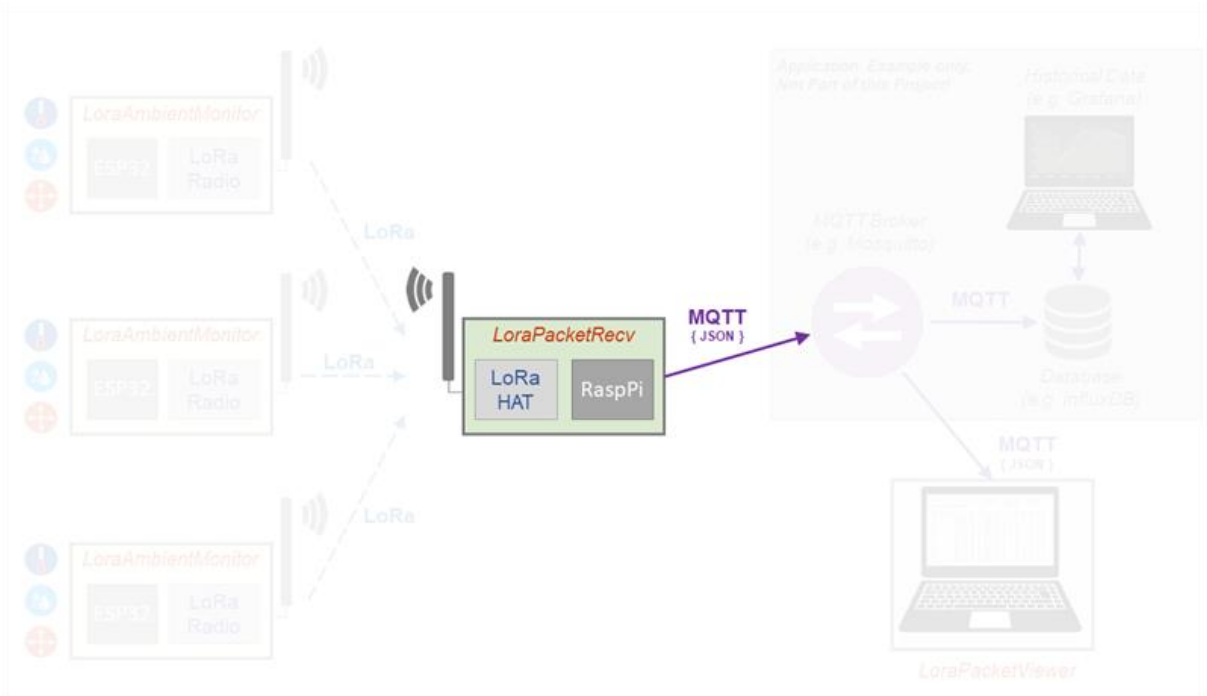
Projekt: https://github.com/ronaldsieber/LoraAmbientMonitor
Teilprojekt: https://github.com/ronaldsieber/LoraAmbientMonitor/LoraPacketRecv
Lizenz: MIT
Autor: Ronald Sieber

# LoraPacketRecv

Dieses Linux-Projekt ist Teil des Hauptprojektes *LoraAmbientMonitor* und umfasst die RaspberryPi Konsolen-Applikation für die Empfängerbaugruppe. Diese Gateway-Software empfängt die von einem oder mehreren im Teilprojekt <*LoraAmbientMonitor*> beschriebenen Sensorbaugruppen gesendeten LoRa-Pakete, dekodiert den Payload und published die Daten als JSON-Record per MQTT an einen zentralen Broker. Zum Empfang der LoRa-Daten wird der *SEEED LORA/GPS RASPBERRY PI HAT* benötigt.

Die dem *LORA PI HAT* beiliegende Stick Antenne eignet sich aufgrund ihrer kompakten und platzsparenden Ausführung gut für Entwicklung und kurze Entfernungen. Zur Erhöhung der Reichweite sollte im Produktiveinsatz besser eine mit Koaxialkabel abgesetzte externe Antenne (6dBi oder besser) verwendet werden.

