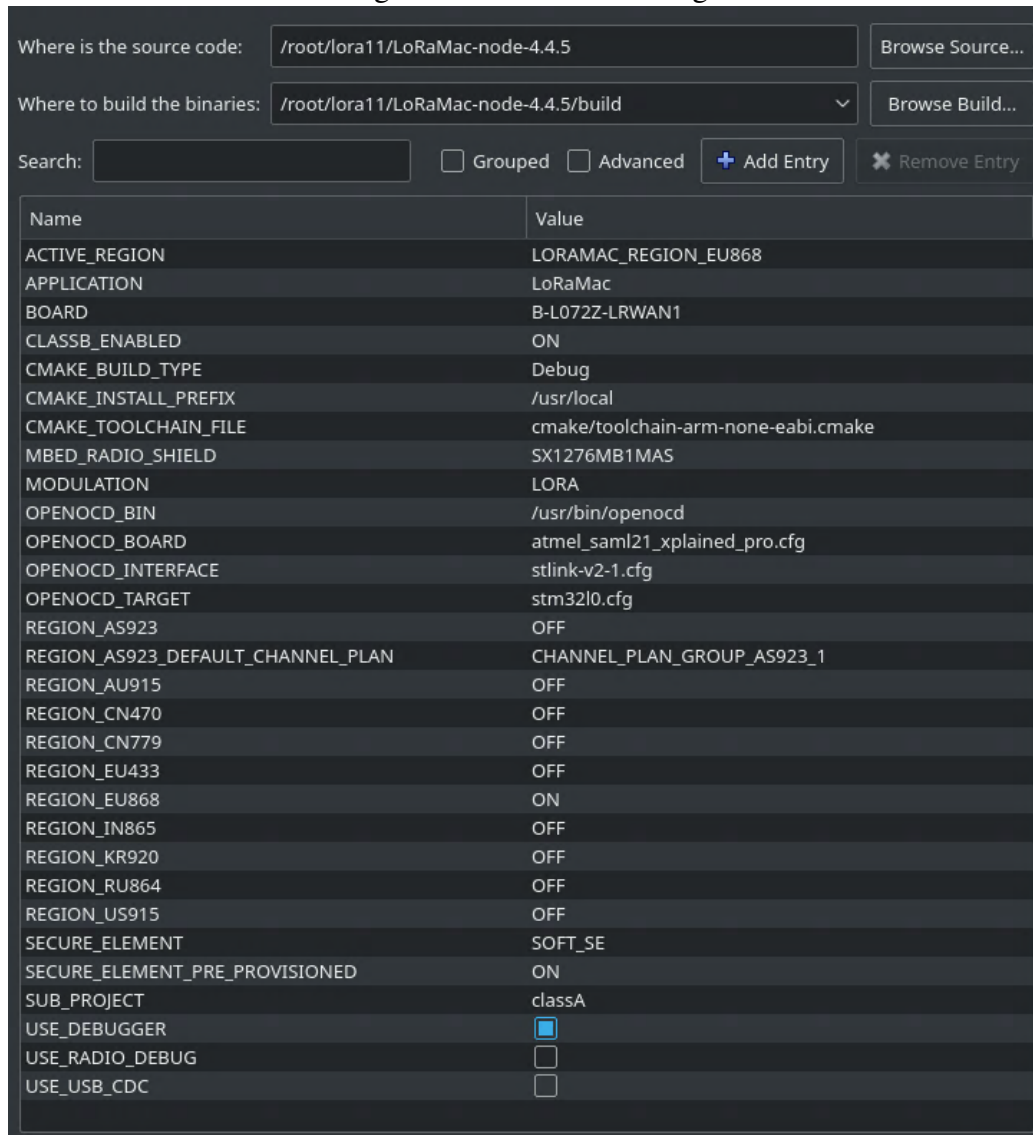
1 user@Hostsystem:~/LoRaMac-node-4.4.5/build#
2 cmake -DCMAKE_TOOLCHAIN_FILE="cmake/toolchain-arm-none
   ↪ -eabi.cmake" ..

```

Der Buildprozess von Cmake selbst kann über spezifische Flags[8] weiter konfiguriert werden. Einfacher und übersichtlicher kann diese Konfiguration aber durch die cmake-gui abgeschlossen werden.

Hierzu können über "Configure" die verfügbaren Flags mit deren Optionen geladen werden. Die Values der Flags kann man folglich über ein Dropdown Menü oder durch eine Tickbox auswählen und über den "Generate" Button als Konfiguration für CMake verwendet werden.

Abbildung 4.5: CMake GUI Konfiguration



Nun kann das Projekt in VSC geöffnet werden und die CMake Extensions von VSC starten eine Selbstkonfiguration für die Verwendung der soeben erstellten CMake Konfigurationsdateien und am unteren Rand von VSC wird der Status von

CMake angezeigt. Sobald dieser auf entweder Debug oder Release eingestellt ist, kann das Projekt über den Build Button kompiliert werden.

Abbildung 4.6: CMake Build in VSC

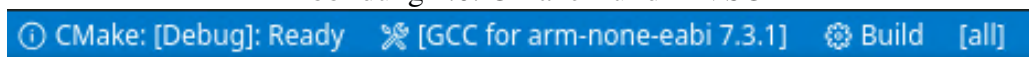


Abbildung 4.7: Buildausgabe in VSC

```
[build] Scanning dependencies of target LoRaMac-classA
[build] [ 97%] Building C object src/apps/LoRaMac/CMakeFiles/LoRaMac-classA.dir/common/NvmCtxMgmt.c.obj
[build] [ 98%] Building C object src/apps/LoRaMac/CMakeFiles/LoRaMac-classA.dir/classA/B-L072Z-LRWAN1/main.c.obj
[build] [100%] Linking C executable LoRaMac-classA
[build]   text      data |  bss      dec      hex filename
[build]   82864      744      6540      90148      16024 LoRaMac-classA
[build] [100%] Built target LoRaMac-classA
[build] Scanning dependencies of target LoRaMac-classA.hex
[build] Scanning dependencies of target LoRaMac-classA.bin
[build] [100%] Built target LoRaMac-classA.bin
[build] [100%] Built target LoRaMac-classA.hex
[build] Build finished with exit code 0
```

Die durch den Build erzeugte Binary kann nun direkt über openocd oder zusammen mit GDB auf das über USB verbundene Board programmiert werden. Hierzu muss openocd mit dem Board, und GDB folglich über den automatisch mit gestarteten openocd-Server verbunden werden. Hierbei empfiehlt es sich, die beiden Programme in jeweil einem eigenem Terminal zu starten.

Starten des openocd-Servers:

```
1 user@Hostsystem:~/loral1/LoRaMac-node-4.4.5/build/src/
   ↳ apps/LoRaMac# openocd -f interface/stlink-v2-1.
   ↳ cfg -f target/stm3210.cfg
2 Open On-Chip Debugger 0.10.0
3 ...
4 Info : using stlink api v2
5 Info : Target voltage: 3.263684
6 Info : stm3210.cpu: hardware has 4 breakpoints, 2
   ↳ watchpoints
```

Vor der Verbindung von GDB zu openocd sollte die zu flashende Datei ausgewählt werden. Folglich kann eine Verbindung über den Standardport von openocd, 3333, aufgebaut werden. Vor dem Beschreiben des Boards mit der gewählten Datei muss laufender Code auf dem Board noch angehalten werden. Über "load" wird das Board dann programmiert woraufhin der Code ausgeführt und live auf dem Board debugged werden kann.

