Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Fachbereich Informatik
Department of Computer Science

Masterthesis

im Master-Studiengang Computer Science

Sichere LPWAN-Übertragungen und IoT-Updates am Beispiel eines Smarten Briefkastens

von

Felix Barz

Betreuer:	Prof. Dr. Karl Jonas
Zweitbetreuer:	Prof. Markus Ullmann
Eingereicht am:	22. Juni 2019

Zusammenfassung

In dieser Arbeit wurde ein Smarter Briefkasten entwickelt, mit dem Schwerpunkt auf Sicherheit und Energieeffizienz. Es handelt sich dabei um eine Mikrocontroller-Anwendung auf Basis von LoRaWAN, einem LPWAN-Protokoll, welche in existierende Briefkästen eingebaut werden kann und eine Benachrichtigung an ein Mobilgerät sendet, wenn sich Post in diesem befindet. Es wurden dazu eine E2E-Verschlüsselung, sowie ein Verfahren zum Austausch von Schlüsselmaterial entwickelt, mit denen der Briefkasten Daten sicher an das Mobilgerät senden kann. Als Kern dieser Arbeit wurde außerdem ein sicherer OTA-Firmware-Update-Mechanismus für LoRaWAN entwickelt und getestet. Es konnte dabei gezeigt werden, dass es möglich ist, einen solchen Algorithmus praxisnah umzusetzen und das Firmware-Updates via LoRaWAN dementsprechend möglich sind. Das Ergebnis dieser Arbeit ist somit eine Anwendung, welche als Beispiel für sichere Datenübertragung und Firmware-Updates im LPWAN basierten IoT dient.

Erklärung

Felix Barz
Ringstraße 16
56204 Hillscheid

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Arbeit selbst angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.
Die Arbeit wurde bisher keiner Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

..... Hillscheid, 17.06.2019 .....

Unterschrift | Ort, Datum

Inhaltsverzeichnis

Abbildungsverzeichnis viii

Tabellenverzeichnis ix

Listings x

Abkürzungsverzeichnis xi

1 Einleitung 1
- 1.1 Der Smarte Briefkasten . . . 3
  - 1.1.1 Anwendungsfall . . . 3
  - 1.1.2 Untersuchte Themen . . . 3

2 Grundlagen 5
- 2.1 LoRaWAN . . . 5
  - 2.1.1 Was ist LoRa . . . 5
  - 2.1.2 Was ist LoRaWAN . . . 6
  - 2.1.3 Datenübertragungsmodi . . . 7
  - 2.1.4 Verschlüsselung in LoRaWAN . . . 9
  - 2.1.5 Netzwerkbeitritt . . . 10
  - 2.1.6 Bekannte Angriffe gegen LoRaWAN . . . 11
- 2.2 NB-IoT . . . 12
- 2.3 Vergleich von LoRaWAN und NB-IoT . . . 13
  - 2.3.1 Allgemeiner Vergleich der Technologien . . . 13
  - 2.3.2 Vergleich im Kontext des Smarten Briefkastens . . . 14
- 2.4 Definition Firmware und OTA-Updates . . . 14
- 2.5 LDPC-Codes . . . 15
  - 2.5.1 Reguläre LDPC-Codes . . . 15
  - 2.5.2 Abgewandelter Algorithmus . . . 16
- 2.6 Kryptografie . . . 20
- 2.7 Verwendete Hard- und Software . . . 21
  - 2.7.1 Die Arduino Plattform . . . 21
  - 2.7.2 Adafruit Feather M0 . . . 21
  - 2.7.3 Adafruit Huzzah32 - ESP32 Feather . . . 22
  - 2.7.4 Adafruit LoRa-Radio . . . 22
  - 2.7.5 Arduino Bibliotheken . . . 23
  - 2.7.6 Qt-Framework für Client-Anwendungen . . . 23
  - 2.7.7 LoRaWAN-Server-Architekturen . . . 24

3 LoRaWAN Ende-zu-Ende-Verschlüsselung 26
- 3.1 Stand der Forschung . . . 26
- 3.2 Methodik . . . 26
- 3.3 Anwendungsfall E2E – Angriffs-Szenarien . . . 27
- 3.4 Auswahl des E2E Algorithmus . . . 28
  - 3.4.1 Algorithmus für Austausch-Verschlüsselung . . . 28
  - 3.4.2 Wahl des IV für den Blockverschlüsselungsmodus . . . 30
  - 3.4.3 Vorschläge für E2E-Algorithmen . . . 31
  - 3.4.4 Angriffe gegen die gewählten Algorithmen . . . 31
  - 3.4.5 Untersuchung des Energieverbrauch der gewählten Algorithmen . . . 32
- 3.5 Implementierung . . . 32

3.6 Versuchsanordnung . . . 33
3.6.1 Versuchsaufbau . . . 33
3.6.2 Verwendete Geräte und Konfiguration . . . 34
3.6.3 Messung auf GND-Level . . . 34
3.6.4 Durchführung der Messungen . . . 34
3.7 Evaluation . . . 34
3.7.1 Auswertung der Messergebnisse . . . 35
3.7.2 Berechnung der Stromaufnahme . . . 35
3.8 Ergebnisse . . . 36
3.8.1 Qualität der Messergebnisse . . . 37
3.8.2 Auswertung des Stromverbrauchs . . . 37
3.8.3 Wahl eines Algorithmus . . . 37
3.8.4 Verbesserungsvorschläge . . . 38

4 Clientseitige Geräte-Aktivierung 39
4.1 Methodik . . . 39
4.2 Vorteile der clientseitigen Aktivierung . . . 39
4.2.1 Angriffe gegen Hersteller-Schlüssel . . . 39
4.2.2 Perfect Forward Secrecy . . . 40
4.2.3 Schlüssel-Verwaltung . . . 40
4.3 Nachteile der clientseitigen Aktivierung . . . 40
4.4 Protokoll . . . 40
4.4.1 Wahl des Sekundär-Kanals . . . 41
4.4.2 Bluetooth Low-Energy . . . 41
4.5 Implementierung . . . 42
4.6 Evaluation . . . 43
4.7 Ergebnisse . . . 43

5 Firmware-Updates via LoRaWAN 44
5.1 Stand der Forschung . . . 44
5.1.1 Herausforderungen an OTA-Updates via LoRaWAN . . . 44
5.1.2 Anwendbarkeit von klassischen OTA-Updates . . . 46
5.1.3 Protokolle für LoRaWAN . . . 46
5.2 Methodik . . . 46
5.2.1 Kriterien der Bewertung . . . 47
5.3 LoRaWAN Application-Layer Spezifikationen . . . 48
5.3.1 Application Layer Clock Synchronization . . . 48
5.3.2 Remote Multicast Setup . . . 48
5.3.3 Fragmented Data Block Transport . . . 49
5.4 Update-Protokoll . . . 50
5.5 Wahl der Basis-Bibliotheken . . . 53
5.6 Unicast Implementierung . . . 54
5.6.1 Fragmented Data Block Transport Protokoll . . . 54
5.6.2 Optimierte Bitmatrix . . . 54
5.6.3 Update-Protokoll . . . 55
5.7 Multicast Implementierung . . . 56
5.7.1 Clock Synchronization Protokoll . . . 56
5.7.2 Remote Multicast Setup Protokoll . . . 56
5.7.3 Anpassungen des Fragmented Data Block Transport Protokolls . . . 57
5.7.4 Portierung der LoRa Basic MAC Bibliothek . . . 57
5.8 Updater-Client-Anwendung . . . 57
5.8.1 Kommandozeile . . . 57
5.8.2 Server-Publish-Architektur . . . 60

5.9 Evaluation . . . 61
5.9.1 Funktionstest Clock Synchronization Protokoll . . . 61
5.9.2 Funktionstest Remote Multicast Setup Protokoll . . . 61
5.9.3 Funktionstest Fragmented Data Block Transport Protokoll . . . 65
5.9.4 Funktionstests Update-Protokoll . . . 67
5.9.5 Funktionstest Kommandozeilen-Updater . . . 68
5.9.6 Leistungstests Update-Protokoll . . . 68
5.10 Ergebnisse . . . 76
5.10.1 Bewertung auf Basis der Untersuchungs-Kriterien . . . 76
5.10.2 Vergleich zwischen theoretischen und praktischen Werten . . . 76
5.10.3 Abschließende Bewertung . . . 77
5.10.4 Vorschläge zur Verbesserung der Übertragungsdauer . . . 78

