

Projekt: <https://github.com/ronaldsieber/LoraAmbientMonitor>  
Lizenz: MIT  
Autor: Ronald Sieber

## LoraAmbientMonitor

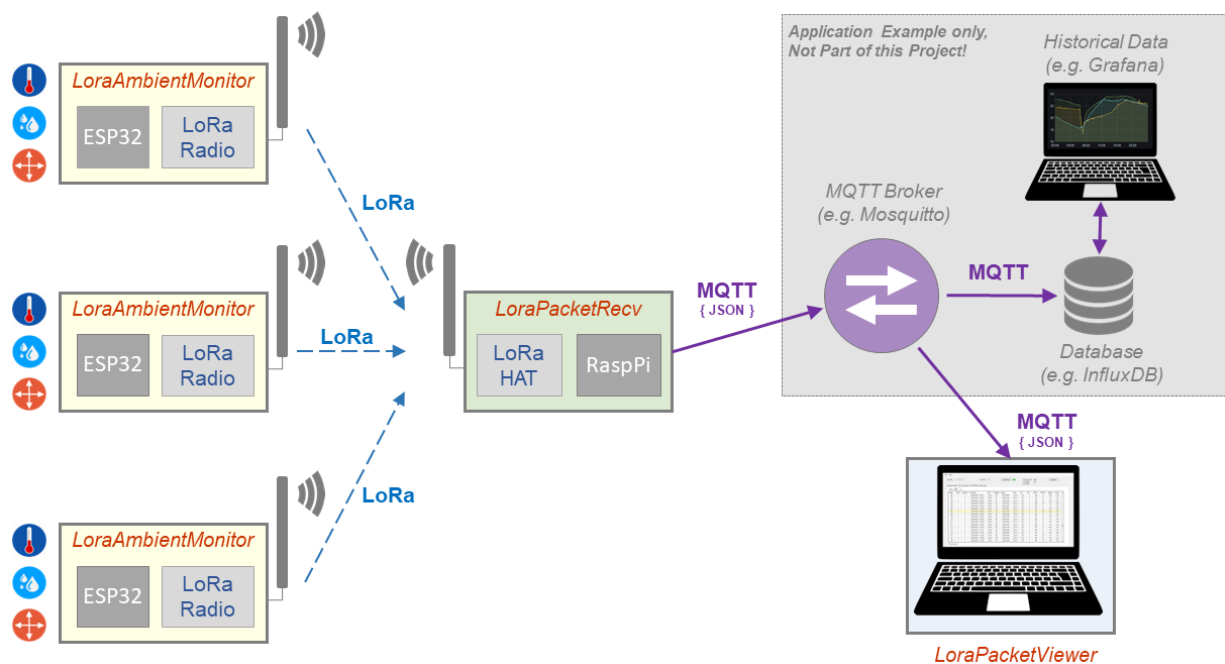
Das Projekt *LoraAmbientMonitor* verfolgt das Ziel, Umgebungsdaten in Nebengebäuden wie z.B. Auto-Garagen zu monitoren und mittels LoRa-Funktechnik zum Hauptgebäude zu übertragen. Die LoRa-Kommunikation erfolgt dabei direkt Punkt-zu-Punkt zwischen Sender und Empfänger (Device-to-Device), wodurch die erreichbare Entfernung auf einige 100 Meter limitiert ist.

Im Projekt *LoraAmbientMonitor* werden folgende Umgebungsdaten erfasst:

- Temperatur
- Luftfeuchtigkeit
- Bewegungen im Raum (PIR-Sensor)
- Umgebungshelligkeit
- optional: Spannung der KFZ-Startbatterie

Das Anwendungsfeld des Projektes *LoraAmbientMonitor* ist vergleichbar mit dem der Projekte *<ESP32SmartBoard\_MqttSensors>* bzw. *<ESP32SmartBoard\_MqttSensors\_BleCfq>*. Die beiden letztgenannten Projekte nutzen WLAN für die Datenübertragung, erzielen damit Reichweiten von einigen 10 Metern und sind somit für das Monitoring von Wohnräumen innerhalb eines Gebäudes gut geeignet. Die Umgebungsdaten werden im Minutentakt erfasst und übertragen. Demgegenüber setzt das hier beschriebene Projekt *LoraAmbientMonitor* auf LoRa-Funktechnik für die Datenübertragung und erzielt damit Reichweiten von einigen 100 Metern, jedoch auf Kosten einer signifikant geringeren Erfassungs- und Übertragungsrate im Stundentakt. Somit erweitert das Projekt *LoraAmbientMonitor* den Einsatzradius der beiden *ESP32SmartBoard*-Projekte vom unmittelbaren Wohnbereich auf externe Nebengebäude im mittleren Umfeld.

### Projekt-Überblick



[Project Overview]

Das Projekt *LoraAmbientMonitor* umfasst folgende Komponenten:

### **LoraAmbientMonitor**

- ESP32 basierte Baugruppe mit Sensorik zur Erfassung der Umgebungsdaten (Temperatur, Luftfeuchtigkeit, Bewegungsmelder, Umgebungshelligkeit, Spannung der KFZ-Startbatterie)
- Hardware: siehe Projekt [<LoRaAmbientMonitor\\_PCB>](#)
- Software: Embedded C++ Arduino-Projekt im Verzeichnis [<LoraAmbientMonitor>](#)



[*LoraAmbientMonitor*]

