HTW-Dresden
Allgemeine Informatik

# Uplink-Downlink Kommunikation zwischen Arduino Uno mit LoRaWAN und MQTT-Client

Projektseminar
Wintersemester 24/25

Name: Gavrilova

Vorname: Anna

Matr. Nr.: 53045

Tag der Einreichung: 13. Februar 2025

Professor: Prof. Dr.-Ing. Jörg Vogt

# Inhaltsverzeichnis

# 1 Einleitung

Dieses Projekt demonstriert die bidirektionale Kommunikation zwischen einem Arduino-Gerät mit LoRaWAN-Modul und einem MQTT-Client über The Things Network (TTN). Es dient als Beispiel für die Implementierung von Up- und Downlinks in einem LoRaWAN-Netzwerk und die Verarbeitung der Daten mit einem MQTT-Client.

## 1.1 Hardware

- **Arduino Uno**
- **LoRa BEE v1.1**
- **Antenne**
- **LoRaWAN-Gateway**

Eine detaillierte Beschreibung der Übertragungstechnik, Architektur und Hardware ist verfügbar unter: `https://github.com/HTWDD-RN/LoRaWAN-Wetterstation-SEN-15901/blob/main/docs/documentation.pdf`

# 2 Zielsetzung

Das Ziel dieses Projekts war die Implementierung einer bidirektionalen Kommunikation (Uplink und Downlink) zwischen einem Arduino Uno, ausgestattet mit einem LoRaWAN-Modul, und einem MQTT-Client. Die Kommunikation sollte über The Things Network (TTN) erfolgen.

# 3 Projektdurchführung

## 3.1 Einrichtung der Arduino-Plattform und LoRaWAN-Anbindung

- **Hardware-Setup:** Ein Arduino Uno wurde mit einem LoRaWAN-Modul verbunden, um die Kommunikation über das LoRaWAN-Protokoll zu ermöglichen.
- **Software-Einrichtung:** Die Arduino IDE wurde verwendet, um die erforderlichen Bibliotheken, insbesondere die LMIC (LoRaWAN in C) Bibliothek, zu installieren.
- **TTN-Geräteregistrierung:** Das Gerät (Arduino mit LoRaWAN-Modul) wurde auf der Webseite von The Things Network (`https://www.thethingsnetwork.org/`) registriert. Dabei wurden die Device EUI, Application EUI und der App Key generiert. Diese Schlüssel sind für die Authentifizierung des Geräts im Netzwerk unerlässlich.
- **Konfiguration:** Die `config.h`-Datei der LMIC-Bibliothek wurde angepasst, um die korrekten Frequenzen für die Region (Europa, 868 MHz) einzustellen. Die Konfigurationsdatei ist essenziell für den korrekten Betrieb.
- **Beispielcode:** Als Ausgangspunkt diente der Beispielcode der LMIC-Bibliothek, der für die Uplink- und Downlink-Ausgabe angepasst wurde.

## 3.2 Uplink-Kommunikation

- **Implementierung:** Zunächst wurde die Uplink-Kommunikation implementiert. Der Arduino sendete in regelmäßigen Intervallen (jede Minute) eine einfache "Hello World"-Nachricht an TTN.

- **Verifizierung:** Die erfolgreiche Übertragung der Nachrichten konnte in der TTN-Konsole (Application Data) überprüft werden.

## 3.3 Downlink-Kommunikation-Test

- **Grundlegende Tests:** Die Downlink-Kommunikation wurde getestet, indem Nachrichten im Hexadezimalformat über die TTN-Konsole an den Arduino gesendet wurden.

- **Code-Anpassung:** Der LMIC-Beispielcode wurde erweitert, um nicht nur den Empfang einer Downlink-Nachricht zu bestätigen, sondern auch den Inhalt der Nachricht im seriellen Monitor der Arduino IDE anzuzeigen. Dies diente der Verifizierung der korrekten Übertragung. Beispielcode (Ausschnitt):