6 Methode zur Erkennung des Einwurfs eines Briefes in einen Briefkasten 80
6.1 Methodik . . . 80
6.1.1 Anforderungen an den Mechanismus . . . 80
6.1.2 Kriterien der praktischen Evaluation . . . 80
6.2 Mögliche Sensorik . . . 81
6.2.1 Infrarot Lichtschranke . . . 81
6.2.2 Infrarot Distanzmessung . . . 81
6.2.3 Ultraschall Distanzmessung . . . 82
6.2.4 Kraftmessung . . . 82
6.2.5 Klappen-Detektor . . . 83
6.3 Verwendete Hardware . . . 83
6.3.1 Infrarot Lichtschranke . . . 83
6.3.2 Infrarot Distanzmessung . . . 84
6.4 Evaluation der Hardware . . . 84
6.4.1 Infrarot Lichtschranke . . . 85
6.4.2 Infrarot Distanzmessung . . . 86
6.5 Ergebnisse . . . 87
6.5.1 Verwendung der Infrarot Lichtschranke . . . 87
6.5.2 Verwendung des Infrarot Distanzsensors . . . 88
6.5.3 Vergleich der Lösungen . . . 88

7 Entwicklung des Smarten Briefkastens 89
7.1 Methodik . . . 89
7.2 Anwendungs-Entwurf . . . 89
7.2.1 Anforderungen . . . 89
7.2.2 System-Architektur . . . 90
7.2.3 Definition des Postbox-Protokolls . . . 91
7.2.4 Design des Smarten Briefkastens . . . 93
7.2.5 Design des Caching-Servers . . . 94
7.2.6 Design der Client-Anwendung . . . 98
7.3 Auswahl der Hardware . . . 100
7.4 Implementierung . . . 100
7.4.1 Implementierung der Server-Anwendung . . . 100
7.4.2 Implementierung des Smarten Briefkastens . . . 101
7.4.3 Implementierung der Client-Anwendung . . . 101
7.5 Evaluation . . . 102
7.6 Ergebnisse . . . 104

8 Fazit 105

9 Literaturverzeichnis **107**

Anhang **112**

Abbildungsverzeichnis

1 Anzahl von aktiven IoT-Geräten weltweit [44] . . . 1
2 Vergleich von LPWAN mit anderen Technologien [46] . . . 2
3 Anstieg in bekannten Malware-Samples für IoT-Geräte [45] . . . 2
4 Chirp-Modulation einer LoRa-Nachricht [21] . . . 5
5 Topologie des LoRaWAN-Netzwerks [55] . . . 7
6 LoRaWAN Class-A Übertragung [38] . . . 8
7 LoRaWAN Class-B Beacon und Ping-Slots [38] . . . 8
8 LoRaWAN Class-C Übertragung [38] . . . 9
9 Ablauf LoRaWAN OTAA . . . 11
10 Vergleich LoRaWAN und NB-IoT [56] . . . 13
11 Die Arduino Entwicklungsumgebung . . . 22
12 Überblick Verschlüsselung in LoRaWAN mit E2E-Verschlüsselung . . . 26
13 Sequenz-Diagramm Austausch-Verschlüsselung, Variante 1 . . . 29
14 Sequenz-Diagramm Austausch-Verschlüsselung, Variante 2 . . . 29
15 Übersicht der drei getesteten Varianten . . . 31
16 Versuchsaufbau E2E Stromverbrauchs-Messung . . . 33
17 Stromverbrauch Messreihe ”Extra-Verschlüsselung” . . . 35
18 Aufgenommene elektrische Ladung aller Messreihen im Vergleich . . . 36
19 Sequenz-Diagramm Protokoll clientseitige Geräte-Aktivierung . . . 41
20 Übersicht über die Struktur von BLE und GATT . . . 42
21 Sequenz-Diagramm Update-Protokoll . . . 50
22 Veranschaulichung der optimierten Bitmatrix-Struktur . . . 55
23 LoRaWAN Beacon und Ping-Slots [38] . . . 62
24 Das Drotek DP0106 USB-GPS-Modul . . . 63
25 Versatz zwischen Versandt und Empfang von Ping-Slots . . . 64
26 Einsparung an Arbeitsspeicher, abhängig von m und x . . . 66
27 Update-Zeit in Abhängigkeit vom Spreading-Faktor SF . . . 71
28 Update-Zeit in Abhängigkeit von der Fragmentgröße l_f . . . 71
29 Update-Zeit in Abhängigkeit von der Update-Größe l_u . . . 73
30 Auswertung der Messergebnisse der Update-Zeit . . . 74
31 Auswertung der Messergebnisse der Transferzeit-Zeit pro Fragment . . . 74
32 Auswertung der Messergebnisse der verschiedenen Raten . . . 75
33 Versuchsaufbau Messung IR-Breakbeam Sensor . . . 84
34 Versuchsaufbau Messung IR-Proximity Sensor . . . 85
35 Versuchsaufbau Messung IR-Breakbeam Winkel . . . 86
36 Messung IR-Breakbeam Winkel . . . 86
37 Reichweite des TSSP4038 in Abhängigkeit von Winkel und Frequenz . . . 87
38 Architektur des Smarten Briefkastens . . . 90
39 Sequenz-Diagramm Postbox-Protokoll . . . 91
40 Abhängigkeiten der Komponenten des Smarten Briefkastens . . . 93
41 Programmablauf des Smarten Briefkastens . . . 95
42 Relationaler Datenbankentwurf des Caching-Servers . . . 96
43 Mockup Client: PostboxView, HistoryView, PostboxView (ausgeklappt) . . . 98
44 Mockup Client: AddHandshakeView, AddPresharedView, ConfigDialog . . . 99
45 Abhängigkeiten der Komponenten der Client-Anwendung . . . 99
46 Zustand des ESP32 im ”Deep Sleep” Modus [29] . . . 102
47 Aufbau der Test-Hardware für den Smarten Briefkasten . . . 103
48 Schaltplan des Smarten Briefkastens . . . 103

Tabellenverzeichnis

1 Eignung von LoRaWAN und NB-IoT für den Smarten Briefkasten 14
2 Vorteile der verschiedenen Verschlüsselungs-Varianten 28
3 IV für zusätzliche Verschlüsselung 30
4 PackageVersionAns Payload 51
5 DevIdentifierAns Payload 51
6 DevIdentifierAns DeviceInfo Feld 51
7 Update-Deskriptor 52
8 OtaAnnounce1 Payload 52
9 OtaAnnounce2 Payload 52
10 Messergebnisse: Delta Zeit-Synchronisation 61
11 BCNPayload Feld der LoRaWAN-Spezifikation [38] 65
12 BCNPayload Feld der Regionalen Parameter [36] 65
13 LoRaWAN-Kanäle für die EU [38][36] 70
14 Update-Zeit in Abhängigkeit vom Duty-Cycle 72
15 Vergleich Update-Zeit Duty-Cycle mit Class-B Ping-Slot-Periodizität 72
16 Auswertung der Infrarot-Lichtschranke 85
17 Auswertung der Infrarot-Distanz-Messung 87
18 PackageVersion Payload 92
19 PackageConfigReq Payload 92
20 PackageConfigReq Config Feld 92
21 PostStatus Payload 93
22 PostStatus Status Feld 93
23 Definition der Client-HTTP-REST-API 96

Listings

1 Beispiel einer signierten Metadaten-JSON-Datei 59
2 Beispiel einer RMC-Konfigurationsdatei 60
3 JSON-Definition des Setup-Objekts 97
4 JSON-Definition des Config-Objekts 97
5 JSON-Definition des Message-Objekts 97
6 JSON-Definition des LoRaParams-Objekts 98