Die Software wurde auf einem Raspberry Pi 3 Model B+ entwickelt und getestet. Die hier abgelegten Sourcen lassen sich direkt auf dem RaspberryPi durch Aufruf des Kommandos *"make"* übersetzen.




*[Project Overview - LoraPacketRecv]*

**Ausführung der Software**

Die *LoraPacketRecv* Software benötigt root-Rechte, um auf die Hardware zugreifen zu können. Sie muss daher im *"sudo"* Modus gestartet werden. Folgende Kommandozeilen-Optionen werden unterstützt:

- *-h=<host_url>*
  Host URL des MQTT Broker im Format *URL[:Port]*

- *-l=<msg_file>*
  Logging aller an den MQTT-Broker gesendeten JSON-Records in die angegebene Datei (die Logdatei wird immer im APPEND Mode geöffnet). Die Logdatei kann später mit Hilfe der im Teilprojekt *<LoraPacketViewer>* realisierten GUI-Applikation angezeigt und ausgewertet werden.

- *-a*
  Weiterleitung von JSON-Records für alle empfangenen LoRa-Pakete an den MQTT-Broker, inklusive aller evtl. Duplikate (Gen0/Gen1/Gen2)

- *-t*
  Zusätzlich zu den JSON-Records werden noch kompakte Telemetrie Messages an den MQTT-Broker publiziert, die insbesondere Inbetriebnahme und Diagnose unterstützen (siehe Sektion *"MQTT-Kommunikation"*)

- *-o*
  Offline Modus, ohne Verbindung zum MQTT-Broker

- *-v*
  Verbose Modus mit detaillierten Protokollausgaben

- *--help*
  Anzeige Hilfeseite und der Default-Konfiguration für Host und Portnummer des MQTT-Brokers

Beispiele:

```
sudo  ./LoraPacketRecv  -h=127.0.0.1  -l=./LoraPacketLog.json  -v
```

Mit Hilfe des Linux-Kommandos *"tee"* ist es möglich, die im Terminalfenster angezeigten Ausgaben parallel zur Echtzeitausgabe noch in eine Logdatei zur späteren Auswertung zu speichern:

```
sudo  ./LoraPacketRecv  -h=127.0.0.1  -l=./LoraPacketLog.json  -v  |  tee >
./LoraPacketRecv.log
```

**LoRa Datenempfang**

Der für den Empfang der LoRa-Daten verwendete *SEEED LORA/GPS RASPBERRY PI HAT* kommuniziert mit dem RaspberryPi via SPI-Schnittstelle. Softwareseitig wird die *"RadioHead"* Treiberbibliothek verwendet, um den SX1276 LoRa-Chip zu initialisieren und empfangene Datenpakete auszulesen. Die Treiberbibliothek wird im *LoraPacketRecv* durch das File *LibRf95.cpp* gekapselt.

Den Empfang eines LoRa-Paketes signalisiert der *LORA/GPS HAT* durch eine fallende Flanke an GPIO25 des RaspberryPi. Das Interrupt-Handling des im SysFS abgebildeten GPIO ist vollständig in *GpioIrq.cpp* gekapselt. Innerhalb der Main-Loop ist GPIO25 mit dem File-Descriptor `struct pollfd FdSet[1]` verknüpft. Die Linux Poll-Funktion (`poll(FdSet, 1, 1000)`) wartet auf ein Signalisierungs-Event (LoRa-Datenempfang), alternativ kehrt sie nach dem Timeout von 1000 ms wieder zurück. Die Main-Loop wird somit nach dem Empfang eines LoRa-Paketes oder spätestens nach 1 Sekunde ausgeführt, was die CPU-Belastung minimiert, ohne die Main-Loop komplett in den Sleep-Modus zu versetzen.