Listing 1: Arduino Code für Downlink-Anzeige

```
void onEvent (ev_t ev) {
    // ... anderer Code ...
    case EV_TXCOMPLETE:
        Serial.println(F("EV_TXCOMPLETE (includes
            waiting for RX windows)"));
        if (LMIC.txrxFlags & TXRX_ACK)
          Serial.println(F("Received ack"));
        if (LMIC.dataLen) {
          downlinkCount++;
          if (downlinkCount >= MAX_DOWNLINKS) {
              Serial.println(F("Max downlinks reached,
                stopping."));
              exit(0);
          }
          Serial.print(F("Received "));
          Serial.print(LMIC.dataLen);
          Serial.println(F(" bytes of payload"));
          // Convert received bytes to a string
          char receivedMessage[LMIC.dataLen + 1];
            // +1 for null terminator
          for (int i = 0; i < LMIC.dataLen; i++) {
              receivedMessage[i]
                = (char)LMIC.frame[LMIC.dataBeg + i];
          }
          // Null-terminate the string
          receivedMessage[LMIC.dataLen] = '\0';


          // Print the received message as a string
          Serial.print(F("Message: "));
          Serial.println(receivedMessage);

          // Extract srNr from the decoded message
          char messageStr[LMIC.dataLen + 1];
          strcpy(messageStr, receivedMessage);
          char *srNrIndex = strstr(messageStr, "srNr: ");
          if (srNrIndex != NULL) {
              srNr = atoi(srNrIndex + 6);
              Serial.print(F("Extracted srNr: "));
              Serial.println(srNr);
          }

        }
          // Schedule next transmission
```

```
            os_setTimedCallback(&sendjob, os_getTime()+sec2osticks(TX_INTERVAL), do_send);
            Serial.println(F("Break in EV_Complete"));
            break;
    }
```

## 3.4 Entwicklung des MQTT-Clients

- **Programmiersprache:** Der MQTT-Client wurde in Java implementiert, um eine flexible und plattformübergreifende Lösung zu gewährleisten. Die `HiveMQ` Bibliothek wurde verwendet.
- **Kommunikationsprinzip:**
  1. Der MQTT-Client veröffentlicht eine Downlink-Nachricht auf dem TTN-Topic `.../down/push`.
  2. TTN leitet die Nachricht an das Topic `.../down/queued` weiter.
  3. TTN schickt die Nachricht an das Topic `.../down/sent`.
  4. Der Arduino empfängt die Nachricht und sendet eine Bestätigungsnachricht (ACK) auf dem Topic `.../up`.
  5. Der MQTT-Client abonniert das `.../up`-Topic und wartet auf die Bestätigung. Nach Empfang der Bestätigung sendet der Client die nächste Downlink-Nachricht.
  6. Ein Timeout-Mechanismus (5 Minuten) wurde implementiert, um auf fehlende Uplink-Nachrichten zu reagieren und die Downlink-Nachricht erneut zu senden.
- **Sequenznummern und RTT-Messung:** Um die Kommunikation zuverlässiger zu gestalten und Paketverluste zu erkennen, wurden die Nachrichten mit Sequenznummern versehen. Der Client speicherte die Sequenznummern und konnte so erkennen, ob eine empfangene Uplink-Nachricht zur gesendeten Downlink-Nachricht gehörte. Nach erfolgreicher Übertragung wurde die Round-Trip-Time (RTT) berechnet und in der Konsole ausgegeben.
- **Nachrichtenbegrenzung (TTN Fair Use Policy):** Um die Fair Use Policy von TTN einzuhalten, wurde die Anzahl der Nachrichten begrenzt:
  - **Arduino:** Ein globaler Zähler wurde implementiert, der sowohl Join-, Uplink- als auch Downlink-Nachrichten zählt. Das Arduino-Programm wurde so konfiguriert, dass es nach 10 Nachrichten stoppt.
  - **MQTT-Client:** Der Client wurde auf maximal 5 gesendete Downlink-Nachrichten begrenzt.

# 4 Probleme und Lösungen

## 4.1 Behobene Probleme

- **Downlink-Weiterleitungsproblem:**
  - **Problem:** Downlink-Nachrichten, die vom MQTT-Client auf das `.../down/queued`-Topic gesendet wurden, wurden von TTN auf `.../down/sent` nicht weitergeleitet.
  - **Ursache:** Eine Änderung in der Logik des Arduino-Codes, die die Uplink-Nachrichten nur noch nach Empfang einer Downlink-Nachricht sendete (anstatt regelmäßig jede Minute). TTN scheint Geräte ohne regelmäßige Uplink-Aktivität als inaktiv zu betrachten und leitet keine Downlink-Nachrichten weiter.
  - **Lösung:** Die regelmäßige Übertragung von Uplink-Nachrichten (jede Minute) wurde wiederhergestellt, wodurch das Problem behoben wurde.
  - **Weitere Verbesserungen:** Aufgrund von Zeitmangel konnte keine optimierte Lösung implementiert werden.
- **Konfigurationsproblem:**
  - **Problem:** Der LMIC-Beispielcode konnte anfangs nicht ausgeführt werden.
  - **Ursache:** Fehlende Anpassung der Konfigurationsdatei (`config.h`) an die spezifischen Parameter des LoRaWAN-Moduls und der Region.
  - **Lösung:** Die Konfigurationsdatei wurde korrekt angepasst.