Abkürzungsverzeichnis

ABP Activation by Personalisation

AES Advanced Encryption Standard

API Application programming interface

BLE Bluetooth Low Energy

CMAC Cipher-based Message Authentication Code

DER Distinguished Encoding Rules

DH Diffie-Hellman

DSA Digital Signature Algorithm

E2E Ende-zu-Ende

FSK Frequency Shift Keying

GATT Generic Attribute Profile

GCM Galois/Counter Mode

GMAC Galois Message Authentication Code

GPIO General purpouse input output

GPS Global Positioning System

HD-FDD Half-Duplex Frequency Division Duplexing

HMAC Hash-based Message Authentication Code

HTTP Hypertext Transfer Protocol

IoT Internet of Things

IV Initialisierungsvektor

LAN Local Area Network

LDPC Low-Density-Parity-Check

LiPo Lithium-Polymer

LPWAN Low Power WAN

LTE Long-Term Evolution

MAC Media Access Control

MIC Message Integrity Code

MITM Man In The Middle

MQTT Message Queuing Telemetry Transport

NFC Near Field Communication

OTA Over The Air

OTAA Over-the-Air Activation

PBKDF Pasword-based Key Deriviation Function

PFS Perfect Forward Secrecy

PKCS Public-Key Cryptography Standards

PPS Pulse per Second

PSK Phase-Shift Keying

QML Qt Meta-object Language

QPSK Quadrature PSK

REST Representational State Transfer

RSSI Received Signal Strength Indication

SHA Secure Hash Algorithm

SPI Serial Peripheral Interface

TTN The-Things-Network

UUID Universally Unique Identifier

W-LAN Wireless LAN

WAN Wide Area Network

1 Einleitung

Das Internet of Things ist ein immer größer werdender Teil der Informationswelt. Im Jahr 2018 waren es bereits 7 Milliarden aktive Geräte, bis 2025 wird ein Wachstum auf 21,5 Milliarden erwartet (Siehe Abbildung 1). Besonders im Home-Automation-Bereich finden sich immer wieder neue Anwendungsfälle für Smarte Geräte die sich miteinander vernetzen, um nach und nach mehr Funktionen zu erfüllen.

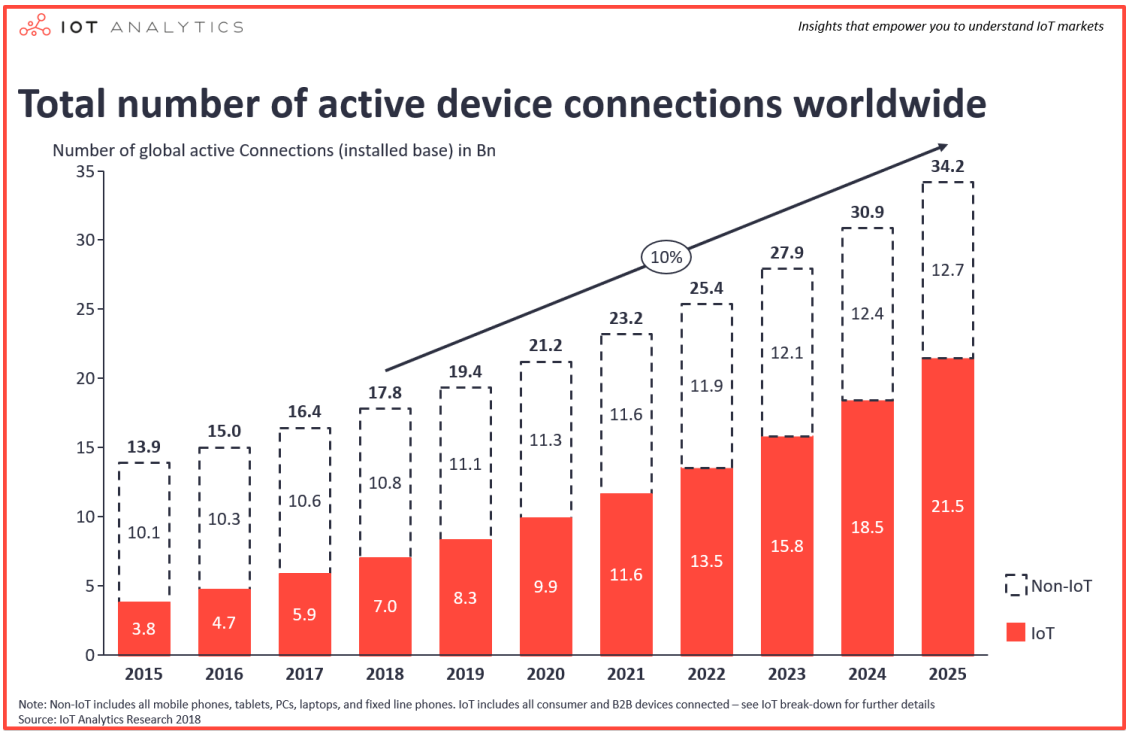

Abbildung 1: Anzahl von aktiven IoT-Geräten weltweit [44]

Die wichtigste Fähigkeit von IoT-Geräten ist dabei die der Kommunikation. Dazu gibt es unterschiedlichste Protokolle mit jeweils unterschiedlichen Anwendungsbereichen. Während im Heimbereich oftmals W-LAN, Bluetooth, Zigbee oder andere Funkstandards für kurze Distanzen zum Einsatz kommen, müssen Geräte an anderen Orten auf alternative Technologien zurückgreifen. Eine solche Gruppe von Technologien sind die LPWAN-Übertragungen. Diese haben den Vorteil einer deutlich höheren Reichweite und einer besseren Energieeffizienz als die meisten anderen Funkstandards, jedoch auf Kosten der Bandbreite. Abbildung 2 zeigt LPWAN im Vergleich zu einigen anderen Drahtlos-Kommunikationsprotokollen. LPWAN-Technologien werden zumeist dann eingesetzt, wenn Endgeräte an Orten installiert werden, die mit anderen Technologien nur schlecht zu erreichen sind, oder wenn Energieeffizienz von hoher Wichtigkeit ist. In dieser Arbeit wird LoRaWAN als LPWAN-Technologie eingesetzt.

Reichweite und Stromverbrauch sind jedoch nicht die einzigen wichtigen Punkte beim IoT. Ein Thema, das oftmals vernachlässigt wird, ist die Sicherheit von vernetzten Geräten. Allein zwischen 2017 und 2018 hat sich die Anzahl von bekannter Malware für IoT-Geräte fast vervierfacht (Siehe Abbildung 3). Doch Malware ist nicht die einzige Bedrohung. Viele IoT-Geräte versenden Daten unzureichend geschützt, wodurch Angriffe auf diese Daten und damit auch auf die IoT-Infrastruktur ermöglicht werden. Spionage, Manipulation von Daten und die komplette Übernahme von Systemen sind beispielhafte Angriffsszenarien.

Ein weiteres relevantes Thema bei IoT-Sicherheit sind Firmware-Updates; sie ermöglichen es Herstellern neue Funktionen auf Geräte zu bringen und im Fall von Sicherheitsvorfällen diese zu beheben, ohne dass der Nutzer aktiv werden muss. Dabei ist es über-

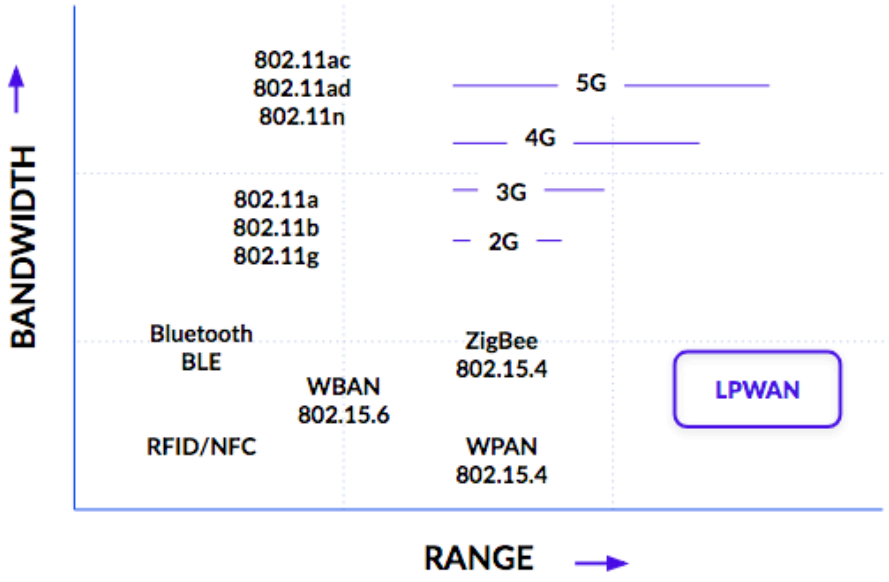

Abbildung 2: Vergleich von LPWAN mit anderen Technologien [46]