```
1 user@Hostsystem:~/loral1/LoRaMac-node-4.4.5/build/src/  
  ↪ apps/LoRaMac# gdb-multiarch  
2 GNU gdb (Debian 8.2.1-2+b3) 8.2.1  
3 ...  
4 (gdb) file ./build/src/apps/LoRaMac/LoRaMac-classA  
5 Reading symbols from ./build/src/apps/LoRaMac/LoRaMac-  
  ↪ classA...done.  
6 (gdb) target extended-remote localhost:3333  
7 Remote debugging using localhost:3333  
8 (gdb) monitor reset halt  
9 target halted due to debug-request, current mode:  
  ↪ Thread  
10 xPSR: 0xf1000000 pc: 0x0800c400 msp: 0x20005000  
11 (gdb) load  
12 Loading section .text, size 0x143a8 lma 0x8000000  
13 Loading section .ARM.exidx, size 0x8 lma 0x80143a8  
14 Loading section .data, size 0x2e8 lma 0x80143b0  
15 Start address 0x800c400, load size 83608  
16 Transfer rate: 7 KB/sec, 10451 bytes/write.  
17 (gdb) continue  
18 Continuing.
```

Openocd kann auch ohne den GDB direkt für das Flashen verwendet werden. Hierzu benötigt openocd zu dem Board passende Konfigurationsdateien, welche bei der Installation bereits mitgeliefert werden, die auszuführenden Commands im Debugger selbst und den Pfad zur Binary mit Boardspezifischer Startadresse des zu überschreibenden Speichers.

Das B-L072Z-LRWAN1 benötigt hierbei als Konfigurationsdateien stlink-v2-1.cfg und stm3210.cfg und als Speicheradresse 0x08000000.

```
1 user@Hostsystem:~/loral1/LoRaMac-node-4.4.5/build/src/  
  ↪ apps/LoRaMac# openocd -f interface/stlink-v2-1.  
  ↪ cfg -f target/stm3210.cfg -c "init" -c "reset  
  ↪ halt" -c "flash probe 0" -c "flash write_image  
  ↪ erase ClassA.bin 0x08000000" -c "reset run" -c "  
  ↪ shutdown"  
2 Open On-Chip Debugger 0.10.0  
3 ...  
4 Info : clock speed 240 kHz  
5 Info : STLINK v2 JTAG v37 API v2 SWIM v26 VID 0x0483
```

```

    ↪ PID 0x374B
6 Info : using stlink api v2
7 ...
8 Info : Device: STM32L0xx (Cat.5)
9 Info : STM32L flash has dual banks. Bank (0) size is
    ↪ 128kb, base address is 0x8000000
10 flash 'stm32lx' found at 0x08000000
11 auto erase enabled
12 ...
13 wrote 94208 bytes from file LoRaMac-periodic-uplink-
    ↪ lpp.bin in 19.566898s (4.702 KiB/s)
14 shutdown command invoked

```

Die Binary ist nun auf das Board programmiert worden und nach einem Neustart wird der Code auf diesem ausgeführt.

### 4.1.3 Gateway

In diesem Beispiel wird als Gateway ein Raspberry Pi zusammen mit einem DPI iC880A SPI LoRaWAN Concentratorboard verwendet. Es wird dazu angenommen, dass der Raspberry bereits über eine funktionsfähige Installation von Raspbian verfügt.

Eine passende Antenne sollte generell immer vor Start des Gateways angeschlossen werden, da sonst der Concentrator beschädigt werden könnte.

Zuerst muss eine SSH Verbindung zu dem Raspberry aufgebaut werden. Hierzu den Raspberry einschalten und über Ethernet an das lokale Netzwerk z.B. über einen Switch anschliessen. Folglich kann man über das Hostsystem die IP des Raspberry herausfinden und zu diesem eine Verbindung aufbauen.

Die IP kann aus dem ARP Table ausgelesen werden. Ältere Debian Distributionen verwenden hierbei noch den “arp” Command anstatt des neueren “ip” Commands.

```

1 #Für ältere Distributionen:
2 user@Hostsystem:~# arp -a
3 #Neuere Distributionen:
4 user@Hostsystem:~# ip -a

```

Die Ausgabe enthält IPs mit deren passenden MAC-Adressen. Eine der Ausgaben kann folglich so aussehen:

```

1 user@Hostsystem:~# ip nip -a

```



```
2 192.168.2.109 [...] b8:27:eb:38:5d:3f
```

Hier sehen wir die MAC-Adresse des Gateways, in diesem Falle b8:27:eb:38:5d:3f zusammen mit der Lokalen IP 192.168.2.109. Mit diesen Informationen kann man sich nun über ssh auf dem Gateway einloggen. Der Standardbenutzer auf Raspbian ist hierbei der User "pi" mit dem Passwort "raspberry".

```
1 user@Hostsystem:~# ssh pi@192.168.2.109
2 pi@192.168.2.109\'s_password:_raspberry
```

Nun müssen einige Einstellungen vorgenommen werden. Zuerst bietet es sich an, das Filesystem von read-only auf writable umzustellen um Änderungen an Systemdaten vornehmen oder neue Dateien anlegen zu können. Hierzu kann das Filesystem mit den rw (read/write) Flags remounted werden.

```
1 pi@LoRaGateway:~# sudo mount -o remount, rw /
```

Als nächstes sollte ein neuer User erstellt werden, auf welchem folglich das Gateway betrieben wird. Hier wird als Nutzernamen "gateway" gewählt. Folglich wird dieser Nutzer noch in die Sudogruppe eingetragen. Je nach Präferenz kann die Passwortabfrage für diesen Nutzer auch deaktiviert werden.

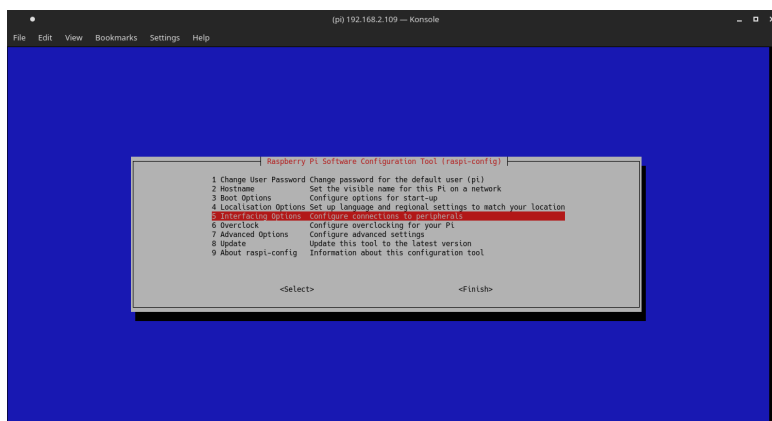
```
1 pi@LoRaGateway:~# sudo adduser gateway
2 pi@LoRaGateway:~# sudo adduser gateway sudo
3 #(Optional) Passwortabfrage deaktivieren
4 pi@LoRaGateway:~# sudo visudo (gateway=(ALL) NOPASSWD:
   ↪ ALL)
```

Bevor der Benutzer gewechselt wird muss SPI aktiviert werden. Dies ist über die Raspi-Config möglich. Das Konfigurationsmenü kann über den folgenden Command geöffnet werden.