### **LoraPacketRecv**

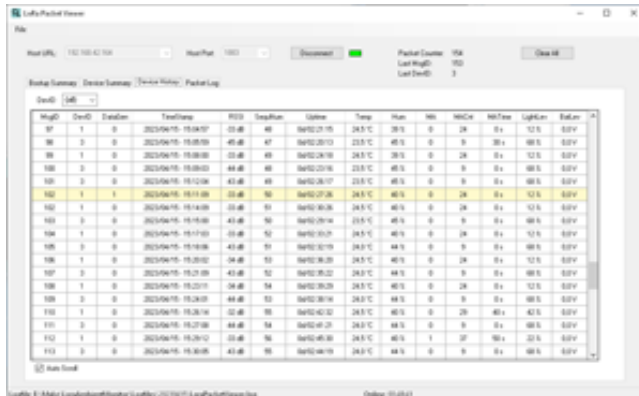
- RaspberryPi basierte Baugruppe mit LoRa GPS HAT
- Gateway, das die von einem oder mehreren *LoraAmbientMonitor* Sensorbaugruppen gesendeten LoRa-Pakete empfängt, deren Payload dekodiert und die Daten als JSON-Record per MQTT an einen zentralen Broker published
- Software: C++ RaspberryPi/Linux-Projekt im Verzeichnis [<LoraPacketRecv>](#)



[*LoraPacketRecv*]

## LoraPacketViewer

- Windows/GUI Applikation zur Beobachtung der von ein oder mehreren *LoraAmbientMonitor* übertragenen Umgebungsdaten, insbesondere für Inbetriebnahme und Diagnose
- Empfängt dazu per MQTT Subscribe die vom *LoraPacketRecv* an den Broker übertragenen JSON-Records
- Software: C#/.NET WindowsForm-Projekt im Verzeichnis <LoraPacketViewer>



ID	DevID	Station	Timestamp	RSSI	SeqNum	Uptime	Temp	Hum	Mo	MoCo	MoTare	LightLm	BatLm
97	1	0	2023/04/15 15:04:17	-21 dB	49	00:02:17:15	24.5 °C	29.5	0	24	0.4	12.5	50.0V
98	2	0	2023/04/15 15:05:00	-45 dB	47	00:02:18:13	23.5 °C	40.5	0	9	0.4	98.5	50.0V
99	1	0	2023/04/15 15:05:00	-23 dB	49	00:02:18:18	24.5 °C	28.5	0	24	0.4	12.5	50.0V
100	2	0	2023/04/15 15:05:03	-44 dB	49	00:02:18:16	23.5 °C	40.5	0	9	0.4	98.5	50.0V
101	2	0	2023/04/15 15:12:04	-43 dB	49	00:02:18:17	23.5 °C	40.5	0	9	0.4	98.5	50.0V
102	1	1	2023/04/15 15:11:08	-23 dB	50	00:02:17:35	24.5 °C	40.5	0	24	0.4	12.5	50.0V
103	1	0	2023/04/15 15:14:09	-23 dB	51	00:02:18:36	24.5 °C	40.5	0	24	0.4	12.5	50.0V
104	2	0	2023/04/15 15:14:00	-43 dB	50	00:02:18:14	23.5 °C	40.5	0	9	0.4	98.5	50.0V
104	1	0	2023/04/15 15:11:03	-23 dB	52	00:02:15:27	24.5 °C	40.5	0	24	0.4	12.5	50.0V
105	2	0	2023/04/15 15:14:06	-43 dB	51	00:02:18:19	24.5 °C	44.5	0	9	0.4	98.5	50.0V
106	1	0	2023/04/15 15:20:02	-24 dB	53	00:02:18:20	24.5 °C	40.5	0	24	0.4	12.5	50.0V
107	2	0	2023/04/15 15:21:09	-43 dB	52	00:02:18:22	24.5 °C	44.5	0	9	0.4	98.5	50.0V
108	1	0	2023/04/15 15:22:11	-24 dB	54	00:02:18:20	24.5 °C	40.5	0	24	0.4	12.5	50.0V
109	2	0	2023/04/15 15:24:05	-44 dB	53	00:02:18:14	24.5 °C	44.5	0	9	0.4	98.5	50.0V
110	1	0	2023/04/15 15:26:14	-22 dB	55	00:02:14:32	24.5 °C	40.5	0	24	40.4	42.5	50.0V
111	0	0	2023/04/15 15:27:08	-44 dB	54	00:02:14:23	24.5 °C	44.5	0	9	0.4	98.5	50.0V
112	1	0	2023/04/15 15:29:12	-23 dB	56	00:02:14:30	24.5 °C	40.5	1	27	100.4	22.5	50.0V
113	2	0	2023/04/15 15:30:26	-43 dB	55	00:02:14:19	24.5 °C	44.5	0	9	0.4	98.5	50.0V

[LoraPacketViewer]

**Nicht mehr Bestandteil dieses Projektes** ist die weitergehende Verarbeitung der an den MQTT-Broker übertragenen Daten der Sensorbaugruppen (JSON-Records). Die Anzeige mit dem *LoraPacketViewer* dient in erster Linie nur der Inbetriebnahme und Diagnose, sie ermöglicht jedoch keine Auswertung und ist nicht für die permanente Speicherung von Umgebungsdaten konzipiert. Im Produktiveinsatz können hier z.B. über einen *Node-RED* Flow die Daten vom MQTT-Broker empfangen und in eine Influx-Datenbank geschrieben werden. Von dort lassen sie dann via *Grafana* auslesen, auswerten und visualisieren. Der Scope dieses Projektes endet jedoch bei der Übergabe der Daten an den MQTT-Broker durch *LoraPacketRecv*. Die Speicherung in einer Datenbank und weitergehende Auswertungen liegen außerhalb des Scopes dieses Projektes (in der Projektübersicht oben grau markiert).