Ein empfangenes LoRa-Paket wird durch die Funktion `RF95GetRecvDataPacket()` aus dem Empfangspuffer des SX1276 ausgelesen. Die aktuelle Systemzeit wird dem Datenpaket als Empfangs-Timestamp zugeordnet, der Wert der Variable `uiMsgID` wird als Message-ID für das Paket übernommen. Anschließend evaluiert die Funktion `PprGainLoraDataRecord()` das Paket und liefert den dekodierten Payload-Inhalt eines als valid qualifizierten Paketes einer *LoraAmbientMonitor* Sensorbaugruppe in Form der Datenstruktur `tLoraMsgData` zurück.

**Generierung der JSON-Records**

Die Funktion `PprBuildJsonMessages()` überführt die in der Datenstruktur `tLoraMsgData` dekodierten Binärdaten der empfangenen LoRa-Pakete in entsprechende JSON-Records. Dies gilt sowohl für Bootup-Pakete als auch für Sensordaten-Pakete. Bei Letzteren wird aus jeder der 3 Generationen von Sensordatensätzen (`kLoraPacketDataGen0`, `kLoraPacketDataGen1` und `kLoraPacketDataGen2`) zusammen mit Header-Informationen ein separater JSON-Record erzeugt.

Der JSON-Record eines Bootup-Paketes hat folgenden beispielhaften Aufbau:

```
{
  "MsgID": 1,
  "MsgType": "StationBootup",
  "TimeStamp": 1678546265,
  "TimeStampFmt": "2023/03/11 - 15:51:05",
  "RSSI": -37,
  "DevID": 1,
  "FirmwareVer": "1.00",
  "DataPackCycleTm": 3600,
  "CfgOledDisplay": 1,
  "CfgDhtSensor": 1,
  "CfgSr501Sensor": 1,
  "CfgAdcLightSensor": 1,
  "CfgAdcCarBatAin": 1,
  "CfgAsyncLoraEvent": 0,
  "Sr501PauseOnLoraTx": 1,
  "CommissioningMode": 0,
  "LoraTxPower": 20,
  "LoraSpreadFactor": 12
}
```

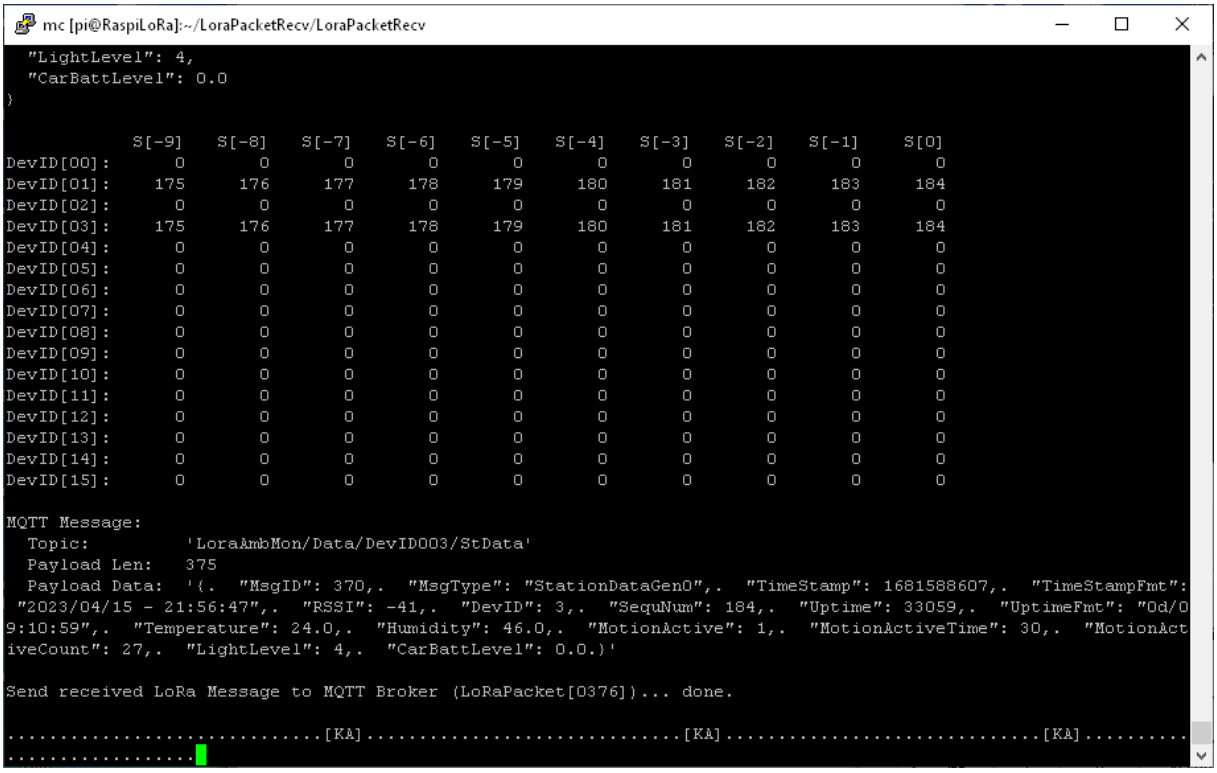
Der JSON-Record eines Sensordaten-Paketes hat folgenden beispielhaften Aufbau:

```
{
  "MsgID": 2,
  "MsgType": "StationDataGen0",
  "TimeStamp": 1678546328,
  "TimeStampFmt": "2023/03/11 - 15:52:08",
  "RSSI": -45,
  "DevID": 1,
  "SequNum": 1,
  "Uptime": 66,
  "UptimeFmt": "0d/00:01:06",
  "Temperature": 21.5,
  "Humidity": 44.0,
  "MotionActive": 0,
  "MotionActiveTime": 10,
  "MotionActiveCount": 1,
  "LightLevel": 8,
  "CarBattLevel": 0.0
}
```