```
1 pi@LoRaGateway:~# sudo raspi-config
```

Folglich öffnet sich das Menü in welchem über folgende Schritte SPI aktiviert werden kann:

Zuerst wählt man “Interfacing Options” aus.



Und in diesem Menü auf “SPI”.

Zuletzt “Yes” wählen und eine Bestätigung der Aktivierung und Nachfrage nach einem Neustart des Systems treten auf. Hier sollte das Gateway auch direkt neu gestartet werden. Nach dem Neustart kann man sich nun wieder über ssh mit dem neu angelegten Nutzer zu dem Gateway verbinden.

Download der Gateway-Software, zu finden auf <https://github.com/ttn-zh/ic880a-gateway>, kann entweder vom Hostsystem aus erfolgen, wobei die Dateien über beispielsweise scp auf das Gateway übertragen werden können oder direkt vom Gateway aus über git. Nach dem Download der Software, kann diese über das beiliegende “install.sh” Skript installiert werden. Hierbei ist darauf zu achten, die `βpi` flag mitzugeben.

```
1 pi@LoRaGateway:~# install.sh spi
```

Bei der Frage, ob eine remote settings file verwendet werden soll, wird in diesem Beispiel Nein gewählt. Weitere Informationen zu remote Configs sind auf der Github Page der Software zu finden. Nach Eingabe des Gateway-Namens, Beschreibung, etc. wird nochmals die EUI des Gateways ausgegeben und eventuell das System neu gestartet. Die EUI entspricht der MAC Adresse des Gateways.

Somit ist das Gateway selbst funktionsfähig und muss nur noch auf bei einem Networking-Server registriert und folglich konfiguriert werden. Die genaue Konfigurierung wird in einem späteren Abschnitt weiter erklärt.

### 4.1.4 Networking Server

Generell ist es eine gute Idee, für seine Networking Server ein bereits öffentliches Netzwerk wie beispielsweise The Things Network zu verwenden. Dies hat den Vorteil, dass man die Gateways anderer Benutzer direkt verwenden kann und generell keinen Aufwand für das Aufsetzen und Betreiben eines eigenen Servers aufwenden muss.

Als Vorbereitung für die Nutzung von TTN wird legentlich ein Benutzeraccount bei TTN benötigt.

Sollte man einen privaten Networking Server einrichten wollen so bietet sich die Implementation von <https://github.com/gotthardp/lorawan-server> an. Diese unterstützt zwar nur LoRaWAN v1.0.3, aber erlaubt es schnell einen privaten Server einzurichten.

Der Download der dateien ist über <https://github.com/gotthardp/lorawan-server/releases> möglich. In diesem Fall wird das debian Package ".deb" benötigt. Diese kann folglich auch direkt über dpkg installiert werden. Das debian Package wurde hier zu "lorawan-server.deb" umbenannt.

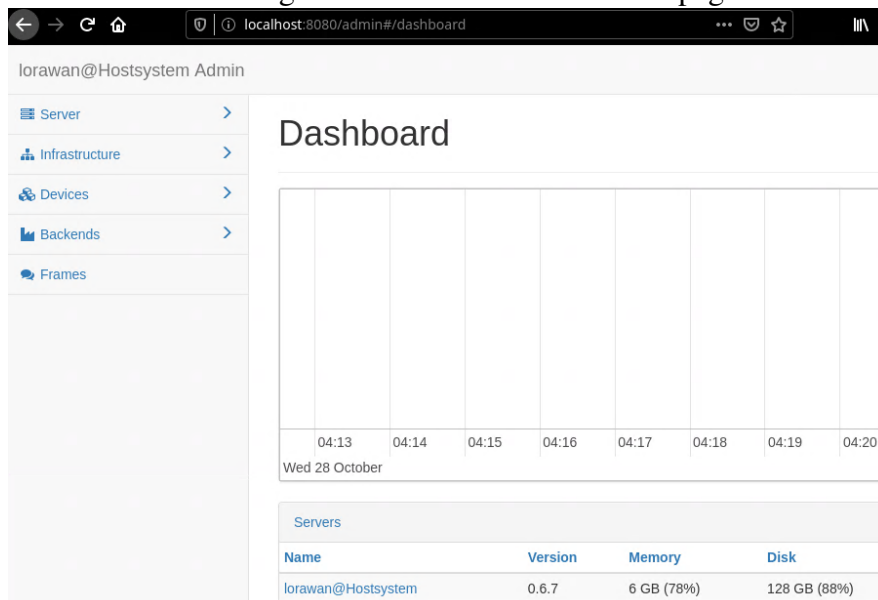
```
1 user@Hostsystem:~# dpkg -i lorawan-server.deb
```

Und somit ist die grundlegende Installation auch abgeschlossen. Der Status des Servers kann über systemctl geprüft und gegebenenfalls geändert werden.