Wie in der Projektübersicht oben dargestellt, wird eine direkte LoRa Device-to-Device Kommunikation zwischen den Sensorbaugruppen und dem RaspberryPi basierten *LoraPacketRecv* Empfänger verwendet. Dabei kommt ein proprietäres, im Nachfolgenden detailliert beschriebenes Protokoll zum Einsatz. Auf die Nutzung von LoRaWAN als Protokoll wurde in diesem Projekt bewusst verzichtet, da es im Umfeld des geplanten Einsatzgebietes keine bereits installierten LoRaWAN-Gateways gibt. Durch die direkte LoRa Device-to-Device Kommunikation kann unabhängig von vorhandener LoRaWAN-Infrastruktur das Projekt überall dort umgesetzt werden, wo direkter Funkkontakt zwischen Sender- und Empfängerbaugruppen gegeben ist.

## LoRa-Funkübertragung

Die kritischste Strecke des Gesamtsystems ist die Funkübertragung der Daten über LoRa. Eine Funkstrecke ist per se stark fehleranfällig, da diese zahlreichen Störquellen ausgesetzt ist (Shared Medium: mehrere Stationen können alle gleichzeitig senden und sich damit gegenseitig stören, elektromagnetische Störungen aus der Umwelt usw.).

Die folgenden Faktoren haben signifikanten Einfluss auf die Resilienz der per Funk übertragenen Daten gegenüber Störfaktoren:

- Kurze Übertragungsdauer

Je kürzer die Übertragungszeit eines Datenpaketes over-the-air ist, desto geringer ist die Wahrscheinlichkeit, dass es während der Übertragung zu Kollisionen mit anderen Paketen oder zu elektromagnetischen Störungen aus der Umwelt kommt. Eine kurze Übertragungszeit werden insbesondere durch ein geringes Payload-Volumen (Datensparsamkeit) und eine hohe Datenrate (Übertragungsgeschwindigkeit) erreicht.

- Hohe Energiedichte

Je höher die Energiedichte der übertragenen Daten ist, umso besser kann der Empfänger die Daten aus dem allgemeinen Rauschteppich extrahieren (Signal-Rausch-Abstand). Entsprechend führt das auch zu einer Erhöhung der per Funkstrecke überbrückbaren Distanz (Reichweite).

Ebenfalls hohen Einfluss auf die erreichbare Qualität der Funkübertragung haben die jeweils verwendeten Antennen. Diese sollten konstruktiv optimal auf das Frequenzband von 868 MHz abgestimmt sein (Antennengewinn), Sende- und Empfangsantenne sollten harmonisch zueinander ausgerichtet werden und im besten Fall sollte zwischen ihnen eine direkte Sichtverbindung ohne abschottende Hindernisse bestehen.

Die dem *Heltec ESP32 Kit* (Sensorbaugruppe) sowie die dem *LORA PI HAT* (Empfänger) beiliegenden Stick Antennen eignen nur zur Überbrückung kurzer Entfernungen. Zur Erhöhung der Reichweite sollte im Produktiveinsatz besser auf beiden Seiten jeweils eine mit Koaxialkabel abgesetzte externe Antenne (6dBi oder besser) verwendet werden.

Die für LoRa-Übertragung relevanten Konfigurationsparameter werden auf der Senderseite in <LoRaAmbientMonitor.ino> definiert:

- Spreadingfaktor (*LORA\_SPREADING\_FACTOR*)

Der Spreadingfaktor ist ein Maß, das angibt, in wievielen Einzelschritten (Chips) ein Datensymbol übertragen wird. Bei einem kleinen Spreadingfaktor werden die Datensymbole mit wenigen Chips kodiert, was letztendlich zu einer kurzen Übertragungszeit eines Datenpaketes over-the-air führt. Damit verringert sich die Wahrscheinlichkeit, dass das Datenpaket während der Übertragung durch externe Einflüsse gestört wird. Umgekehrt bewirkt ein großer Spreadingfaktor, dass ein Datensymbol mit einer hohen Anzahl Chips kodiert und damit zeitlich gestreckt wird. Hierdurch wird jedes Datensymbol mit einer höheren Energiedichte übertragen, was wiederum zu einer erhöhten Störfestigkeit durch einen besseren Signal-Rausch-Abstand führt. Im Allgemeinen resultieren aus einem höheren Spreadingfaktor eine höhere Reichweite, aber eben auch eine höhere Wahrscheinlichkeit von Störbeeinflussungen durch die längere over-the-air Übertragungszeit. Da sich die beiden Aspekte gegenläufig zueinander beeinflussen, kann es bei häufigen Übertragungsfehlern sinnvoll sein, den für die jeweiligen lokalen Gegebenheiten optimal Wert empirisch ermittelt werden.

- Signalbandbreite (*LORA\_SIGNAL\_BANDWIDTH*)

Die Signalbandbreite ist ein Maß für den Frequenzhub, mit dem die Datensymbole kodiert werden. Ähnlich wie der Spreadingfaktor wirkt sich auch die Signalbandbreite auf die zur Übertragung eines Datenpaketes benötigte Zeit aus. Eine Erhöhung der Signalbandbreite bewirkt eine Verkürzung der Übertragungszeit eines Datenpaketes. Damit verringert sich die Wahrscheinlichkeit, dass das Datenpaket während der over-the-air Übertragung durch externe Einflüsse gestört wird. Demgegenüber bewirkt eine Verringerung der Signalbandbreite, dass jedes Datenelement während der Übertragung zeitlich gestreckt wird. Das wiederum bewirkt eine höhere Störfestigkeit durch einen besseren Signal-Rausch-Abstand. Da sich die beiden Aspekte gegenläufig zueinander verhalten, kann es bei häufigen Übertragungsfehlern auch hier sinnvoll sein, den für die jeweiligen lokalen Gegebenheiten optimal Wert empirisch ermittelt werden.

