
PyLoRaWebChat Dokumentation

Release 0.1

Steffen Exler

26.06.2019

Inhaltsverzeichnis:

1	Ziel	1
2	Aufbau	2
2.1	Ziel Aufbau	2
2.2	Probleme mit dem Ziel Aufbau	2
2.3	Umgesetzter Aufbau	3
2.4	Docker	3
2.5	Website	4
3	Quickstart	5
3.1	HIMO-01P mit den Linux Host verbinden	5
3.2	Einstellungen	5
3.3	Software herunterladen und starten	5
4	Protokoll	7
4.1	Funk Richtlinien	7
4.2	Gerät Adresse	7
4.3	Anwesenheit im Netzwerk senden	8
4.4	Nachricht an anderen Node senden & empfangen	8

KAPITEL 1

Ziel

Ziel war es mithilfe des LoRa Moduls HIMO-01P innerhalb unseres Kurses ein eigenständiges Netzwerk aufzubauen, wobei jeder Teilnehmer an dem Protokoll mitgewirkt hatte aber die Implementation selbst schreiben musste, siehe [Abb. 1.1](#).

Als Hardware Voraussetzung wurde nur das HIMO-01P vorgegeben, ob dieses Modul mit ein Arduino, Raspberry Pi, Android oder PC betrieben wird, war den Teilnehmer überlassen.

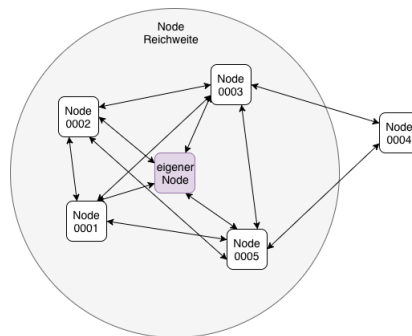


Abb. 1.1: LoRa Netzwerk

2.1 Ziel Aufbau

Um den Aufbau mit möglichst wenig Bauteilen zu erstellen, sollte das HIMO-01P Funkmodul über ein Arduino direkt an ein MacBook angeschlossen werden. Der Arduino soll als Serielle Brücke zwischen den Host System und dem Funkmodul dienen, siehe Abb. 2.1.

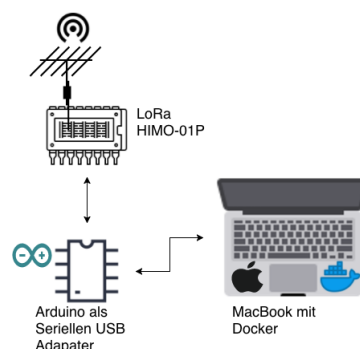


Abb. 2.1: Ziel Aufbau

Als Endprodukt soll via eine Website ein Echtzeit-Chat aufgebaut werden, worüber alle Teilnehmer des Netzwerkes automatisch aufgelistet werden und jeder Teilnehmer ein eigenen Chat Verlauf erhält.

Als Grundlage für die Website dient Django, welches in einem Docker Container gestartet werden soll, um die Anwendung Plattformunabhängig zu gestalten. Außerdem dient Docker in diesem Fall, das System auch in Zukunft auf neuen System auszuführen, ohne Anpassungen am Quellcode zu tätigen.

2.2 Probleme mit dem Ziel Aufbau

2.2.1 Nicht vollfunktionsfähige Arduino Nanos Kopien

Nicht alle Arduino Kopien funktionieren wie vorgeschrieben, in diesen Aufbau gab es bei 2 verschiedenen Modellen unterschiedliche Probleme:

1. Das erste Modell funktionierte die Kommunikation über den RX und TX Port nicht.

2. Bei dem zweiten Modell lieferte über den 3.3V Stromausgang eine zu niedrige Spannung, wodurch das HIMO-01P Modul nicht betrieben werden konnte.

2.2.2 Defekte Kabel

Ohne das dies am Anfang bemerkt wurde, gabe es in den ersten Versuchen defekte Kabel:

1. Ein einardriges Kabel zur Verbindung von dem Arduino mit dem HIMO-01P war gebrochen.
2. Das Micro USB Kabel, zum betreiben des Arduinos, besaß keine Leitung zur Kommunikation zwischen dem Host und dem Arduino. Es lieferte ausschließlich Strom.

2.2.3 Docker USB Geräte Freigabe unter Mac

Da für MacOS und Windows andere USB-Treiber für die Endgeräte notwendig sind als Linux, ist es nicht möglich USB-Hardware für Docker freizugeben. Das Problem ist bei den Docker Entwickler seid Jahren bekannt, da dies aber eine niedrige Prioität besitzt, gibt es bislang keine Offizielle Lösung.