```
1 user@Hostsystem:~# systemctl start lorawan-server
2 user@Hostsystem:~# systemctl status lorawan-server
3   lorawan-server.service - LoRaWAN Server
4   Loaded: loaded (/lib/systemd/system/lorawan-server.
         ↪ service;
5   disabled; vendor preset: enabled)
6   Active: active (running) since Mon
7   2020-10-12 11:57:54 JST; 6s ago
```

Grundlegende Konfiguration des Servers, wie Ports oder Adminlogins, können in `/usr/lib/lorawan-server/releases/<version>/sys.config` vorgenommen werden. Das Administrationsinterface wird über den Browser aufgerufen und ist by default über `localhost:8080` erreichbar.

Abbildung 4.8: LoRaWan-Server Adminpage



## 4.2 Konfiguration

Nun sind alle Devices und deren grundlegende Software bereit für ihre Konfiguration und folglich für den Aufbau des Netzwerks.

### 4.2.1 The Things Network

Der Aufbau des Netzwerks über TTN erfolgt durch das Registrieren des Gateways und folglich der End-Nodes. Hierzu muss man zuerst auf die TTN Console und sich mit seinem TTN Benutzer einloggen.

#### 4.2.1.1 Gateways

Zum Registrieren eines Gateways navigiert man auf der Mainpage der Console einfach zu Gateways und klickt auf den "register gateway" Button.

Folgende Daten sind nun einzutragen:

- Gateway ID - Hier muss eine eindeutige ID für das Gateway gewählt werden. Dabei bietet sich die EUI/MAC des Gateways an.

- Legacy Packet Forwarder - Im Falle dieser Gateway Implementierung wird ein Legacy Forwarder (Semtech UDP Protocol) verwendet.
- Description - Eine einfache Beschreibung des Gateways. (Ort, Zweck, etc.)
- Frequency Plan - Je nach Region in welcher das Gateway arbeiten soll unterschiedlich, für Europa wird 868MHz gewählt.
- Router - Networking Server Router zu welchem die Pakete vom Gateway gesendet werden. Hier wird ttn-router-eu für den Europäischen Router gewählt.
- Location - Hier kann auf der Karte der Standpunkt des Gateways gewählt werden. Dies ist hilfreich um Karten zu generieren welche die Abdeckung durch Gateways anzeigen.
- Antenna Placement - Hier einfach angeben, ob sich das Gateway innerhalb oder außerhalb eines Gebäudes befindet.

Die Registrierung würde mit diesen Einstellungen folglich so aussehen:

Abbildung 4.9: TTN Gateway Registrierung

**REGISTER GATEWAY**

**Gateway EUI**  
The EUI of the gateway as read from the LoRa module

BB 27 EB FF FF 38 5D 3F 8 bytes

**I'm using the legacy packet forwarder**  
Select this if you are using the legacy [Semtech packet forwarder](#).

**Description**  
A human-readable description of the gateway

Gateway used in my Bachelor's Thesis located in Aichach, Bavaria, Germany

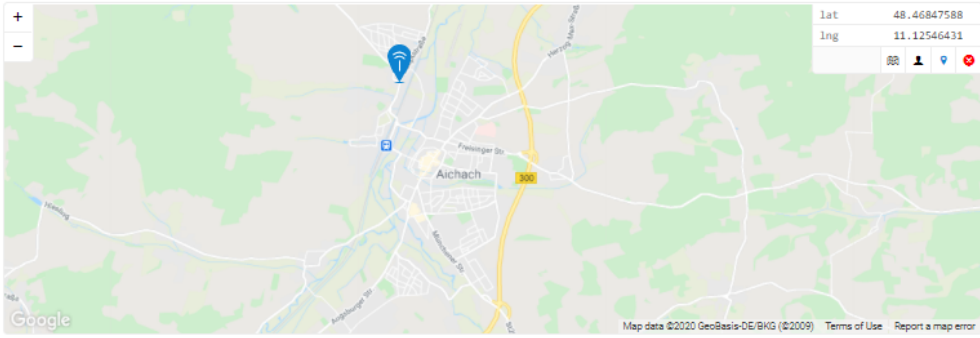
**Frequency Plan**  
The [frequency plan](#) this gateway will use

Europe 868MHz

**Router**  
The router this gateway will connect to. To reduce latency, pick a router that is in a region which is close to the location of the gateway.

ttn-router-eu

**Location**  
The exact location of your gateway. This will be used if your gateway cannot determine its location by itself. Set a location by clicking on the map.



lat	48.46847588
lng	11.12546431

**Antenna Placement**  
The placement of the gateway antenna

indoor  outdoor

Somit ist die Registrierung nach Bestätigung der Eingaben auch abgeschlossen und das Gateway selbst muss nur noch passend konfiguriert werden.

#### 4.2.1.2 End-Nodes

Nun müssen die End-Nodes noch registriert und folglich für Kommunikation konfiguriert werden. Der Registrierungsprozess ist für beide hier verwendeten End-Nodes gleich.

Zuerst muss eine Application erstellt werden, diese wird folglich die einzelnen

End-Nodes enthalten. Hierzu einfach über die TTN Konsole auf “Applications” und auf “add application”. Die Daten hier können wie gewünscht gewählt werden.

Abbildung 4.10: TTN Applikationserstellung

**ADD APPLICATION**

**Application ID**  
The unique identifier of your application on the network

bachelor-thesis-application

**Description**  
A human readable description of your new app

Application used in my Bachelor's Thesis.

**Application EUI**  
An application EUI will be issued for The Things Network block for convenience, you can add your own in the application settings page.

EUI Issued by The Things Network

**Handler registration**  
Select the handler you want to register this application to

ttn-handler-eu

Nun in der Applikation unter “Devices” auf “register device”. Device ID kann hierbei ein einfach Name oder eine ID für die End-Node sein, für die Device EUI kann entweder ein zufällig generierter Wert oder die EUI/MAC der End-Node verwendet werden. App Key und App EUI werden automatisch gesetzt.

Abbildung 4.11: TTN Deviceregistrierung

**REGISTER DEVICE** [bulk import devices](#)

**Device ID**  
This is the unique identifier for the device in this app. The device ID will be immutable.

lopy-lora\_version\_10-01

**Device EUI**  
The device EUI is the unique identifier for this device on the network. You can change the EUI later.

70 B3 D5 49 95 C2 54 3E 8 bytes

**App Key**  
The App Key will be used to secure the communication between you device and the network.

this field will be generated

**App EUI**

70 B3 D5 7E D0 03 2D 9F

In der Device Overview der registrierten End-Node können nun nochmals alle nötigen Keys für die End-Node oder auch Datenflüsse eingesehen oder Daten an vom Networking-Server an die End-Node gesendet werden.

Beispielcode zum Testen der End-Nodes wird nach der Konfiguration des privaten Servers angesprochen.

## 4.2.2 Private Server

Der Aufbau des Netzwerks für den Privaten Server funktioniert über das bereits angesprochene Administrationsinterface.

### 4.2.2.1 Gateways

Für die Erstellung eines Gateways muss zuerst eine Area angelegt werden. Diese gruppieren Gateways in einzelne, eigenständige Gebiete.

Abbildung 4.12: Erstellung einer Area

## Create new area

Name *	<input type="text" value="MyArea"/>	✓
Administrators	<input type="text" value="admin"/>	✓
Slack Channel	<input type="text"/>	✓
Log Ignored?	<input type="text" value="true"/>	✓

Folglich können Gateways erstellt werden. Die Einstellungen für TX Chain und Antenna Gain können hierbei aus den Spezifikationen des Gateways entnommen werden.



Abbildung 4.13: Gatewayerstellung

## Create new gateway

General

MAC *	<input type="text" value="B827EBFFFF385D3F"/>	✓
Area	<input type="text" value="MyArea"/>	✕ ✓
TX Chain *	<input type="text" value="0"/>	✓
Antenna Gain (dBi)	<input type="text" value="2"/>	✓
Description	<input type="text" value="My Test Gateway"/>	✓

Nun muss ein Network aufgesetzt werden. Das Network dient zur Gruppierung der Nodes in eine Obergruppe und definiert regionale Einstellung wie z.B. den Frequenzbereich der End-devices. Die Einstellungen für ein Network können aus der Spezifikation von LoRaWAN entnommen werden.

Abbildung 4.14: Gatewayerstellung

## Create new network

General   ADR   Channels

Name *	<input type="text" value="TestNetwork"/>	✓
NetID *	<input type="text" value="000000"/>	✓
Region *	<input type="text" value="EU 863-870MHz"/>	✓

Ein Network selbst ist wiederum in Groups aufgeteilt. Diese können wiederum über eigene Administratoren und Alert-Channel kontrolliert werden. SubID kann hier gesetzt werden, um den einzelnen Gruppen fixierte Bits in ihren Adressen zu geben. Dadurch wird die Identifikation bei mehreren Gruppen einfacher.

Abbildung 4.15: Gatewayerstellung

## Create new group

Name *	<input type="text" value="TestGroup"/>	✓
Network *	<input type="text" value="TestNetwork"/>	✓
SubID	<input type="text" value="e.g. 0:3"/>	✓
Administrators	<input type="text" value="admin"/>	✓
Slack Channel	<input type="text"/>	✓
Can Join?	<input type="text" value="true"/>	✓

Alle End-devices haben des weiteren ein Profil welches deren generelle Konfiguration beinhaltet. Diese sollten passend zu den Devices gewählt werden. Des weiteren kann einem Profil eine Application zugewiesen werden. Das erstellen einer Application wird zu einem späteren Punkt beschrieben.

Abbildung 4.16: Gatewayerstellung

## Create new profile

General	ADR	
Name *	<input type="text" value="TestLoPy"/>	✓
Group *	<input type="text" value="TestGroup"/>	✓
Application *	<input type="text" value="testapp"/>	✓

Nun kann das eigentliche Device unter "Comissioned" eingetragen und registriert werden. Hierzu wird die EUI/MAC des zu registrierenden Devices und dessen vorherig erstellte Profil benötigt. Weiterhin muss eine AppEUI und ein AppKey erstellt werden. Der AppKey sollte hierbei natürlich geheim gehalten werden.

Nachdem ein erstelltes Device zum ersten mal mit dem Server in Verbindung gesetzt wird, wird das Node-Feld automatisch ausgefüllt und auch die Last Joins

zeigen von dort an selbstständig Daten an.

Abbildung 4.17: Gatewayerstellung

## Create new device

<b>DevEUI *</b>	70B3D54995C2543E	✓
<b>Profile *</b>	TestLoPy	✓
<b>App Arguments</b>		✓
<b>AppEUI</b>	1234567891011121	✓
<b>AppKey *</b>	12345678910111213141516123456789	✓
<b>Description</b>	LoPy test node	✓
<b>Last Joins</b>		
<b>Node</b>	e.g. ABC12345	✓

Dies schliesst den generellen Aufbau des Netzwerks ab und ein registriertes End-device kann nun über ein registriertes Gateway mit dem Server kommunizieren. Zur Erweiterung kann man nun aber noch einen Applikationsserver einbinden, welcher die Daten der Nodes weiter verarbeiten kann.

Hierfür wird zuerst ein Handler definiert. Der Handler ist hierbei das selbe wie eine Applikation und definiert die Formatierung der Daten bei der Weiterleitung an einen Applikationsserver. Die einfachste Formatierung ist hierbei ASCII Text, welche die eingehenden Daten in einem einfachen JSON Format weiterleitet.

Über Uplink/Event Fields können hierbei vorgegebene Datenfelder ausgewählt werden welche folglich übertragen werden. D/L Expires steuert, ob ein Downlink verworfen werden soll, wenn das Zieldevice momentan keine Downlink Nachrichten annimmt.

Abbildung 4.18: Gatewayerstellung

## Create new handler

Application *	testapp	✓
Uplink Fields	data ×	✓
Payload	ASCII Text	× ✓
Parse Uplink		✓
Event Fields	event × deveui ×	✓
Parse Event		✓
Build Downlink		✓
D/L Expires *	Never	✓

Die Application muss nun nur noch an einen Connector angeschlossen werden. Der Connector selbst stellt die Verbindung zu einem Applikationsserver dar und kann über HTTPS, Websockets, Clodservices oder auch direkte Datenbankverbindungen betrieben werden. Dieses Beispiel verwendet HTTP.

Unter Application muss der passende Handler ausgewählt werden. Die URI ist die Adresse des Applikationsservers welche wiederum um die Handles für Uplinks und Events erweitert wird. In diesem Beispiel werden alle Uplinks der Applikation festappän `http://localhost:5569/ul` gesendet. Received Topic stellt den Handler für Downlinks dar. Über diesen kann der Applikationsserver einen Downlink an das Device mit der passenden EUI anfragen. Z.B. kann mit dieser Konfiguration ein Downlink über einen POST Request an `http://localhost:8080/70B3D54995C2543E` an das End-device mit EUI `70B3D54995C2543E` gesendet werden. Folglich muss die Checkbox bei Enabled noch ausgewählt werden, da diese angibt ob der Connector momentan aktiv ist.

Abbildung 4.19: Gatewayerstellung

## Create new connector

General
Authentication

<b>Connector Name *</b>	testbackend <span style="float: right; color: green;">✓</span>
<b>Application</b>	testapp <span style="float: right; color: blue;">✕</span> <span style="float: right; color: green;">✓</span>
<b>Format *</b>	JSON <span style="float: right; color: green;">✓</span>
<b>URI *</b>	http://localhost:5569 <span style="float: right; color: green;">✓</span>
<b>Publish QoS</b>	Filter values <span style="float: right;">▼</span>
<b>Publish Uplinks</b>	ul <span style="float: right; color: green;">✓</span>
<b>Publish Events</b>	event <span style="float: right; color: green;">✓</span>
<b>Subscribe QoS</b>	Filter values <span style="float: right;">▼</span>
<b>Subscribe</b>	
<b>Received Topic</b>	/{deveui} <span style="float: right; color: green;">✓</span>
<b>Enabled *</b>	<input checked="" type="checkbox"/> <span style="float: right; color: green;">✓</span>

Ein Beispiel für einen Applikationsserver geschrieben in golang, welcher sowohl Uplinks und Events als auch Downlinks implementiert kann im Git-Repository[11] unter "beispielcode" gefunden werden.

### 4.2.3 Gateway

Die Konfiguration des Gateways erfolgt über eine json Datei auf dem Gateway selbst oder bei Benutzung einer remote Config über die json Datei in dem Git-Repository der Konfigurationsdateien.

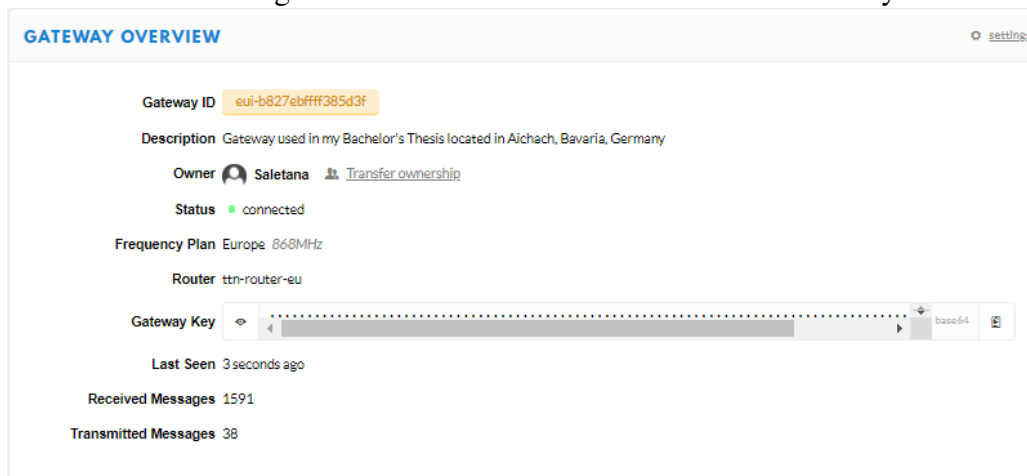
Die lokale Datei, local\_conf.json befindet sich hierbei auf dem Gateway unter /opt/ttn-gateway/bin. Hierbei müssen die passenden Daten der Zielservers (servers) und eventuell die neue ID des Gateway (gateway\_ID) gesetzt werden.