- Sendeleistung (*LORA\_TX\_POWER*)

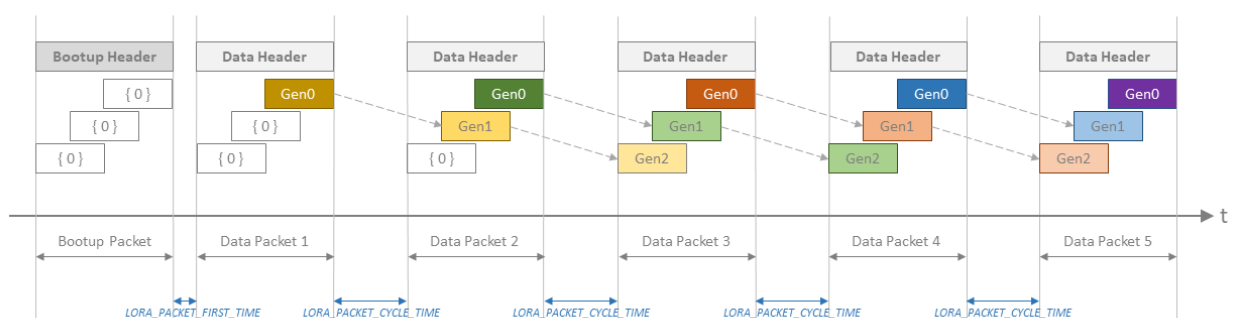
Eine hohe Sendeleistung bewirkt einen hohen Signal-Rausch-Abstand, damit eine hohe Resilienz gegenüber Störfaktoren und führt letztendlich somit auch zu einer hohen erreichbaren Übertragungreichweite. Allerdings bedeutet eine hohe Sendeleistung ein unkooperatives Verhalten gegenüber anderen Nutzern. Aus diesem Grund ist die maximale Sendeleistung daher regulatorisch auf 20 dB begrenzt.

Neben den aufgeführten Sendeparametern ist die zu übertragende Datenmenge (Payload-Volumen) von elementarer Bedeutung. Ein großes Payload-Volumen bedingt eine entsprechend lange Belegung des Sendekanals, der während dieser Zeit anderen Nutzern nicht zur Verfügung steht (Shared Medium). Gleichzeitig vergrößert eine lange over-the-air Übertragung die Anfälligkeit gegenüber Störungen. Daher sollte das Payload-Volumen so gering wie möglich sein.

Aus diesem Grund werden die einzelnen Datenelemente auf Senderseite in ein möglichst kompaktes Format überführt. So wird beispielsweise die Temperatur nur 0.5°-Schritten übertragen (statt der 0.1° Auflösung des Sensors), die Luftfeuchtigkeit nur als Ganzzahlwert, die Umgebungshelligkeit nur in 2%-Schritten usw. Durch die reduzierte Auflösung ist es möglich, den kompletten Sensordatensatz inklusive minimaler Headerinformationen in einem 64Bit-Feld vom Typ *uint64\_t* abzulegen. Der Sensordatensatz wird durch eine zusätzliche 16Bit CRC vom Typ *uint16\_t* gesichert. Somit ergibt sich für einen Datensatz inklusive CRC eine Gesamtgröße von 80Bit (= 10 Bytes).

Um die Wahrscheinlichkeit für mindestens einen erfolgreichen Empfang zu erhöhen, wird jeder Sensordatensatz in jeweils 3 aufeinanderfolgenden LoRa-Paketen (mit einem zeitlichen Abstand von *LORA\_PACKET\_CYCLE\_TIME* Sekunden) übertragen. Der mit *kLoraPacketDataGen0* gekennzeichnete Datensatz enthält die jeweils zum Sendezeitpunkt aktuellen Daten, gefolgt von *kLoraPacketDataGen1* mit den Daten vom vorherigen Sendezeitpunkt sowie *kLoraPacketDataGen2* mit den Daten des vorletzten Sendezeitpunktes. Somit lassen sich selbst bei einem Verlust von 2 aufeinanderfolgenden LoRa-Paketen noch alle Daten lückenlos rekonstruieren.

Um Paketverluste auf Empfängerseite erkennen zu können, beinhaltet jedes LoRa-Sensordatenpaket in seinem Header eine fortlaufende 24Bit Sequenz-Nummer (*uint24SeqNum*). Bei fehlerfreiem Empfang bildet diese Sequenz-Nummer eine streng monoton steigende Folge. Ein Paketverlust führt zu Sprüngen auf Empfängerseite.



### [LoRa Packet Scheme]

Ein LoRa-Paket hat eine Payload-Länge von 40 Bytes. Es besteht aus einem Paket-Header und den oben genannten 3 Generationen von Sensordatensätzen (Gen0/Gen1/Gen2). Jedes dieser vier Elemente ist jeweils mit seiner eignen 16Bit CRC vom Typ *uint16\_t* gesichert und besitzt eine Länge von 10 Bytes. Um einen Sensordatensatz zu dekodieren, müssen jeweils die CRC des Headers und des jeweiligen Sensordatensatzes (Gen0, Gen1 oder Gen2) intakt sein. Durch diese Segmentierung können auch dann noch einzelne Sensordatensätze aus einem LoRa-Paket extrahiert werden, wenn das Paket partielle Übertragungsdefekte aufweist. Dies erhöht die Resilienz gegenüber Störungen auf der Funkstrecke. Den detaillierten Aufbau eines vollständigen LoRa-Paketes beschreibt [<LoRa Payload Design.pdf>](#).