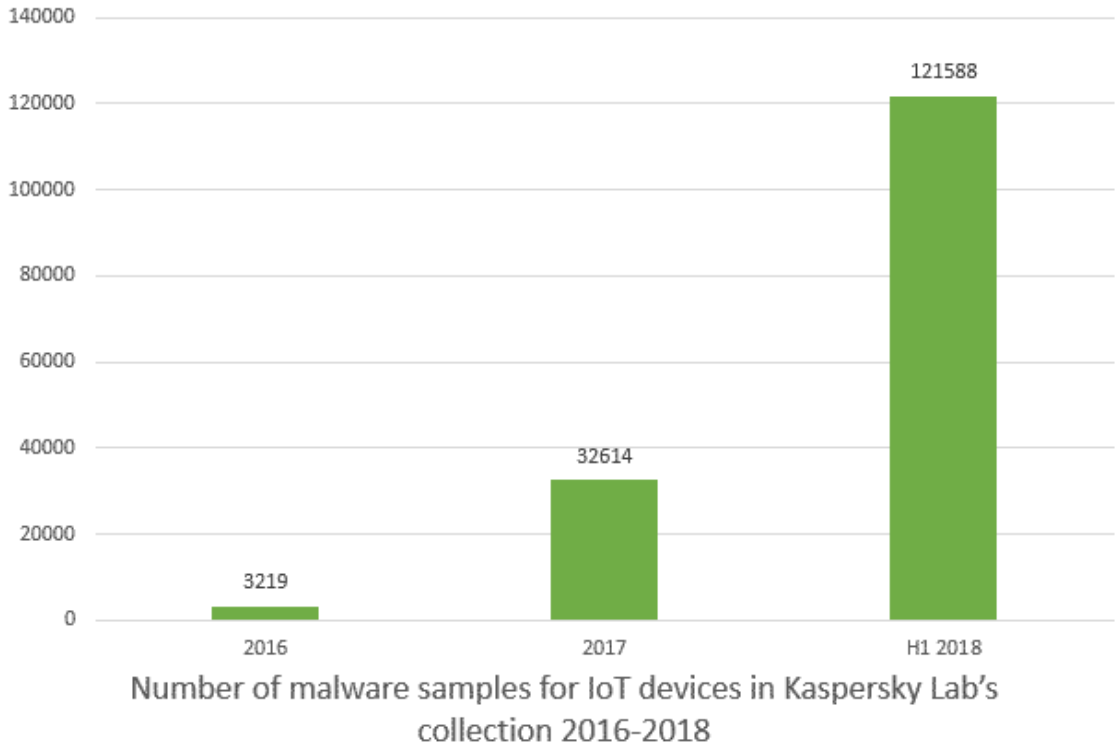

Abbildung 3: Anstieg in bekannten Malware-Samples für IoT-Geräte [45]

aus wichtig, dass die Updates ebenfalls auf sichere Weise durchgeführt werden, sodass Angreifer nicht in der Lage sind, einem Gerät gefälschte Firmware unterzuschieben.

Kombiniert man Firmware-Updates und Endgeräte, welche ein LPWAN-Protokoll zur Datenübertragung verwenden, so entsteht eine völlig neue Herausforderung. Während für IP-basierte Technologien, wie z. B. W-LAN, bereits einige Vorschläge für konkrete Protokolle existieren, sind Updates via LPWAN noch weitestgehend unerforscht. Der Grund dafür, dass die klassischen Protokolle nicht verwendet werden können, ist in den Einschränkungen der LPWAN-Technologien zu finden. LoRaWAN beispielsweise hat hohe Limitierungen bezüglich der Datenrate und Sendezeit und verfügt über kein standardisiertes Transport-Protokoll, womit Verluste bei der Datenübertragung ausgeglichen werden könnten.

1.1 Der Smarte Briefkasten

In dieser Arbeit wird ein konkreter Anwendungsfall betrachtet und an diesem verschiedene Themen der IoT-Sicherheit untersucht. Entwickelt wurde eine Smarter Briefkasten, welcher den Nutzer über dessen Smartphone benachrichtigt, wenn Post im Briefkasten eingegangen ist.

1.1.1 Anwendungsfall

Die Grundidee des Smarten Briefkastens ist es, dafür zu sorgen, dass der Nutzer nicht ständig seinen Briefkasten öffnen muss, um herauszufinden, ob sich Post darin befindet. Stattdessen soll er eine Benachrichtigung auf sein Mobilgerät erhalten, sobald sich Post im Briefkasten befindet. Dies hat den Vorteil, dass die Zeit des Nutzers nicht durch Briefkästen, die weit vom Nutzer entfernt sind, oder die nur selten Post enthalten, unnötig in Anspruch genommen wird.

Die Funktionen der Anwendung sind dabei bewusst klein gehalten, da der Schwerpunkt bei Sicherheit und Energie-Effizienz liegt. Zudem wurde die Anwendung als ”Erweiterung” entwickelt. Es soll möglich sein, diese ohne großen Aufwand seitens des Nutzers in beliebige Briefkästen einzubauen. Batteriebetrieb und ein kleiner Formfaktor sind somit ebenfalls Voraussetzungen für das Projekt.

1.1.2 Untersuchte Themen

Wie bereits zuvor erwähnt, liegt der Schwerpunkt der Arbeit auf der IoT-Sicherheit. Als Nebenpunkt wird außerdem die Energieeffizienz betrachtet. Aus diesen Schwerpunkten ergeben sich mehrere Unterthemen, die anhand des Smarten Briefkastens untersucht worden sind.

Der erste Punkt ist dabei die sichere, E2E-verschlüsselte Datenübertragung. Besonders bei Briefkästen in öffentlichen Bereichen ist es wichtig, dass ein Angreifer nicht erfahren kann, ob und wie viel Post sich in dem Briefkasten befindet. Damit wird verhindert, dass dieser auf einfache Weise herausfinden kann, ob sich ein Einbruch lohnen würde. Aus diesem Grund müssen Daten verschlüsselt werden. Ebenfalls wichtig ist die Absicherung gegen Manipulationen, um zu verhindern, dass dem Nutzer Falschinformationen über den Zustand des Briefkastens untergeschoben werden. Zusätzlich wird dabei untersucht, inwiefern die Aushandlung von kryptografischen Schlüsseln für diese Aufgaben auf sichere Weise durchgeführt werden kann ohne, dass der Hersteller des Gerätes oder Dritte diese in Erfahrung bringen können.

Der nächste Haupt-Untersuchungspunkt sind die sicheren Firmware-Updates via LoRaWAN. Aktuell gibt es keinen offiziellen Standard bezüglich der Übertragung von Firmware-

Updates via LoRaWAN. Insofern ist es eine der Hauptaufgaben dieser Arbeit, ein solches zu entwerfen und zu prüfen. Am Ende soll der Smarte Briefkasten in der Lage sein, ohne Eingriff des Nutzers Firmware-Updates via LoRaWAN durchzuführen. Diese Updates müssen außerdem kryptografisch abgesichert sein, um Manipulation zu verhindern.

Abschließend wird noch untersucht, wie Post, welche in einen Briefkasten eingeworfen wird, von dem Gerät erkannt werden kann. Dazu wurden verschiedene Technologien untersucht und eine geeignete ermittelt.