```
1  {
2    "gateway_conf": {
3      "gateway_ID": "B827EBFFFF385D3F",
4      "servers": [ {
5        "server_address": "router.eu.thethings.
6          network",
7        "serv_port_up": 1700, "serv_port_down":
8          1700,
9        "serv_enabled": true } ],
10     "ref_latitude": 48.4684,
11     "ref_longitude": 11.1254,
12     "ref_altitude": 500,
13     "contact_email": "marco.ziegler@hs-
14       augsburg.de",
15     "description":
16       "Gateway located in Aichach, Bavaria,
17       Germany."
```

Je nach Einstellungen muss bei Server der richtige, bei der Registrierung des Gateways gewählte, Router ausgewählt werden. Weitere Informationen wie Kontaktmail oder Koordinaten können hier auch hinterlegt werden. Bei Verwendung eines privaten Servers muss die IP oder der Hostname des Zielservers inklusive Zielports eingetragen werden.

Nach einem Neustart des Gateways und einer kurzen Wartezeit wird das Gateway auf der TTN Konsole bzw. auf dem Dashboard des Privatserver als Verbunden angezeigt.

Abbildung 4.20: TTN Konsole mit verbundenem Gateway



## 4.2.4 End-Nodes

Konfiguration der End-Nodes erfolgt generell nach dem gleichen Prinzip. Die statischen Schlüssel und IDs müssen der Node mitgeteilt werden damit diese sich bei dem Networking Server authentifizieren können.

Bei OTAA, over the air authentication, wird von der End-Node hierbei die Device EUI, Application EUI und der App Key benötigt. Weitere Informationen zu den Schlüsseln befinden sich auf der TTN Website unter Security[20].

### 4.2.4.1 LoPy

Konfiguration und Verbindungsaufbau des LoPy über OTAA erfolgt simpel durch den Aufruf der join Funktion eines LoRa Objektes. Die Dokumentation des LoRa Objektes befindet sich auf der Seite von Pycom[18].