Das Projekt *LoraAmbientMonitor* ist so angelegt, dass Umgebungsdaten parallel von bis zu 16 Sensorbaugruppen erfasst und übertragen werden können. Senden dabei mehrere Baugruppen gleichzeitig, führt dies zu Kollisionen auf der Funkstrecke und damit in der Regel zum Verlust der Datenpakete. Da typischerweise für alle Geräte ein einheitlicher Firmwarestand verwendet wird, nutzen auch alle Geräte dasselbe Sendezeitregime basierend auf den Zeitkonstanten *LORA\_PACKET\_INHIBIT\_TIME*, *LORA\_PACKET\_FIRST\_TIME* und *LORA\_PACKET\_CYCLE\_TIME* (alle definiert in *<LoRaAmbientMonitor.ino>*). Um zu verhindern, dass die identische Konfiguration zu permanenten, systematischen Sende-Kollisionen führt, wird der jeweilige Sendezeitpunkt durch einen zufälligen Wert im Bereich +/- 5% variiert.

In Summe muss das Gesamtergebnis der hier beschriebenen Konstanten den regulatorischen Vorgaben an den Duty Cycle zur Nutzung des 868 MHz-Bandes genügen. Hierbei ist festgelegt, dass ein Sender das Frequenzband nur für maximal 1% der Zeit belegen darf, um die kooperative Nutzung durch andere Teilnehmer nicht zu stören. Das bedeutet, dass der *LoraAmbientMonitor* den Funkkanal nur für maximal 36 Sekunden je Stunde nutzen darf. Wie diskutiert, haben sowohl Sendeparameter als auch Payload-Länge einen signifikanten Einfluss auf die für die over-the-air Übertragung des LoRa-Paketes benötigte Zeit. Diese lässt sich anhand der Einzelparameter mit Hilfe des "LoRa Air time calculator" ermitteln: <https://loratools.nl/#/airtime>.

## Aufbau der LoRa-Pakete

Den Aufbau der LoRa-Pakete für die over-the-air Übertragung beschreibt die Headerdatei *<LoraPacket.h>*, die für Sender (*LoRaAmbientMonitor*) und Empfänger (*LoraPacketRecv*) identisch sein muss.

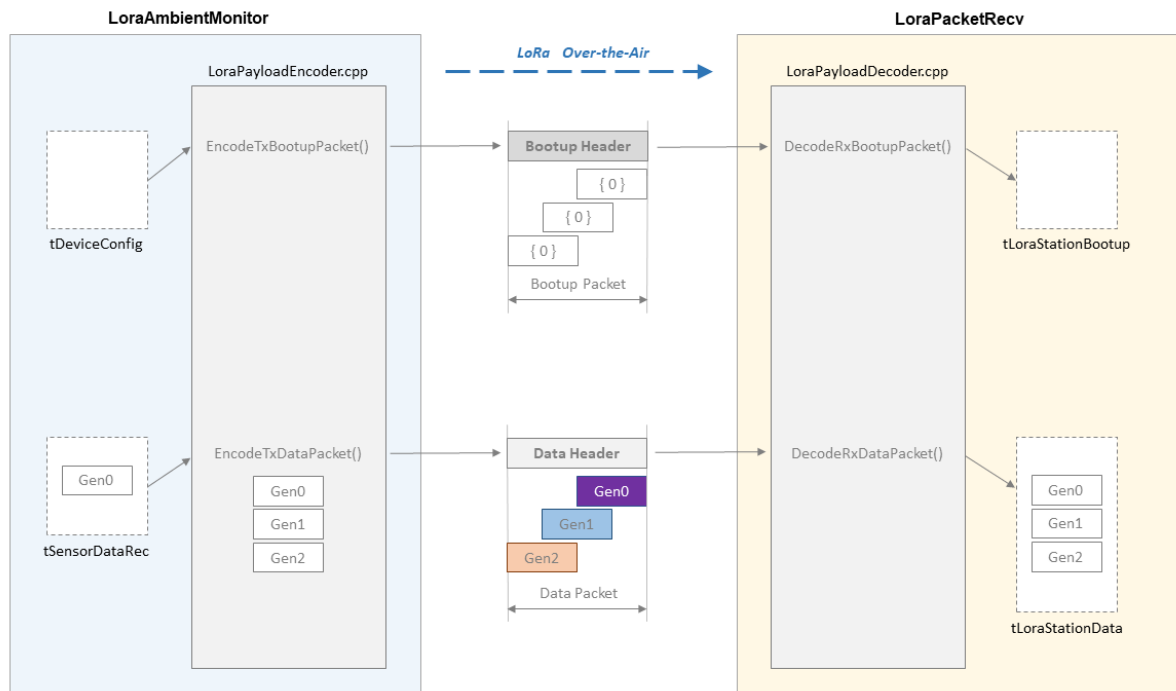
Hinweis: Da die Arduino IDE keine Dateien außerhalb des Sketch-Verzeichnisses verarbeiten kann, ist es nicht möglich, die Headerdatei *LoraPacket.h* zentral in einem gemeinsamen Include-Verzeichnis für beide Teilprojekte *LoRaAmbientMonitor* und *LoraPacketRecv* abzulegen (siehe <https://stackoverflow.com/questions/68733232/define-a-relative-path-to-h-file-c-arduino>). Stattdessen nutzen beide Teilprojekte jeweils separate Kopien von *LoraPacket.h*.

Den vollständigen Aufbau eines over-the-air übertragenen Datenpaketes mit 40 Bytes Länge definiert die Struktur *tLoraDataPacket*. Sie enthält einen Header und wie oben beschrieben 3 Generationen von Sensordatensätzen (Gen0/Gen1/Gen2).

Als erstes Paket nach dem System wird zunächst einmalig ein Bootup-Paket gesendet, dessen Paket-Header durch die Struktur *tLoraBootupHeader* definiert ist. Die 3 Segmente vom Typ *tLoraDataRec* enthalten keine Nutzdaten und sind komplett mit Null belegt. Das Bootup-Paket beschreibt die aktuelle Geräte-Konfiguration (Parametrierung durch 4-fach DIP-Schalter SW2, Firmware-Version, statische Firmware-Konfiguration, LoRa-Sendeparameter).