2 Grundlagen

In den folgenden Kapiteln werden einige Grundlagen erläutert, die als Basis für diese Arbeit dienen. Weitere Details zu den jeweiligen Themen können in den angegebenen Quellen gefunden werden.

2.1 LoRaWAN

LoRaWAN ist eine LPWAN-Lösung für IoT-Anwendungen, mit der geringe Datenmengen energieeffizient und über große Distanzen drahtlos übertragen werden können. Es besteht dabei zum einen aus LoRa-Radio, einem Protokoll für die physikalische Datenübertragung und zum anderen aus LoRaWAN selbst, einem MAC-Protokoll, welches auf LoRa aufbaut und ein standardisiertes Verfahren zum Transfer von Daten über LoRa bereitstellt [32]. LoRaWAN, als einer der Kernpunkte in dieser Arbeit, wurde für die Kommunikation mit den Endgeräten, also auch dem Smarten Briefkasten, verwendet.

2.1.1 Was ist LoRa

LoRa ist ein von Semtech entwickeltes Frequenz-Modulations-Verfahren, welches eine drahtlose Kommunikation zwischen zwei Kommunikationspartnern erlaubt [50]. Es ist somit ein physikalisches Protokoll (OSI-Layer 1), welches nur die Modulation der physikalischen Datenübertragung übernimmt. LoRa nutzt Frequenz-modulierte Chirps, um Symbole zu codieren.

Die verwendete Chirp-Modulation nutzt sogenannte ”Chirps”, um Symbole zu übertragen. Dabei wird über einen definierten Zeitraum die Frequenz kontinuierlich über die Bandbreite hinweg verändert. Die übertragenen Symbole werden über den Beginn des Chirps definiert. Abbildung 4 zeigt, wie eine solche Nachricht aussieht.

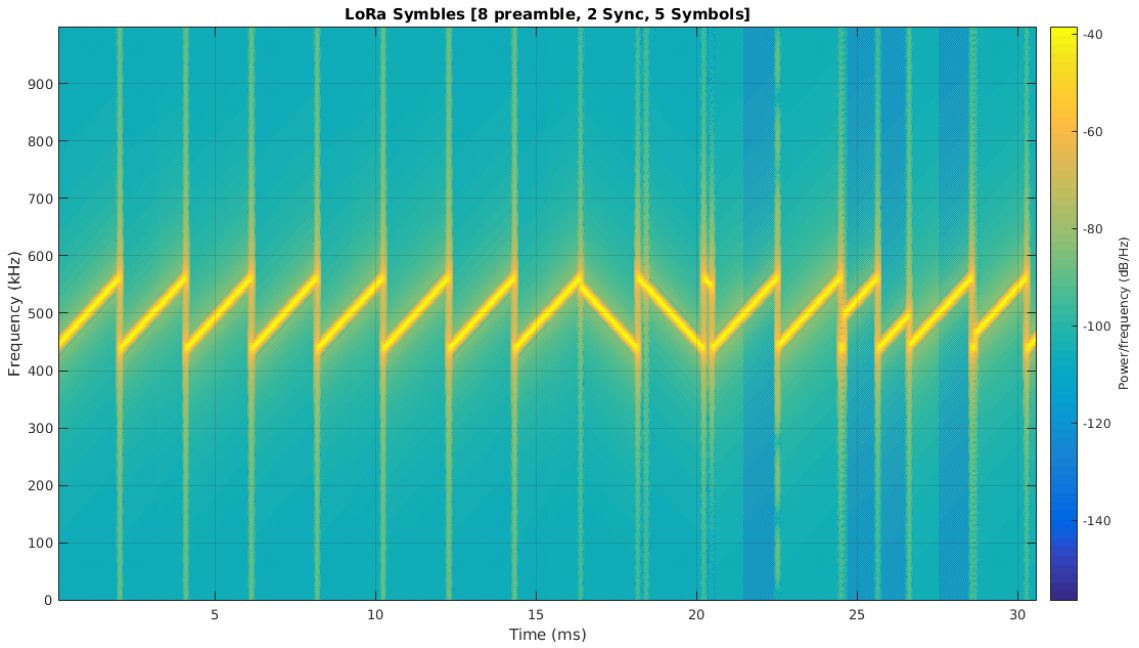

Abbildung 4: Chirp-Modulation einer LoRa-Nachricht [21]

Die primären Vorteile, die diese Modulation im Vergleich zu FSK oder PSK bietet, sind dabei die große Reichweite und die Robustheit gegen Rauschen. Beide werden von dem verwendeten Spreading-Faktor und der Bandbreite bestimmt [52]. Der Spreading-Faktor legt dabei fest, wie lange ein einzelner Chirp dauert, also wie breit dieser ”ge-

spreizt" wird. Ein höherer Faktor bedeutet breitere Symbole, was zwar für größere Übertragungsreichweiten sorgt, aber auch zu langsamerer Datenübertragung führt. In LoRa sind Spreading-Faktoren von 7 bis 12 definiert, womit Übertragungsgeschwindigkeiten von maximal 37.5 kBit/s bis minimal 300 Bit/s erreicht werden können [31]. Die Bandbreite ist auf 125 kHz, 250 kHz oder 500 kHz festgelegt und hat ebenfalls Einfluss auf die Reichweite und Geschwindigkeit des Signals. Die konkrete Wahl dieser Parameter wird durch LoRaWAN festgelegt.

Die Frequenzen, die LoRa verwendet, hängen von der Region ab. In Europa kann auf 868 MHz oder auf 433 MHz gesendet werden [36]. Dabei ist wichtig zu erwähnen, dass es sich bei diesen Frequenzen um lizenzfreie Spektren handelt, es muss also keine Lizenzgebühr für deren Verwendung gezahlt werden. Im Ausgleich dafür gelten zeitliche Sende-Beschränkungen, die alle Geräte einhalten müssen. Diese liegen zwischen 0,1% und 10%, abhängig von der verwendeten Frequenz [12].

2.1.2 Was ist LoRaWAN

LoRaWAN ist ein MAC-Protokoll (OSI-Layer 2), welches auf LoRa aufbaut (aber auch mit FSK verwendet werden kann), und außerdem einige Elemente eines Netzwerk-Protokolls (OSI-Layer 3) enthält [51]. Es definiert ein Nachrichten-Format, sowie MAC-Kommandos zum Steuern der Übertragung. Auch die Parameter für die zugrundeliegende LoRa-Übertragung werden von LoRaWAN festgelegt.

Die Spezifikation ist dabei in zwei Teile aufgeteilt. Der erste Teil ist die eigentliche Spezifikation, die die Nachrichtenformate, die MAC-Kommandos und den Ablauf festlegt [38]. Als Erweiterung dazu existieren die regionalen Parameter, welche konkrete Einstellungen für LoRa, sowie einige Anpassungen bzw. Ergänzungen zum LoRaWAN Protokoll definieren, abhängig von der jeweiligen Region [36].

Ein LoRaWAN Netzwerk besteht dabei aus mehreren Gruppen von Teilnehmern und ist in einer Stern-von-Sternen-Topologie organisiert, wie in Abbildung 5 zu sehen ist [32]. In der Mitte steht dabei der Netzwerkserver, welcher die serverseitige Verwaltung des LoRaWAN-Netzwerks übernimmt und eine API für Client-Anwendungen bietet, um LoRaWAN-Anwendungen zu verwalten und Nachrichten sowohl zu versenden als auch zu empfangen. Dieser Server kommuniziert über eine IP-Verbindung mit mehreren Gateways. Deren primäre Aufgabe ist es, die vom Netzwerk-Server empfangenen LoRaWAN-Pakete an die Endgeräte über LoRa weiterzuleiten und umgekehrt. Sie dienen entsprechend als Schnittstelle für den Wechsel des physikalischen Mediums. Am Ende stehen dann die Endgeräte, die mit einem oder mehreren Gateways kommunizieren, um ihre Daten zu übertragen. Dabei wird das LoRaWAN-Protokoll nur zwischen dem Gateway und den Endgeräten verwendet. Für die restlichen Pfade ist kein Standard definiert und das Format somit abhängig von den konkret verwendeten Anwendungen.