```
1 from network import LoRa
2
3 eui = 'XXXXD57ED003XXXX'
4 key = 'XXXXXX2F58C26686942A38F38CXXXXXX'
5 app_eui = ubinascii.unhexlify(eui)
6 app_key = ubinascii.unhexlify(key)
7
8 lora = LoRa(mode=LoRa.LORAWAN, region=LoRa.EU868)
9 lora.join(activation=LoRa.OTAA,
10 auth=(app_eui, app_key), timeout=0)
```

Dieser Code konfiguriert das LoPy auf die Benutzung von LoRaWAN im Frequenzbereich von Europa. Der Joinrequest wird über OTAA mit app\_eui für die Application EUI und app\_key für den App Key ausgeführt. Der timeout von 0 gibt die Wartezeit in Millisekunden auf einen Join-Accept vom Networking-Server an.

Die Device EUI wird automatisch von dem Device selbst ausgelesen, sollte man diese manuell setzen wollen so muss die eigene Device EUI im auth-Tuple vorne angehängt werden.

```
1 lora.join(activation=LoRa.OTAA, auth=(dev_eui, app_eui
    ↪ , app_key),
2 timeout=0)
```

Der Status des Join kann über lora.has\_joined() abgefragt werden und nach erfolgreichem Join kann Socket für das Empfangen und Senden von Datenpaketen aufgesetzt werden.

```
1 while not lora.has_joined():
2     time.sleep(2)
3     print('Wait_for_connection_here...')
4     print('Connection_established!')
```

Folglich können LoRa-Sockets für das Senden und Empfangen von LoRa-Packets erstellt werden.

```
1 #AF_LORA erstellt einen LoRa Socket
2 #SOCK_RAW erzeugt einen "raw" Socket ohne IP-
    ↪ Capsulation
3 s = socket.socket(socket.AF_LORA, socket.SOCK_RAW)
4
5 #SOL_LORA definiert weitere Optionen für LoRa
6 #SO_DR gibt die Datenrate des Sockets an, in diesem
    ↪ Fall 5
7 s.setsockopt(socket.SOL_LORA, socket.SO_DR, 5)
```

Nach Erstellen der Sockets können direkt Pakete gesendet und empfangen werden. Hier beginnt die Hauptschleife der Node in welcher z.B. Daten aus Sensoren ausgelesen und an das Netzwerk gesendet werden und eingehende Pakete bearbeitet werden können.

```
1 #Sensordaten auslesen, Pseudocode
2 sensor_data = getSensordata()
3
4 #Senden von Daten über den Socket
```



```
5 s.setblocking(True)
6 #Zu sendende Beispieldaten mit Sensordaten und Device-
  ↳ EUI
7 out = {
8 'data' : sensor_data,
9 'eui'   : str(binascii.hexlify(LoRa().mac()).upper())
  ↳ [2:-1]
10 }
11
12 #Senden der Daten über den Socket im json Format
13 s.send(json.dumps(out))
14 s.setblocking(False)
15
16 #Empfangen von Datenpaketen
17 #Auslesen der ersten 64 Byte im Socket
18 in = s.recv(64)
19
20 #Leere Bytestrings werden durch b'' angezeigt
21 if str(data) is not str("b' '"):
22     print('Received:_' + str(data))
23 else:
24     print('No_new_data._' + str(data))
25
26 #Sleeptime um zu warten, Beispielsweise 10s
27 time.sleep(10)
```

Damit ist die LoPy End-Node funktionsbereit und verbindet sich nach start der Node automatisch mit dem passenden Network, solange ein Gateway für jenes Network in Reichweite des LoPy ist. Beispielcode befindet sich ebenfalls im Git-Repository unter "beispielcode".

#### 4.2.4.2 B-L07Z-LRWAN1

Die Konfiguration der LoRaMac-Node Software wird durch eine Kombination aus den Einstellungen von cmake und Einträgen in C-Header Files vorgenommen.

Für einen einfachen Verbindungsaufbau sollten zuerst die cmake Flags passend gesetzt werden. Hardwarespezifikationen sind hierbei dem Specsheat des jeweiligen End-Node Boards zu entnehmen.

- ACTIVE\_REGION - Der zu verwendende Frequenzbereich der End-Node,

muss passend zur Region (EU868 für Europa) in welcher sich die End-Node befindet gesetzt werden.

- APPLICATION - Generelle Funktion des End-Node, LoRaMac ist für normale Kommunikation von End-Node über Gateways zu Networking-Servern.
- BOARD - Die Hardware der End-Node, in diesem Fall B-L072Z-LRWAN1.
- MBED\_RADIO\_SHIELD - Sollte passend zur End-Node gewählt werden. Im Falle des B-L072Z-LRWAN1 ist dies SX1276MB1MAS.
- OPENOCD\_INTERFACE - Board-Interface Konfigurationsdatei für openocd, hier stlink-v2-1.cfg.
- OPENOCD\_TARGET - Hardware Konfigurationsdatei für openocd, hier stm3210.cfg.
- SECURE\_ELEMENT - Die zu verwendende Security Implementierung. SOFT\_SE für die Standardimplementierung.
- SUB\_PROJECT - Hier kann die Klasse bzw. Funktionsart der End-Node bestimmt werden. classA für eine normale End-Node der Klasse A.

Mit diesen Einstellungen ist die Konfiguration von cmake selbst abgeschlossen und es muss folglich nur noch über C-Header Files die restliche Konfiguration vorgenommen werden.

Die wichtigsten Header Files sind hierbei se-identity.h und Commissioning.h.

Zuerst sollte Commissioning.h falls nötig angepasst werden. In dieser Datei wird zwischen OTAA oder ABP gewechselt und eingestellt, ob das Zielnetzwerk ein öffentliches oder privates Netzwerk ist.

se-identity.h hält Einstellungen für EUIs und Keys. Unter anderem werden hier also der AppKey, AppEUI oder auch eine statische DeviceEUI oder Device Address eingetragen welche mit den Daten des Networking-Servers übereinstimmend eingetragen werden müssen.

Die restliche Funktionalität der End-Node ist bereits in LoRaMac-Node selbst implementiert und nach korrekter Einstellung von cmake und Eintragen der passenden Einstellungen und EUIs/Keys in den C-Header Files, kann der Code über die Build Funktion von VSC oder make kompiliert und folglich über openocd oder gdb-multiarch auf das Board programmiert werden. Nach dem Flashvorgang ist auch diese End-Node funktionsbereit und verbindet sich über ein Gateway mit dem passenden Network.

# Kapitel 5

## Verhaltensanalyse

Nach Aufbau der Netzwerke kann nun das Verhalten der verschieden versionierten End-Nodes und Netzwerke untereinander näher betrachtet werden. Hierzu werden für die End-Nodes äquivalente Softwareimplementationen erstellt und folglich das Verhalten dieser in den verschiedenen Netzwerken betrachtet.

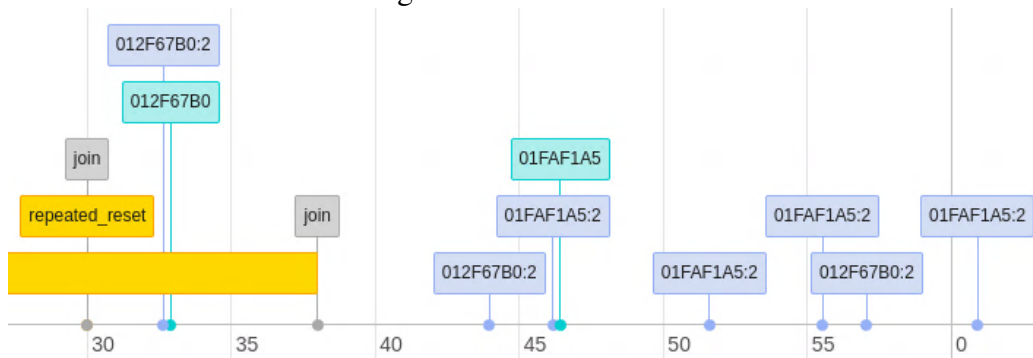
Da viele Neuerungen von Version 1.1 in Security Enhancements bei Join und genereller Übertragung von Daten liegen ist eine nähere Betrachtung der Kompatibilität zwischen Versionen ein interessanter Analysepunkt. Aus der Spezifikation von LoRaWAN 1.1[14] sollte hierbei ein Fallback der höher versionierten Komponente auf die Version der älteren Komponente stets möglich sein.

Dies bedeutet, dass ein Server der Version 1.1 bei einem Joinrequest einer 1.0x versionierten End-Node einen Fallback auf Version 1.0x durchführen sollte. Eine End-Node mit höherer Version sollte ebenfalls analog einen Fallback durchführen.

Der Test besteht hierzu aus einem einfachen Join der jeweiligen End-Nodes zu TTN und dem Privatserver. Alle Joins sollten ohne Probleme ablaufen wenn der Fallback auf den höher Versionierten Komponenten wie geplant funktioniert. TTN unterstützt hierbei sowohl OTAA als auch ABP, da der Private LoRaWAN Server nur OTAA unterstützt, wird auf diesem kein ABP getestet.

Bei dem Privatserver werden die Events für Join, Uplink und Downlink direkt auf dem Dashboard angezeigt. Eine Ausgabe beim gleichzeitigen Start beider End-Nodes kann folglich so aussehen:

Abbildung 5.1: Privatserver Dashboard



Die in gelb angezeigten Events sind hierbei Reset Events welche durch den Neustart der End-Nodes erzeugt werden. Beide Join Events korrespondieren jeweils zum Join des LoPy und des B-L072Z-LRWAN1. Die beiden hellblauen Events zeigen Downlinks zu den End-Nodes der zusammen angezeigten Device-Address an. Registriert sind die beiden End-Nodes auf dem Server als 012F67B0 für den LoPy und 01FAF1A5 für den B-L072Z-LRWAN1. Blaue Events zeigen Uplink-Events von den jeweiligen End-Nodes an.

Die Events werden an den Applikationsserver weitergeleitet welcher die Daten beliebig verwenden kann.

Abbildung 5.2: Eingehende Events auf dem Applikationsserver