Alle nachfolgenden LoRa-Pakete bestehen wie oben beschrieben aus einem Paket-Header (*tLoraDataHeader*) und den 3 Generationen von Sensordatensätzen (Gen0/Gen1/Gen2), definiert durch den Typ *tLoraDataRec*.

Auf Sender-Seite (*LoRaAmbientMonitor*) erfolgt die Kodierung der nativen Sensordaten in die für die over-the-air Übertragung verwendete Struktur *tLoraDataPacket* durch die Klasse *LoraPayloadEncoder*. Das Komplement für die Dekodierung auf Empfängerseite (*LoraPacketRecv*) bildet die Klasse *LoraPayloadDecoder*.



[LoRa Payload Encode/Decode]

## Timestamp-Erzeugung für Sensordatensätze

Der ESP32 des *LoRaAmbientMonitor* stellt nur eine relative Uptime bereit, d.h. die Anzahl der Sekunden seit dem Systemstart. Eine absolute Echtzeit mit Datum und Uhrzeit (RTC) ist nicht verfügbar.

Die vollständige Uptime ist als 32Bit-Wert im Header eines LoRa-Paketes abgelegt ( $2^{32}$  Sekunden => 136 Jahre). Sie definiert den Sendezeitpunkt des LoRa-Paketes auf dem *LoRaAmbientMonitor*. In jedem Sensordatensatz ist eine verkürzte Form der Uptime als 12Bit-Wert in 10 Sekunden Schritten enthalten ( $2^{12} = 4096 * 10$  Sekunden => 11 Stunden).

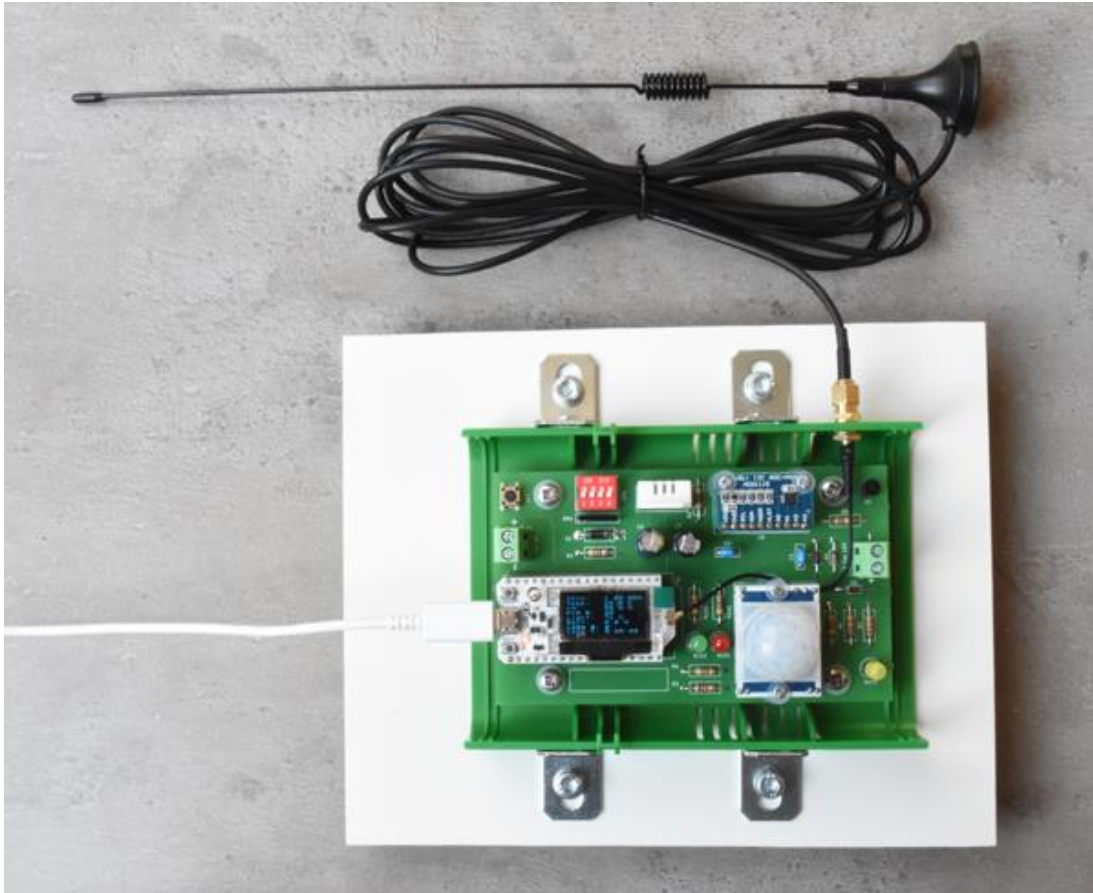
Der Echtzeit-Timestamp mit Datum und Uhrzeit wird erst auf der Empfängerseite (RaspberryPi) vom *LoraPacketRecv* erzeugt. Dazu wird im Datenpaket vom Typ *tLoraMsgData* die Localtime des RaspberryPi zum Zeitpunkt des LoRa-Datenempfangs eingetragen (*tLoraMsgData.m\_tmTimeStamp*). Dieser Empfangs-Timestamp wird unmittelbar als Timestamp für den Gen0 Sensordatensatz übernommen. Für die historischen Sensordatensätze Gen1 und Gen2 wird der Timestamp rekonstruiert, indem der aktuelle Empfangs-Timestamp jeweils um die Uptime-Differenz zwischen Gen0 und Gen1 bzw. zwischen Gen0 und Gen2 verringert wird.