2.3 Umgesetzter Aufbau

Um die Probleme zu lösen, wurde das Host System mit ein Ubuntu 18.04 ausgetauscht und als Arduino wird ein Arduino Uno verwendet, siehe [Abb. 2.2](#).

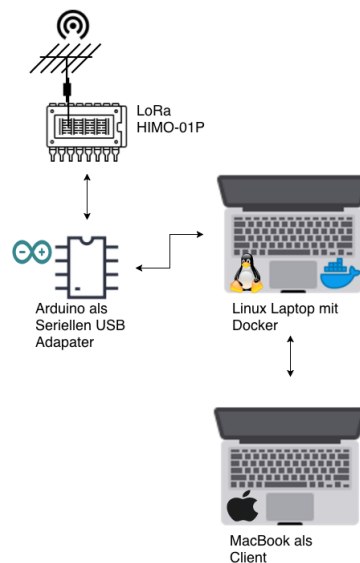


Abb. 2.2: Umgesetzter Aufbau

2.4 Docker

Die Docker Container werden via einer Docker-Compose Datei verwaltet. Darin werden die Abhängigkeiten innerhalb der Container festgelegt, auch welche Resourcen die Container von dem Host System erhalten.

Als Grundlage des Projektes wurde [cookiecutter-django](#) genommen, welches eine funktionierende Django Anwendung für Docker erstellt.

Für diesen Aufbau werden 3 Container erstellt:

- `django`: Enthält die Website basierend auf Django und ein Daemon zur Interaktion mit HIMO-01P

- `postgres`: Postgres Datenbank
- `redis`: Redis Server als Cache für Websockets und als Message Queue für den HIMO-01P Daemon

In Abb. 2.3 wird der Aufbau grafisch Dargestellt.

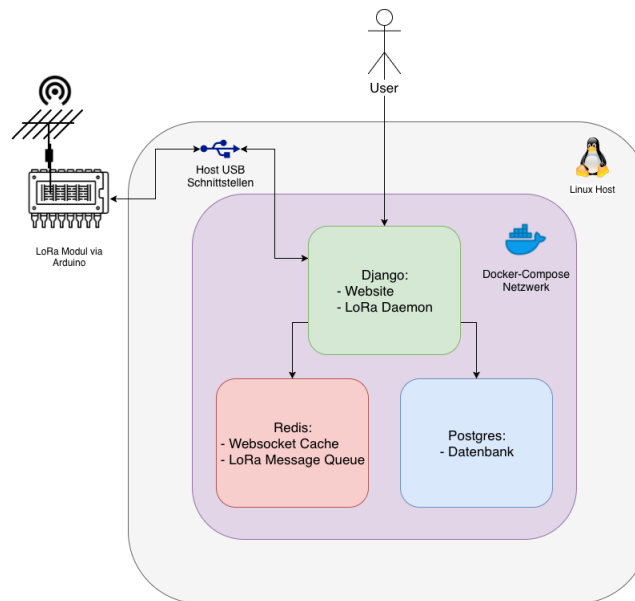


Abb. 2.3: Docker Aufbau

2.5 Website

Die Website enthält die Komponenten Frondend, Backend, Redis Cache und den LoRa Daemon für das HIMO-01P Modul.

Das Frondend kommuniziert über Websockets direkt mit dem Django Backend, es sendet zum Server ausgehende Nachrichten und erhält vom Backend neue bzw. update von Nachrichten und Nodes.

Für das caching von den Nachrichten nutzt das Backend ein Redis Server für die Websockets & Ausgehende LoRa Nachrichten. Sobald ein Node oder Message Objekt innerhalb der Datenbank geändert oder neu erstellt wird, sendet anschließend das Backend diese Information zum Client via Websocket.

In Abb. 2.4 ist die Kommunikation als Diagramm Dargestellt.

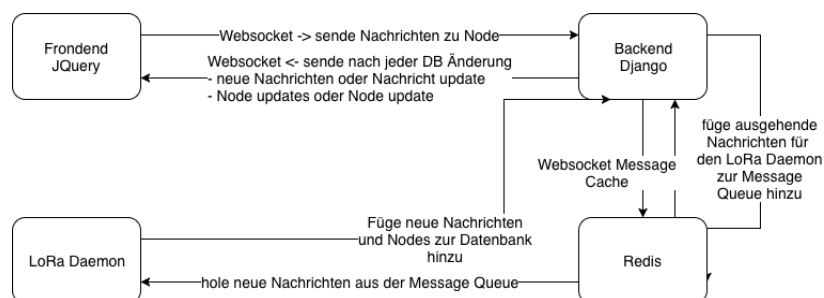


Abb. 2.4: Website Aufbau

Warnung: Diese Anleitung ist für Debian / Ubuntu mit installierten Docker und Docker-Compose geschrieben.