```

2020/11/14 18:37:32 Running on port 5569
2020/11/14 18:41:48 127.0.0.1:39705 POST /event
2020/11/14 18:41:48 {Reader:{"datetime":"2020-11-14T18:41:48Z","deveui":"70B3D54995C2543E","event":"joined"}}
2020/11/14 18:41:56 127.0.0.1:39705 POST /ul
2020/11/14 18:41:56 {Reader:{"eui": "70B3D54995C2543E", "data": "Testdata - Uplink"}}
2020/11/14 18:42:03 127.0.0.1:39705 POST /event
2020/11/14 18:42:03 {Reader:{"datetime":"2020-11-14T18:42:03Z","deveui":"343531316E376712","event":"joined"}}
[...]
2020/11/14 18:42:11 127.0.0.1:39705 POST /ul
2020/11/14 18:42:11 {Reader:{"eui": "343531316E376712", "data": "Testdata - Uplink"}}
    
```

Die Verbindungen und Events können bei TTN direkt auf dem Dashboard der einzelnen End-Nodes bzw. des Gateways ausgelesen werden und zeigen bei korrekter Konfiguration gleiche Ergebnisse.

Dieser Test zeigt, dass der Fallback der 1.1 Versionierten Server und End-Nodes erfolgreich die Kompatibilität beim Join gewährleistet.

Tabelle 5.1: Verbindungen zwischen End-Nodes und Netzwerken

	TTN (OTAA)	TTN (ABP)	Private (OTAA)
LoPy	✓	✓	✓
B-L072Z	✓	✓	✓

Nach einem Join muss ebenfalls der Transfer von Up- und Downlinks gewährleistet sein. Diese Test erfolgen direkt über die TTN Console bzw. den vorherig angesprochenen Applikationsserver für den Private Server. Sowohl die TTN Console als auch der Applikationsserver bieten hierbei die Möglichkeit, Uplinks zu empfangen und Downlinks an die End-Nodes zu senden.

Auf der TTN-Console Seite werden die Uplink und Downlink Events tabellarisch aufgezeichnet.

Abbildung 5.3: Eventauflistung bei TTN

time	counter	port	
16:22:56	1		payload: 44 6F 77 6E 6C 69 6E 6B 20 44 61 74 61 21
16:22:58	1	2	payload: 7B 22 65 75 69 22 3A 20 22 37 30 42 33 44 35 34 39 39 35 43 32 35 34 33 45 22 2C 20 22 64 61 74
16:22:54	1	scheduled	payload: 44 6F 77 6E 6C 69 6E 6B 20 44 61 74 61 21
16:22:45	0	2	payload: 7B 22 65 75 69 22 3A 20 22 37 30 42 33 44 35 34 39 39 35 43 32 35 34 33 45 22 2C 20 22 64 61 74
16:22:34			dev addr: 26 01 4B 70    app eui: 70 B3D5 7E D0 03 2D 9F    dev eui: 70 B3D5 49 95 C2 54 3E

Der gelbe Blitz zeigt einen Join-Request der Node an. Blaue Pfeile beziehen sich auf Up- bzw. Downlinks und der graue Pfeil zeigt einen geplanten (scheduled) Downlink ausgehend von TTN an. Sobald die End-Node ein offenes Receive-Window anzeigt, wird der geplante Downlink durchgeführt.

Analog hierzu können die Events auch auf dem Gateway ausgelesen werden. Auf diesem sieht man auch ein Event angezeigt mit einem grünen Blitz welcher den Join-Accept des Networking Servers repräsentiert.

Abbildung 5.4: Analoge Events auf dem Gateway

▼ 16:22:58	867.5	loro	4/5	SF 7 BW 125	66.8	0	dev addr: 26 01 4B 70	payload size: 27 bytes
▲ 16:22:58	867.5	loro	4/5	SF 7 BW 125	128.3	1	dev addr: 26 01 4B 70	payload size: 69 bytes
▲ 16:22:45	867.3	loro	4/5	SF 7 BW 125	128.3	0	dev addr: 26 01 4B 70	payload size: 69 bytes
⚡ 16:22:40	868.3		4/5	SF 7 BW 125	71.9			
⚡ 16:22:36	868.3		4/5	SF 7 BW 125	61.7		appeul: 70 B3D57ED0032D 9F	dev eul: 7C

Die Events können auch auf den End-Devices betrachtet bzw. die Daten des Downlink beliebig verwendet werden.

Rechts sieht man hierzu eine mögliche Konsolenausgabe einer End-Node bei der zuerst ein Join Request über OTTA gesendet wird. Nach Warten auf eine Antwort, wird der Join-Accept des Networking-Servers über das Gateway empfangen und die Verbindung ist etabliert. Nun kann die End-Node über Uplink-Events Daten an den Networking-Server senden und über Downlink-Events von diesem empfangen.