In diesem Kontext übernimmt LoRaWAN einige Aufgaben, die im Folgenden weiter erläutert werden. Dazu gehören die verschiedenen Kommunikations-Klassen, mit denen auf unterschiedliche Weisen Daten übertragen werden können, die beiden Möglichkeiten, Geräte einer LoRaWAN-Anwendung hinzuzufügen, die Verschlüsselung und Integritätsprüfung von übertragenen Daten, sowie die verschiedenen MAC-Kommandos zum Steuern der Verbindung [38]. Letztere werden dabei nicht weiter erläutert, da sie sehr spezifisch und für diese Arbeit nicht von Relevanz sind.

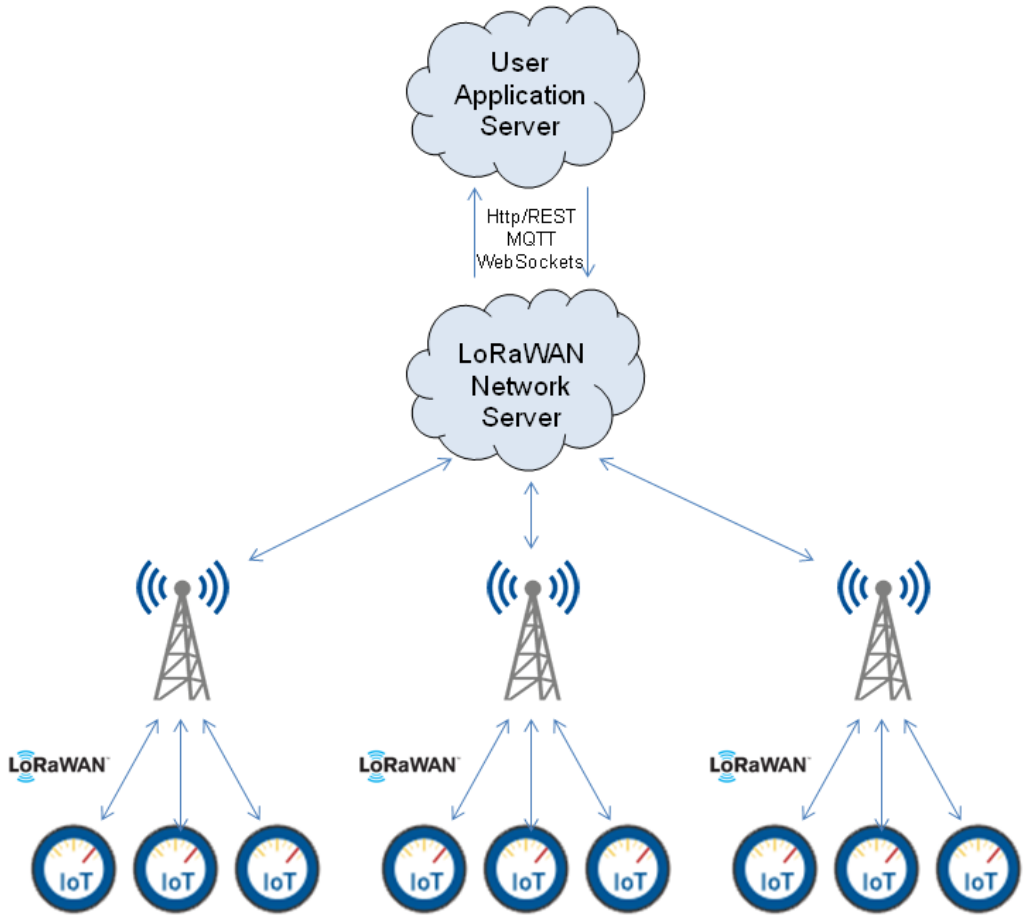

Abbildung 5: Topologie des LoRaWAN-Netzwerks [55]

2.1.3 Datenübertragungsmodi

LoRaWAN unterstützt drei verschiedene Modi zur Datenübertragung [38]. Jeder dieser Modi hat konkrete Anwendungsfälle, sowie Vor- und Nachteile, die im Folgenden aufgelistet werden.

Class-A

Der Class-A Modus ist der primäre Übertragungsmodus, der von allen Endgeräten und Gateways unterstützt werden muss. Er erlaubt bidirektionale Kommunikation zwischen Endgerät und Gateway nach dem ALOHA-Prinzip. Im Falle von LoRaWAN bedeutet das, dass ein Endgerät zu jedem Zeitpunkt Daten verschicken kann, aber nur für zwei kurze Intervalle nach dem Versand eines Datenpakets auch Daten empfangen kann. Dies wird in Abbildung 6 dargestellt.

Der Vorteil dieses Modus ist, dass ein Endgerät nur während dem Versand von Daten und kurz nach diesem zum Empfangen einer Antwort den LoRa-Transceiver einschalten muss. Somit kann dieser die meiste Zeit deaktiviert bleiben, was Energie spart. Der Nachteil ist jedoch, dass das Endgerät entsprechend zu allen anderen Zeitpunkten keine Daten empfangen kann. Außerdem kann für jedes gesendete Datenpaket nur eines empfangen werden.

Class-A-Betrieb gibt somit am meisten Sinn, wenn primär Uplink-Nachrichten versendet werden, und nur selten Downlink-Nachrichten. Da LoRaWAN vorzugsweise in Sensoren und ähnlichen leistungsschwachen Endgeräten verwendet wird, die im Normalfall nur Sta-

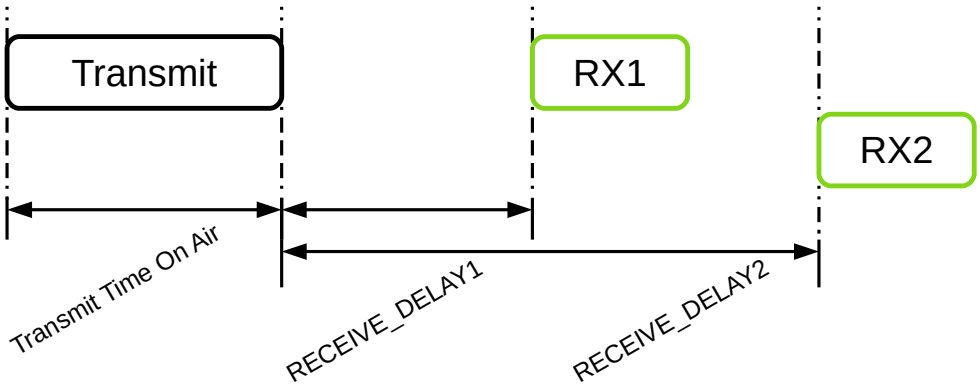

Abbildung 6: LoRaWAN Class-A Übertragung [38]

tusinformationen für eine Endanwendung bereitstellen, ist dies für die meisten Endgeräte der bevorzugte Datenübertragungsmodus.

Class-B

Class-B ist eine Erweiterung, die nicht von Endgeräten unterstützt werden muss. Durch diesen Modus können Endgeräte zusätzlich zu Class-A in regelmäßigen Zeitabständen Daten vom Gateway empfangen, ohne vorher selbst welche versenden zu müssen. Dazu wird vom Gateway alle 128 s ein sogenanntes Beacon gesendet, welches einige Status-Informationen über das Gateway enthält. Endgeräte, die dieses empfangen, können dann aus dem Beacon und der Ping-Slot-Periodizität berechnen, zu welchen Zeiten Daten empfangen werden können. Somit können diese zum richtigen Zeitpunkt für kurze Momente den LoRa-Transceiver einschalten, um eventuell vorhandene Daten zu empfangen. Der Ablauf ist in Abbildung 7 dargestellt.