3.1 HIMO-01P mit den Linux Host verbinden

HIMO-01P Modul mit den Arduino wie folgt verbinden.

Arduino	HIMO-01P
TX	TX
RX	RX
GND	GND
3.3V	VIN
RESET -> GND	

Der Arduino muss nun via USB an den Linux Host angeschlossen werden. Jetzt sollte das HIMO-01P Modul unter den Port `/dev/ttyACM0` direkt ansprechbar sein.

3.2 Einstellungen

Um Änderungen wie die Adresse oder der HIMO-01P Einstellungen zu ändern, muss die Datei `.envs/.local/.serial` angepasst werden.

3.3 Software herunterladen und starten

Die aktuelle Projekt Version herunterladen via Git.:

```
$ git clone git@github.com:linuxluigi/PyLoRaWebChat.git
$ cd PyLoRaWebChat
```

Docker Images erstellen:

```
$ docker-compose -f local.yml build
```

Website starten:

```
$ docker-compose -f local.yml up -d django # in Background starten
$ docker-compose -f local.yml up django    # starten mit Log output
```

Admin Benutzer erstellen:

```
$ docker-compose -f local.yml run --rm django python manage.py createsuperuser
```

LoRa Daemon starten:

```
$ docker-compose -f local.yml run --rm django python manage.py lora_daemon
```

Nun ist es möglich auf der Website mit dem erstellten Admin Benutzer im Backend ein zu loggen um zugriff auf die Hauptwebsite zu erhalten. Dafür auf den Hostname des Host Systems im Browser eingeben wie `http://localhost:8000/admin`. Anschließend auf `http://localhost:8000/` gehen um den LoRa Echtzeit-chat einzusetzen.

4.1 Funk Richtlinien

Feld	Wert
Trägerfrequenz	433000000
Leistung	20
Bandbreite	6
Spreizfaktor	10
Fehlerkorrekturcode	1
CRC- Prüfung	1
impliziter Header	0
einmaliger Empfang	0
Frequenzmodulation	0
Frequenzmodulationsperiode	0
Empfangszeitlimitzeit	3000
Benutzerdatenlänge	8
Präambellänge	4

HIMO-01P Befehl zur Gerät Konfiguration:

```
AT+CFG=433000000,20,6,10,1,1,0,0,0,0,3000,8,4
```

4.2 Gerät Adresse

Jeder Node erhält eine eigene Adresse, die Adresse ist auf jedes HIMO-01P Modul aufgeklebt.

HIMO-01P Befehl zur Adressen eingabe für Gerät 10:

```
AT+ADDR=0010
```

4.3 Anwesenheit im Netzwerk senden

Jeder Netzwerk Teilnehmer (Node) soll alle 30 bis 60 Sekunden ein RTI zum Broadcast senden. Um dauerhafte Kollisionen zu vermeiden, soll das RTI jede Übertragung in unregelmäßigen Abständen gesendet werden.

RTI steht hierbei für Routing Table Information. Der Befehl für das HIMO-01P Modul ist:

```
AT+DEST=FFFF
AT+SEND=3
RTI
```

Jeder Teilnehmer erhält nun LR, 0010, 03, RTI:

- LR Nachrichten eingang
- 0010 Adresse des Absenders, absender Adresse ist immer 4 Stellen groß
- 03 Länge der Nachricht
- RTI RTI Nachricht

Jeder Teilnehmer kann nun neue Nodes im Netzwerk finden und diese zu ihrer eigenen Routing Information Table hinzufügen.

4.4 Nachricht an anderen Node senden & empfangen

Die maximale Nachrichten Länge beträgt 250 Zeichen, diese muss den HIMO-01P Modul vor dem Senden mitgeteilt werden, sowie die Ziel Adresse. Der Befehl zum senden der Nachricht Hallo, welche 5 Zeichen lang ist, an der Adresse 0001 von dem Absender 0010 ist:

```
AT+DEST=0001
AT+SEND=5
Hallo
```

Der Empfänger erhält dadurch LR, 0010, 05, Hallo:

- LR Nachrichten eingang
- 0010 Adresse des Absenders, absender Adresse ist immer 4 Stellen groß
- 05 Länge der Nachricht
- Hallo Nachricht