```
Device EUI: b'70B3D54995C2543E'
Joining using OTAA.
Not yet joined...
Not yet joined...
Not yet joined...
Joined
Sending...
Received: b'Downlink Data!'
```

Nach Testen der Uplink und Downlink Übertragungen zu und von den beiden End-Nodes zusammen mit den beiden Servern ergibt sich folgendes Ergebnis:

Tabelle 5.2: Verbindungen zwischen End-Nodes und Netzwerken

	TTN Uplink	TTN Downlink	Private Uplink	Private Downlink
LoPy	✓	✓	✓	✓
B-L072Z	✓	✓	✓	✓

Die Testergebnisse hier zeigen ebenfalls, dass die Kompatibilität zwischen den Versionen bei Up- und Downlink weiterhin gewährleistet ist.

# Kapitel 6

## Zusammenfassung

LoRaWAN 1.1 hat viele Änderungen vor allem im Bereich der Verschlüsselung und dem Kommunikationsablauf zwischen End-Nodes und Netzwerkservers in das LoRaWAN Protokoll etabliert und somit potentielle Inkompatibilitäten zwischen Servern und End-Nodes der neueren und älteren Versionen erzeugt. Dies ist vor allem problematisch, da viele ältere End-Nodes bereits in Benutzung sind und generell noch für mehrere Jahre funktionieren sollten. Desweiteren ist hierbei ein einfaches Softwareupdate nicht nur, aufgrund der Anzahl und weiten Verstreuung der End-Nodes, schwierig sondern wegen neuen Hardwareanforderungen wie persistenten Speicher teils nicht möglich.

Um dieses Problem zu vermeiden sollte es aber höher versionierten Komponenten möglich sein auf eine ältere Version der Kommunikationsprotokolle zurückzufallen um eine Kompatibilität zwischen beispielsweise neuen Servern und alten End-Devices zu gewährleisten.

Um zu testen ob diese Fallbacks in der Praxis wirklich implementiert und auch funktional sind, wurde ein gemischtes Netzwerk aus End-Nodes und Servern verschiedener Versionen aufgebaut. Eines dieser Netzwerke basiert auf dem öffentlichen LoRaWAN Netzwerk “The Things Networ” und stellt eine moderne Implementierung eines Version 1.1 kompatiblen Networking Servers dar. Als Testserver für Version 1.0x wurde eine Networking-Server Implementierung[6] für ein privates LoRaWAN Netzwerk verwendet um auch neuere End-Nodes im Umgang mit älteren Servern zu testen. Als End-Nodes wurden ein LoPy1.0r für Version 1.0x und ein B-L072Z-LRWAN1 als 1.1 kompatibles Device verwendet. Folglich wurden die Server und End-Nodes mit einem RaspberryPi/ic880A Gateway verbunden und die Kommunikation angefangen von Network Joins der End-Nodes bis zu Datentransfer über Uplinks und Downlinks auf Anomalien untersucht.

Hierbei hat sich herausgestellt dass diese vorgesehenen Fallbacks auf Kommu-

nikationsprotokolle, welche mit allen an der Verbindung beteiligten Devices und Servern kompatibel sind, funktioniert. Daher ist es nicht nötig ältere Devices auf eine neue Softwareversion zu aktualisieren oder diese aufgrund von Hardwarebegrenzungen gar komplett zu ersetzen.



# Literatur

- [1] *Atom*. URL: <https://atom.io/>.
- [2] *Classes | The Things Network*. URL: <https://www.thethingsnetwork.org/docs/lorawan/classes.html>.
- [3] *Diskussion iC880a Gateway*. URL: <https://www.thethingsnetwork.org/forum/t/anybody-running-an-ic880a-based-gateway-tips-comments/3109/10>.
- [4] *Gateways | The Things Network*. URL: <https://www.thethingsnetwork.org/docs/gateways/>.
- [5] *GDB: The GNU Project Debugger*. URL: <https://www.gnu.org/software/gdb/>.
- [6] *GitHub - gotthardp/lorawan-server: Compact server for private LoRaWAN networks*. URL: <https://github.com/gotthardp/lorawan-server>.
- [7] *GitHub - Lora-net/LoRaMac-node: Reference implementation and documentation of a LoRa network node*. URL: <https://github.com/Lora-net/LoRaMac-node>.
- [8] *GitHub - LoRaMac-node/development-environment*. URL: <https://github.com/Lora-net/LoRaMac-node/blob/master/doc/development-environment.md>.
- [9] *GitHub - stlink-org/stlink: Open source STM32 MCU programming toolset*. URL: <https://github.com/stlink-org/stlink>.
- [10] *GitHub - ttn-zh/ic880a-gateway: Reference setup for iC880a gateways running The Things Network*. URL: <https://github.com/ttn-zh/ic880a-gateway>.
- [11] *GitLab - BA-LoRaWan1.1*. URL: <https://r-n-d.informatik.hs-augsburg.de:8080/mz1337/ba-lorawan1.1>.
- [12] *Installing Atom*. URL: <https://github.com/atom/atom/releases>.

- [13] *LoRa Examples*. URL: <https://docs.pycom.io/tutorials/networks/lora/>.
- [14] *LoRaWAN Specification v1.1 | LoRa Alliance*®. URL: <https://loralliance.org/resource-hub/lorawanr-specification-v11>.
- [15] *Network Architecture | The Things Network*. URL: <https://www.thethingsnetwork.org/docs/network/architecture.html>.
- [16] *Nodes Lebenszeit Seite 9*. URL: <https://www.ntu.edu.sg/home/limo/papers/TOSN-LoRa.pdf>.
- [17] *OpenOCD - eLinux.org*. URL: <https://elinux.org/OpenOCD>.
- [18] *Pycom - LoRa Documentation*. URL: <https://docs.pycom.io/firmwareapi/pycom/network/lora/>.
- [19] *The Things Network*. URL: <https://www.thethingsnetwork.org/>.
- [20] *The Things Network - Security*. URL: <https://www.thethingsnetwork.org/docs/lorawan/security.html>.
- [21] *Visual Studio Code - Code Editing. Redefined*. URL: <https://code.visualstudio.com/>.
- [22] *Visual Studio Code - Download*. URL: <https://code.visualstudio.com/download>.
- [23] *Visual Studio Code - Download*. URL: <https://snapcraft.io/store>.