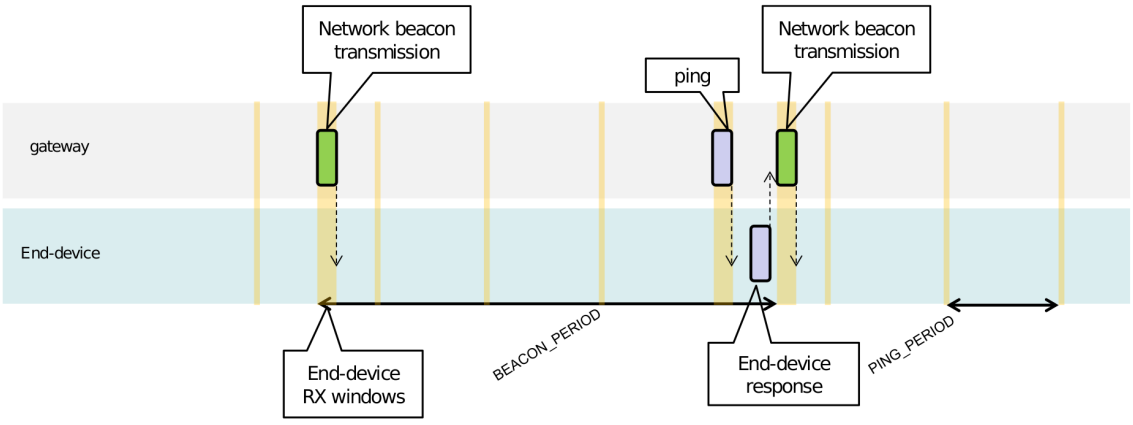

Abbildung 7: LoRaWAN Class-B Beacon und Ping-Slots [38]

Der primäre Vorteil, den der Class-B Modus bietet, ist, dass nun auch Nachrichten vom Endgerät empfangen werden können, ohne dass vorher selbst Daten versendet werden müssen. Class-B bietet dabei eine gute Balance zwischen Erreichbarkeit und Energieverbrauch, da durch die Anpassung der Ping-Slot-Periodizität gesteuert werden kann, wie oft empfangen werden soll. Somit wird zwar mehr Energie als für reinen Class-A Betrieb verbraucht, es bleiben aber immer noch lange Zeiträume, in denen der LoRa-Transceiver ausgeschaltet werden kann. Ein weiterer Vorteil von Class-B ist die Möglichkeit, Daten im

Multicast an mehrere Endgeräte gleichzeitig übertragen zu können, solange Adresse und Schlüssel gleich sind. Es können also Multicast-Gruppen erstellt werden.

Die Verwendung von Class-B ergibt dann Sinn, wenn ein Endgerät öfter Daten empfangen muss, ohne selbst Daten zu senden, das Gerät allerdings trotzdem noch energieeffizient arbeiten muss. Ein typischer Anwendungsfall wären Endgeräte, die regelmäßig angesteuert werden, ohne dabei zeitkritisch zu sein.

Class-C

Der letzte, ebenfalls optionale, Übertragungsmodus ist Class-C. In diesem schaltet das Endgerät permanent auf Empfang um jederzeit Daten vom Gateway empfangen zu können. Dabei bleibt die Möglichkeit zur Übertragung mit Class-A weiterhin erhalten, wie in Abbildung 8 zu sehen.

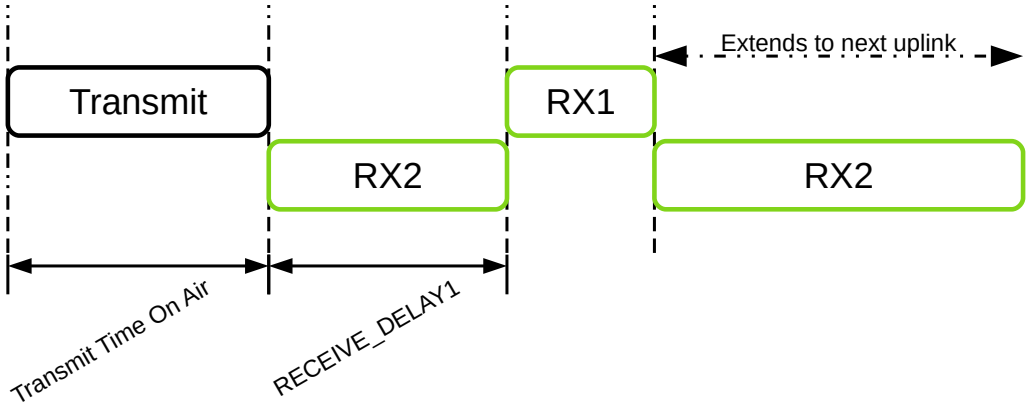

Abbildung 8: LoRaWAN Class-C Übertragung [38]

Der Vorteil von Class-C ist, dass zu jeder Zeit Daten empfangen werden können. Der Preis dafür ist jedoch ein hoher Energieverbrauch, da das Endgerät permanent den LoRa-Transceiver aktiv halten muss. Auch hier ist eine Multicast-Übertragung möglich.

Class-C sollte nur dann verwendet werden, wenn große Mengen an Daten über einen kurzen Zeitraum hinweg übertragen werden müssen, oder wenn zeitkritische Übertragungen stattfinden. Dabei sollten solche Endgeräte über eine permanente Stromversorgung verfügen, da dieser Modus für den Batterie-Betrieb zu viel Energie verbraucht ist.

2.1.4 Verschlüsselung in LoRaWAN

In LoRaWAN sind alle Nachrichten zwischen dem Endgerät und dem Netzwerkserver verschlüsselt. Verwendet wird dazu AES-128 im CTR-Modus, mit dem `AppSKey` als Schlüssel. Dabei wird ein 16 Bit IV aus Daten, welche dem Endgerät und dem Server bekannt sind, berechnet und dann mit AES verschlüsselt. Der IV wird aus der Geräte-Adresse, der Richtung (Uplink oder Downlink) und dem zugehörigen Frame-Counter generiert. Mit jedem Zyklus wird dieser dabei modifiziert, um somit einen Schlüsselstrom zu erzeugen. Dieser wird mit dem Klartext xoriert, um die Chiffre zu erstellen. Der zugehörige geheime Schlüssel wird als Teil des gewählten Beitritt-Verfahrens entweder vorher generiert oder beim Verbinden mit dem Gateway ausgehandelt.

Zusätzlich zu dieser Verschlüsselung werden die Daten noch mit einem weiteren Schlüssel, dem `NetSKey`, für die LoRa-Übertragung gegen Manipulation gesichert. Dazu wird ein 4-Byte MIC mithilfe von AES berechnet, der vom Empfänger verifiziert wird. Es wird dabei

ein leicht abgewandelter IV verwendet, der zusätzlich die Länge der Nachricht miteinbezieht.

Diese Verschlüsselung wird dabei auf alle via LoRaWAN übertragenen Nachrichten angewendet. Die einzigen Ausnahmen sind die `JOIN-Request` Nachricht, welche komplett unverschlüsselt versendet wird, und die `JOIN-Accept` Nachricht, welche mit dem `AppKey` verschlüsselt wird. Der MIC beider Nachrichten wird ebenfalls aus diesem Schlüssel berechnet.

2.1.5 Netzwerkbeitritt

In LoRaWAN werden Geräte über eine 32 Bit Adresse identifiziert. Die Geräte wiederum werden in einer logischen LoRaWAN-Anwendung verwaltet. Wenn ein Gerät Daten an ein Gateway übertragen möchte, dann muss es seine Adresse sowie einige weitere Informationen kennen, um sich verbinden zu können. Um diese Informationen dem Endgerät mitzuteilen, gibt es zwei Varianten, die im Folgenden erläutert werden.

Activation by Personalisation

Die simplere Variante der beiden ist Activation by Personalisation (ABP). Bei dieser Variante werden alle benötigten Informationen, also die Adresse, die kryptografischen Schlüssel und die Parameter für LoRa, vom Hersteller fest im Gerät einprogrammiert.

Während diese Variante einfach ist, hat sie den großen Nachteil, dass diese Parameter nicht mehr geändert werden können. Das Gerät kann somit nicht das Netzwerk wechseln und keine Schlüssel austauschen.