Die Funktion `PprBuildJsonMessages()` liefert die JSON-Records als Vector-Objekt vom Typ `std::vector<tJsonMessage>` zurück. Das Vector-Objekt enthält abhängig vom Typ (`StationBootup`, `StationDataGen0/1/2`) bis zu 3 JSON-Records.

### Verarbeitung der JSON-Records

Sofern *LoraPacketRecv* nicht mit dem Kommandozeilenparameter *"-a"* gestartet wurde, ermittelt die Funktion `MquIsMessageToBeProcessed()` die Relevanz der Übertragung des aktuellen JSON-Records an den MQTT Broker. Records mit *"MsgType"* = *"StationDataGen0"* werden immer übertragen. Records mit *"MsgType"* = *"StationDataGen1"* werden nur dann verarbeitet, wenn das vorangegangene Paket mit den Gen0-Daten nicht empfangen wurde, *"StationDataGen2"* Pakete werden nur dann übertragen, wenn sowohl die Gen0-Daten als auch die Gen1-Daten verloren gingen. Zur Bewertung der Relevanz dient das Array `aui32SequNumHistList_l` (in *MessageQualification.cpp*). In diesem wird für jede *LoraAmbientMonitor* Sensorbaugruppe eine separate Liste der letzten verarbeiteten Records auf Basis ihrer LoRa-Paket Sequenz-Nummer mitgeführt.



[LoraPacketRecv_Console]

Beim Start von *LoraPacketRecv* mit dem Kommandozeilenparameter *"-a"* werden alle JSON-Records an den Broker weitergeleitet, also auch Gen1-Daten und Gen2-Daten von bereits verarbeiteten Daten.

### MQTT-Kommunikation

Als MQTT Client Implementierung für Kommunikation zwischen *LoraPacketRecv* und Broker wird die *"Paho MQTT Embedded/C"* Bibliothek verwendet. Diese wird innerhalb von *LoraPacketRecv* durch das File *LibMqtt.cpp* gekapselt.