## 4.2 Unbehobene Probleme

- **Verbindungsprobleme (Join-Requests):**
  - **Problem:** Während des gesamten Projekts traten immer wieder Verbindungsprobleme auf. Der Arduino konnte häufig keine Verbindung zu TTN herstellen, selbst nach erfolgreichem Senden von Join-Requests. Selbst geringfügige Codeänderungen konnten die Verbindung unterbrechen.
  - **Ursachenforschung:** Das Problem trat sowohl bei der Arbeit zu Hause als auch in der Nähe eines Gateways auf. Der Kontakt mit dem TTN-Techsupport legte nahe, dass die Probleme möglicherweise mit den verwendeten Geräten (LoRa BEE v1.1, Antennen) zusammenhängen.
  - **Auswirkungen:** Die instabile Verbindung behinderte die Entwicklung und das Testen erheblich.
  - **Keine Lösung:** Es konnte keine zuverlässige Lösung für dieses Problem gefunden werden.
- **Speicherbeschränkungen des Arduino:**
  - **Problem:** Der begrenzte Speicher (SRAM) des Arduino führte dazu, dass nach dem Empfang mehrerer Downlink-Nachrichten der Speicher voll war, was weitere Aktionen (Ausgabe, Senden von Nachrichten) verhinderte.

- **Keine vollständige Lösung:** Aufgrund von Zeitmangel konnte das Problem nicht vollständig behoben werden.
- **Mögliche Lösungsansätze:**
  - Ersetzen von `String`-Objekten durch `char[]`-Arrays.
  - Verwendung von `byte` anstelle von `int`, wo immer möglich.
  - Minimierung der Anzahl globaler Variablen.
  - Auslagern von Konstanten in den PROGMEM-Speicher (Flash-Speicher).

# 5 Erkenntnisse und Lernergebnisse

- **The Things Network (TTN):** Ein tiefgreifendes Verständnis von TTN (`https://www.thethingsnetwork.org/`) und seiner Fair Use Policy wurde erworben.
- **LoRaWAN-Kommunikation:** Die Funktionsweise der Kommunikation zwischen einem Endgerät (Arduino mit LoRaWAN-Modul) und einem Client (MQTT-Client) über LoRaWAN und TTN wurde verstanden.
- **Arduino-Programmierung:** Erfahrung in der Programmierung der Arduino- Plattform mit der LMIC-Bibliothek wurde gesammelt.
- **MQTT-Client-Entwicklung:** Ein eigener MQTT-Client wurde in Java entwickelt und implementiert.
- **Speicherbeschränkungen:** Die Herausforderungen der begrenzten Speicherressourcen von Mikrocontrollern wurden erkannt und erste Lösungsansätze wurden erarbeitet. Dies ist besonders relevant, da moderne Computer in der Regel über ausreichend Speicher verfügen, sodass dieses Problem im alltäglichen Programmieren oft nicht auftritt.
- **Fehlersuche:** Erfahrung mit dem systematischen finden von Fehlern und deren Dokumentierung.

# 6 Fazit und Ausblick

Das Projektziel, eine bidirektionale Kommunikation zwischen einem Arduino und einem MQTT-Client über TTN aufzubauen, wurde grundsätzlich erreicht. Es konnten sowohl Uplink- als auch Downlink-Nachrichten erfolgreich übertragen werden. Die Entwicklung des MQTT-Clients ermöglichte eine flexible Steuerung der Kommunikation.

Allerdings blieben einige wesentliche Probleme, insbesondere die instabile Verbindung und die Speicherbeschränkungen des Arduino, ungelöst. Diese Probleme beeinträchtigten die Zuverlässigkeit und die langfristige Nutzbarkeit des Systems.

Für zukünftige Arbeiten wären folgende Punkte relevant:

- **Ursachenforschung der Verbindungsprobleme:** Eine gründliche Analyse der Hardware (Antennen, Verbindungen, LoRaWAN-Modul) und der Software (LMIC-Bibliothek, Konfiguration) ist erforderlich, um die Ursache der Verbindungsprobleme zu identifizieren und zu beheben. Es könnte auch hilfreich sein, verschiedene Arduino-Boards und LoRaWAN-Module zu testen.

- **Optimierung des Speichermanagements:** Der Arduino-Code muss optimiert werden, um den Speicherverbrauch zu minimieren und eine stabile Funktion auch bei längeren Laufzeiten und häufigeren Downlink-Nachrichten zu gewährleisten.

Trotz der aufgetretenen Herausforderungen war das Projekt lehrreich und bot wertvolle Einblicke in die Entwicklung von IoT-Anwendungen mit LoRaWAN und MQTT. Die gewonnenen Erkenntnisse bilden eine gute Grundlage für zukünftige Projekte in diesem Bereich.