## Best Practice (1): Montage der *LoRaAmbientMonitor* Sensorbaugruppe

Für ein optimales Monitoring der Umgebung sollte die Sensorbaugruppe so im Raum positioniert sein, dass der Bewegungsmelder (PIR-Sensor) das Öffnen bzw. Schließen des Garagentores direkt erfasst sowie Temperatur und Luftfeuchtigkeit etwa in Raummitte gemessen werden. Dazu eignet sich eine ABS-Kunststoffplatte (ca. 16x20cm), auf der 4 Montagewinkel (mit Langloch) so angeordnet werden, dass sich die Sensorbaugruppe zwischen ihnen festklemmen lässt. Die Schrauben für die Montagewinkel fixieren gleichzeitig auf der Platten-Rückseite ein flexibles Montageband. Dieses kann leicht abgewinkelt und zwischen Deckenbalken und Deckenplatte hindurchgeschoben werden. Durch den Biegewinkel des Montagebandes lässt sich die Sensorbaugruppe optimal im Raum ausrichten, so dass diese annähernd senkrecht von der Decke herabhängt.



Zum Senden der Datenpakete wird eine mit Koaxialkabel abgesetzte Stabantenne (6 dBi) verwendet. Dies erlaubt eine flexible Wand-/Decken-Montage und optimale Ausrichtung unabhängig von der Sensorbaugruppe.



[LoraAmbientMonitor\_Mounted]

## Best Practice (2): Inbetriebnahme des Systems

Aus reiner Softwaresicht ist die Inbetriebnahme des Systems relativ trivial. Eine gewisse Herausforderung bereitet jedoch die optimale Positionierung und Ausrichtung der Antennen. Hierzu empfiehlt es sich, mehrere Tests an verschiedenen Stellen im Raum durchzuführen. Um beim Positionieren der Sendeantenne unmittelbar vor Ort ein Feedback über das Empfangsergebnis zu erhalten, hilft die Anzeige der Telemetriedaten auf einem Mobilgerät (Smartphone). Dazu wird temporär ein öffentlich zugänglicher MQTT-Broker genutzt, der auch vom Smartphone aus über eine mobile Internetverbindung erreichbar ist (z.B. "*public.mqtthq.com*"):

### Schritt 1 – Start *LoraPacketRecv* (Empfänger)

Die *LoraPacketRecv* Software wird mit den Kommandozeilenparametern "*-h=<host>*" und "*-t*" angewiesen, sich mit einem öffentlich zugänglichen Broker zu verbinden und diesem kompakte, für die Anzeige auf dem Smartphone gut geeignete Telemetriedaten-Pakete zu senden:

```
sudo ./LoraPacketRecv -h=public.mqtthq.com -t
```



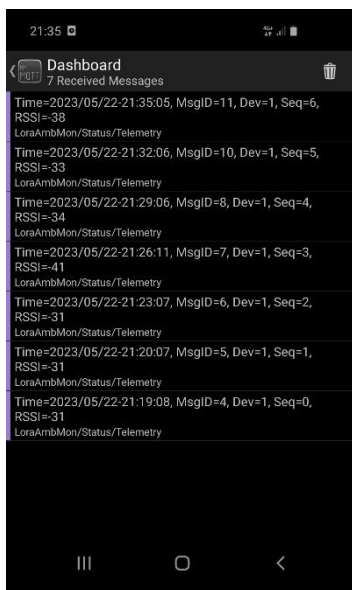
## Schritt 2 – Start MQTT Client App auf dem Mobilgerät

Auf dem Mobilgerät wird eine MQTT Client App benötigt, wie z.B. *"MyMQTT"*. Diese App ist ebenfalls mit demselben öffentlich zugänglichen Broker zu verbinden wie *LoraPacketRecv*. Anschließend ist das Topic *"LoraAmbMon/Status/Telemetry"* zu abonnieren (subscribe). Dadurch werden die von *LoraPacketRecv* beim Empfang eines LoRa-Paketes generierten Telemetriedaten angezeigt. Diese enthalten unter anderem folgende Informationen:

**Sequenz-Nummer:** fortlaufende Paketnummer, fehlende Nummern sind ein Anzeichen für den Verlust der betreffenden Pakete

**RSSI-Wert:** Signalstärke des empfangenen Paketes, je größer der RSSI-Wert, umso besser ist die Qualität des empfangenen Signals.

Hinweis: RSSI-Werte sind negative Zahlen, die Signalstärke ist also umso besser, je kleiner der Zahlenwert ist (-30dBi sind besser als -50dBi).



[MyMQTT App]

## Schritt 3 – Start Sensorbaugruppe

Insbesondere für Inbetriebnahme und Diagnose ist der *"Commissioning Mode"* vorgesehen, der mit verkürzten LoRa-Sendeintervallen arbeitet. Durch das häufigere Senden der LoRa-Pakete können Maßnahmen wie das Verändern der Antennen-Position bzw. -Ausrichtung schneller bewertet werden.

Um die Sensorbaugruppe im *"Commissioning Mode"* zu starten, muss beim Booten (Power-on oder Reset) der CFG-Taster gedrückt sein. Der *"Commissioning Mode"* wird durch ein permanentes Blinken der grünen LED angezeigt.

Durch Beobachtung des auf dem Mobilgerät zusammen mit den Telemetriedaten ausgegebenen RSSI-Wertes lassen sich nun komfortabel verschiedene Positionen und Ausrichtungen der Sendeantenne ausprobieren und bewerten.

Hinweis: Neben den Telemetriedaten werden weiterhin auch alle anderen JSON Records mit Bootup- und Sensordaten-Paketen an den MQTT-Broker übertragen. Für die Zeit der Verbindung zu einem öffentlichen Broker sind damit auch die vollständigen Umgebungsdaten öffentlich zugänglich.