Für das Publishen von Nachrichten an den MQTT-Broker sind in *Main.cpp* folgende Topics definiert:

```
const char* MQTT_TOPIC_TMPL_BOOTUP  = "LoraAmbMon/Data/DevID%03u/Bootup";
const char* MQTT_TOPIC_TMPL_ST_DATA = "LoraAmbMon/Data/DevID%03u/StData";
const char* MQTT_TOPIC_TELEMETRY    = "LoraAmbMon/Status/Telemetry";
const char* MQTT_TOPIC_KEEPALVIE    = "LoraAmbMon/Status/KeepAlive";
```

Für das Publishen von Bootup- und Sensordaten-Paketen werden jeweils getrennte Topics verwendet (`MQTT_TOPIC_TMPL_BOOTUP` und `MQTT_TOPIC_TMPL_ST_DATA`). Die Funktion `BuildMqttPublishTopic()` individualisiert das Topic für jedes Sensormodul durch Einfügen der

DevID (die am DIP-Schalter eingestellte Knotennummer). Zudem erfolgt die MQTT-Übertragung mit der Kennzeichnung *"Retain"*, so dass der Broker jeweils die letzte empfange Nachricht eines Topics speichert.

Das bewirkt, dass ein Client wie beispielsweise der zum Hauptprojekt *LoraAmbientMonitor* gehörige *<LoraPacketViewer>* bei seiner Anmeldung am Broker von jedem Sensormodul dessen zuletzt gesendete Bootup-Record (`MQTT_TOPIC_TMPL_BOOTUP` = Gerätekonfiguration) und Sensordaten-Record (`MQTT_TOPIC_TMPL_ST_DATA` = aktuelle Station Umgebungsdaten) erhält. Der Client kennt somit unmittelbar nach seiner Anmeldung am Broker den aktuellen Zustand des Gesamtsystems.

Beim Start von *LoraPacketRecv* mit dem Kommandozeilenparameter *"-t"* werden zusätzlich zu den JSON-Records noch kompakte Telemetrie Messages mit dem Topic `MQTT_TOPIC_TELEMETRY` an den MQTT-Broker publiziert. Diese enthalten folgende Informationen als ASCII String:

- Empfangs-Timestamp
- Message-ID (fortlaufende Nummer über alle Pakete auf Empfängerseite)
- DevID (Node-ID) der Sensorbaugruppe
- Sequenz-Nummer (fortlaufende Nummer für alle gesendeten Pakete einer Sensorbaugruppe)
- RSSI Level des empfangenen Paketes

Die Telemetrie Message hat folgenden beispielhaften Aufbau:

```
Time=2023/03/11-15:52:08, MsgID=2, Dev=1, Seq=1, RSSI=-45
```

Um die Verbindung zum Broker aktiv aufrecht zu erhalten, nutzt *LoraPacketRecv* das Topic `MQTT_TOPIC_KEEPALVIE` zum Senden von Keep-Alive Nachrichten. Als Zeitbasis dient die Konstante `MQTT_KEEPALIVE_INTERVAL`. Für das Senden der Nachrichten ist die Funktion *MqttKeepAlive()* verantwortlich.

**Autostart für LoraPacketRecv**

Eine hohe Verfügbarkeit der *LoraPacketRecv* Gateway-Software ist eine elementare Voraussetzung für die erfolgreiche Weitergabe der von den Sensorbaugruppen per LoRa gesendeten Daten an einen zentralen MQTT-Broker. Daher soll die Gateway-Software beim Booten des RasperryPi automatisch mitgestartet werden. Sollte es zur Laufzeit zu einer unbeabsichtigten Beendigung der Software kommen, so ist diese ebenfalls sofort wieder neu zu starten ("respawn").

Für das Start-/Stop-Handling der *LoraPacketRecv* Gateway-Software sind die im Unterverzeichnis *"Autostart"* enthaltenen Skripte zuständig. Damit diese auf dem RaspberryPi ausgeführt werden können, benötigen sie entsprechende Rechte:

```
cd /home/pi/LoraPacketRecv/Autostart
chmod +x LoraPacketRecv
chmod +x Start_LoraPacketRecv.sh
chmod +x Stop_LoraPacketRecv.sh
```

- ***LoraPacketRecv:***
  Dieses Skript dient zum Starten und Stoppen der Gateway-Software auf Basis des Linux */etc/init.d/* Prozesses. Es sorgt unter anderem dafür, dass die Gateway-Software während des Bootvorgangs vom Linux Init-Prozess automatisch gestartet wird. Dazu bedient es sich wiederum der beiden Skripte *"Start_LoraPacketRecv.sh"* und *"Start_LoraPacketRecv.sh"*. Damit das Skript automatisch gerufen wird, muss es zunächst in das Verzeichnis *"/etc/init.d/"* kopiert und dort registriert werden:

  ```
  cd home/pi/LoraPacketRecv/Autostart
  chmod +x LoraPacketRecv
  sudo cp ./LoraPacketRecv /etc/init.d/
  cd /etc/init.d
  sudo update-rc.d LoraPacketRecv defaults
  ```

- ***Start_LoraPacketRecv.sh:***
  Dieses Skript startet die *LoraPacketRecv* Gateway-Software mit root-Rechten als Hintergrundprozess. Dazu wartet es zunächst, bis die Ethernet-Schnittstelle vollständig verfügbar und mit einer gültigen IP-Adresse konfiguriert ist. Dies ist Voraussetzung für den Aufbau der Verbindung zum MQTT-Broker. Die Broker-Adresse selbst wird aus der Datei *"mqtt_host"* ausgelesen, die entsprechend anzupassen ist. Da ein Hintergrundprozess keine direkten Ausgaben in einem Terminal tätigen kann, werden sämtliche Ausgaben in die über die Umgebungsvariable `LORA_PACKET_RECV_LOG` konfigurierte Logdatei geschrieben. Um die SD-Karte möglichst nicht mit permanenten Schreibzugriffen zu belasten, wird für die Ausgaben standardmäßig die Datei *"/run/LoraPacketRecv.log"* im RAM-Filesystem des RaspberryPi genutzt. Das Logging der JSON-Records in die über den Kommandozeilen-Parameter *"-l=<msg_file>"* spezifizierte Datei wird über die Umgebungsvariable `JSON_MSG_LOG` gesteuert. Auch hier wird mit *"/run/LoraPacketRecv.log"* standardmäßig ebenfalls das RAM-Filesystem verwendet. Die while-Schleife am Ende des Skriptes realisiert die respawn-Funktionalität, d.h. sie startet die Gateway-Software erneut, sollte diese unkontrolliert beendet wurden sein. Um eine kontrollierte Beendigung zu ermöglichen, schreibt das Skript seine eigene Prozess-ID in die mittels `PID_FILE` definierte Datei (*"/run/PID_LoraPacketRecv"*), die bei Bedarf von *"Stop_LoraPacketRecv.sh"* ausgewertet wird.

- ***Stop_LoraPacketRecv.sh:***
  Dieses Skript beendet die *LoraPacketRecv* Gateway-Software kontrolliert. Dazu liest es die vom Skript *"Start_LoraPacketRecv.sh"* in der mittels `PID_FILE` definierten Datei (*"/run/PID_LoraPacketRecv"*) hinterlegte Prozess-ID aus. Diese Prozess-ID wird dann als Parameter für den Aufruf von *"kill -9 <pid>"* verwendet. Damit werden sowohl die *LoraPacketRecv* Gateway-Software selbst als auch das Start-Skript mit seiner respawn-Schleife beendet.

**Verwendete Drittanbieter Komponenten**

1. **LoRa Radio**

   Für den SX1276 LoRa-Chipsatz des *SEEED LORA/GPS RASPBERRY PI HAT* wird die *"RadioHead"* Treiberbibliothek verwendet:
   https://github.com/idreamsi/RadioHead

2. **MQTT Client**

   Für die MQTT Client Implementierung wird die *"Paho MQTT Embedded/C"* Bibliothek verwendet:
   https://github.com/eclipse/paho.mqtt.embedded-c