### Best Practice (3): Produktiveinsatz

Für den Produktiveinsatz sind eine oder mehrere *LoraAmbientMonitor* Sensorbaugruppen sowie als Empfänger ein RaspberryPi mit der *LoraPacketRecv* Gateway Software erforderlich. Die Windows-Applikation *LoraPacketViewer* unterstützt bei Bedarf Inbetriebnahme und Diagnose, ist aber für den Produktiveinsatz selbst nicht notwendig.

- ***LoRaAmbientMonitor* Sensorbaugruppe:**

Beim Einsatz mehrerer *LoraAmbientMonitor* Sensorbaugruppen ist darauf zu achten, dass für jede Baugruppe an den DIP-Schaltern **DIP3/4** eine individuelle Node- bzw. DevID konfiguriert wird. **DIP2** sorgt optional dafür, dass während des Sendens der LoRa-Pakete der Bewegungsmelder ignoriert wird, um Fehlauflösungen durch Störbeeinflussung durch den LoRa-Transmitter zu unterdrücken. Über **DIP1** lässt sich steuern, ob neben den zyklischen LoRa-Paketen zusätzlich noch asynchrone Pakete bei besonderen Ereignissen wie dem Auslösen des Bewegungsmelders oder dem An- bzw. Abklemmen KFZ-Startbatterie gesendet werden.

- ***LoraPacketRecv* Gateway:**

Die Gateway-Software sollte beim Booten des RaspberryPi automatisch mitgestartet werden, um so eine hohe Verfügbarkeit zu gewährleisten. Für das Start-/Stop-Handling sind die im Unterverzeichnis "*LoraPacketRecv/Autostart*" enthaltenen Skripte zuständig. Mit dem Skript "*LoraPacketRecv*" wird der Autostart der Gateway-Software konfiguriert. Details beschreibt der Abschnitt "*Autostart für LoraPacketRecv*" in der Dokumentation [<LoraAmbientMonitor\\_LoraPacketRecv>](#).

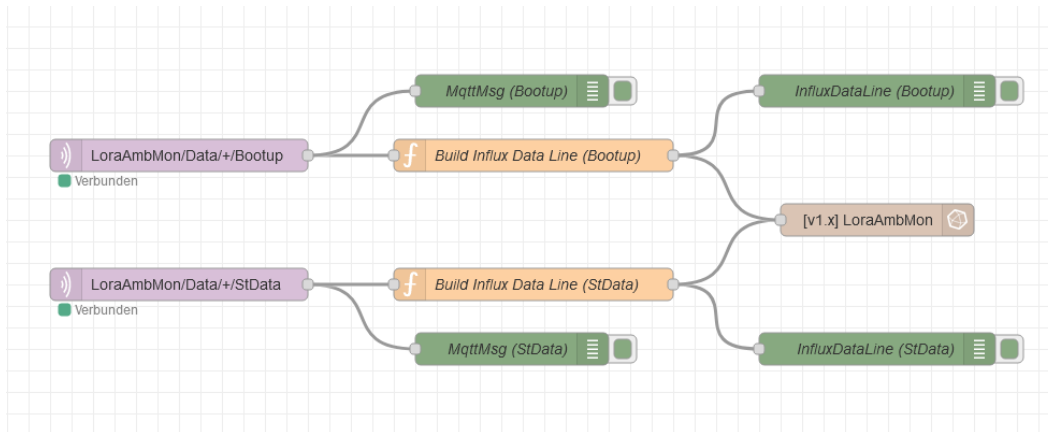
Durch die Bereitstellung der Sensordatenpaket in Form von JSON-Records über MQTT ist eine vielfältige Weiterverarbeitung und Auswertung mit zahlreichen freien Softwarepaketen möglich. So lässt sich beispielsweise der "*Eclipse Mosquitto MQTT Broker*" direkt auf demselben RaspberryPi mit installieren, der auch als *LoraPacketRecv* Gateway fungiert. Als weitere Softwarepakete für Weiterverarbeitung und Auswertung bieten sich *Node-RED*, *InfluxDB* und *Grafana* an. Dabei ist es unerheblich, ob die Pakete ebenfalls mit auf dem RaspberryPi des *LoraPacketRecv* Gateway laufen, oder auf verschiedenen Systemen.

Mit den genannten Paketen ist es möglich, die Sensordatenpakete in *Node-RED* mit Hilfe eines MQTT Input Nodes einzulesen und über einen *InfluxDB* Output Node in eine *InfluxDB* zu speichern. In *Node-RED* lassen sich aktuelle Sensorwerte zudem übersichtlich in einem Dashboard anzeigen. Für die Darstellung der zeitlichen Verläufe in Form von Charts bietet sich hingegen *Grafana* an, das dazu auf die in der *InfluxDB* gespeicherten Daten zurückgreift.

#### Best Practice (4): Archivierung der Daten in einer Datenbank

Die Windows-Applikation *LoraPacketViewer* dient in erster Linie der Inbetriebnahme und Diagnose, sie ermöglicht jedoch keine persistente Aufzeichnung und damit auch keine Auswertung der Umgebungsdaten. Dazu ist es notwendig, die Daten dauerhaft in einer Datenbank, wie z.B. einer *InfluxDB*, zu speichern.

Für eine simple Umsetzung bietet sich *Node-RED* an, da es sowohl Nodes zum Empfangen von MQTT-Datenpaketen als auch zum Schreiben von Daten in eine *InfluxDB* unterstützt. Die Konvertierung der Sensordaten aus den per MQTT übertragenen JSON-Records in das *InfluxDB Line Format* erfolgt durch einfachen JavaScript Code innerhalb eines User Function Nodes. Eine einfache Umsetzung in Form eines vollständigen Node-RED Flows zeigt [<Flow Mqtt InfluxDB.json>](#).



[Node-RED Flow MQTT to InfluxDB]