Over-the-Air Activation

Die etwas aufwendigere, dafür aber sicherere und somit empfohlene Variante ist die Over-the-Air Activation (OTAA). Bei dieser verfügt jedes Gerät über eine eindeutige `DevEuI`, einer Identifikationsnummer nach dem IEEE EUI64 Format. Diese dient entsprechend der Identifikation des Geräts. Des Weiteren existiert eine optionale `AppEuI`, welche der Zuordnung des Gerätes zu einer LoRaWAN-Anwendung dient, und ebenfalls dem IEEE EUI6 Format entspricht. Als letzter Parameter wird der `AppKey` benötigt, ein 128 Bit AES-Schlüssel, mit dem der Austausch abgesichert wird.

Möchte sich das Gerät mit dem Netzwerk verbinden, so wird eine `JOIN-Request` Nachricht an das Gateway gesendet, welche die `DevEuI` und `AppEuI`, zusammen mit einer Nonce, enthält. Es handelt sich dabei um eine Nachricht in einem speziellen Datenformat, welches sich von den restlichen LoRaWAN-Nachrichten unterscheidet. Werden die Daten vom Netzwerk-Server akzeptiert, so sendet das Gateway eine `JOIN-Accept` Nachricht zurück, welche eine Adresse, eine weitere Nonce und Parameter für LoRa-Kanäle enthält. Diese Antwort wird dabei mit dem `AppKey` verschlüsselt. Beide Nachrichten verwenden diesen Schlüssel außerdem, um einen MIC für die Authentifizierung der Nachrichten zu generieren. Beide Seiten können dann aus den bekannten Informationen die kryptografischen Schlüssel berechnen. Der komplette Ablauf wird in Abbildung 9 dargestellt. Ab diesem Zeitpunkt verfügt das Endgerät über alle Informationen, um gewöhnliche LoRaWAN-Nachrichten zu senden und zu empfangen.

Der größte Vorteil dieser Variante ist die dynamische Zuweisung von Adressen und kryptografischen Schlüsseln. Dies verbessert die Sicherheit und macht es dem Gerät möglich, das Netzwerk bei Bedarf zu wechseln. Außerdem können so vom Gateway Übertragungskanäle nach Bedarf definiert werden.

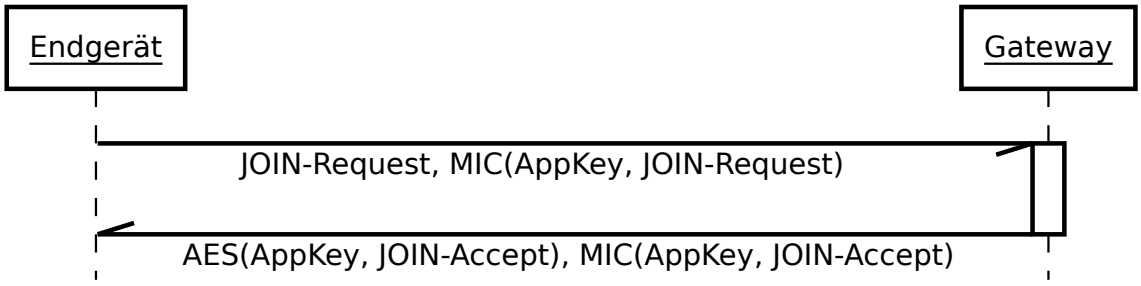

Abbildung 9: Ablauf LoRaWAN OTAA

2.1.6 Bekannte Angriffe gegen LoRaWAN

In diesem Kapitel werden einige bekannte Angriffe gegen LoRaWAN beschrieben, sowie deren Auswirkungen und mögliche Gegenmaßnahmen. Diese gelten, falls vorhanden, als Voraussetzung für diese Arbeit, um vermeidbare Sicherheitsrisiken zu umgehen.

Replay-Angriff gegen JOIN-Nachrichten

Wenn ein Gerät dem LoRaWAN Netz via OTAA beitreten möchte, muss als Erstes eine `JOIN-Request` Nachricht gesendet werden. Diese ist unverschlüsselt, was ein Angreifer ausnutzen kann, um einen Replay-Angriff gegen das Gerät durchzuführen mit dem Ziel, dieses aus dem Netzwerk auszuschließen [54]. Dazu muss dieser `JOIN-Request` Nachrichten mitschneiden und diese dann wiederholen, kurz bevor das Endgerät dem Netzwerk beitreten möchte. In der LoRaWAN-Spezifikation ist jedoch vorgesehen, dass der Netzwerkserver eine Liste von bereits verwendeten Nonces führt, um solche Angriffe zu verhindern [38]. Dieser Angriff funktioniert also nur dann, wenn das Gerät so oft dem Netzwerk beitritt, dass ältere Nonces nicht mehr in der Liste enthalten sind.

Eine Zweite Variante des Angriffs mit dem gleichen Ziel geht dabei leicht anders vor. Anstatt Nachrichten mitzuschneiden wird stattdessen der Zufallsgenerator des Endgeräts, der typischerweise den RSSI-Wert nutzt, angegriffen [59]. Durch gezieltes Jamming kann das Gerät dazu gebracht werden, mit hoher Wahrscheinlichkeit gleiche Zufallszahlen zu generieren. Damit schließt sich das Gerät selbst aus dem Netzwerk aus, da sich wiederholende Nonces vom Netzwerkserver blockiert werden.

Replay-Angriff gegen Daten-Nachrichten

Eine weitere Variante des Replay-Angriffs kann gegen Daten-Nachrichten durchgeführt werden. Dabei kann der Angreifer Nachrichten mitschneiden und später erneut senden, ohne diese entschlüsseln zu müssen [7]. Dieser Angriff ist jedoch nur dann möglich, wenn die Validierung von Frame-Countern deaktiviert wurde, da diese dafür sorgen, dass keine zwei Nachrichten die gleiche verschlüsselte Nachricht ergeben. Da LoRaWAN 32-Bit Counter benutzt, ist der Überlauf dieser ebenfalls keine realistische Gefahr. Bei einem Gerät, dass jede Sekunde eine Nachricht senden würde, würde es 136 Jahre dauern, bis dieser Überlauf stattfindet. Solange also Frame-Counter validiert werden, ist ein Replay-Angriff gegen LoRaWAN-Datenpakete nicht möglich.

Bit-Flipping-Angriff

Ein anderer Angriff, der gegen LoRaWAN Datennachrichten eingesetzt werden kann, ist der Bitflipping-Angriff. Bei diesem Angriff manipuliert ein Angreifer einzelne Bits des Cipher-Textes, mit dem Ziel, den Klartext sinnvoll zu manipulieren [6]. Dieser Angriff ist

möglich, da LoRaWAN den CTR-Modus zur Verschlüsselung benutzt. Bei diesem wird ein Schlüsselstrom mit dem Klartext xoriert, sodass Änderungen am Cipher-Text direkte Auswirkungen auf den Quelltext an der gleichen Stelle haben. Der Angriff ist außerdem effektiv gegen LoRaWAN, da oftmals nur reine Sensor-Daten versendet werden, bei denen die Manipulation eines Wertes zwar für falsche, aber immer noch syntaktisch und semantisch gültige Daten sorgt.

Als Gegenmaßnahme verwendet LoRaWAN den MIC. Dieser wird jedoch zum einen nur zwischen Gateway und Endgerät eingesetzt, was Angriffe an anderer Stelle immer noch möglich macht und kann zum anderen vom Angreifer gebrochen werden [30].

Gefahren von Multicast-Nachrichten

Eine weitere Gefahrenquelle bei LoRaWAN ist die falsche Verwendung von Multicast-Nachrichten. Die Gefahr, die von diesen ausgeht, ist, dass jedes Endgerät, das Teil der Gruppe ist, selbst Nachrichten an andere Geräte dieser versenden kann [48]. Aus diesem Grund ist es wichtig, dass keine sicherheitsrelevanten Daten via Multicast verschickt werden. Dies schließt die MAC-Kommandos des LoRaWAN-Protokolls mit ein.

Angriffe gegen Beacons

Die letzte Art von Angriffen, die hier betrachtet werden sollen, sind solche gegen LoRaWAN-Gateways. Diese senden regelmäßige Beacons, die unter anderem die exakten GPS-Koordinaten des Gateway enthalten. Angreifer können dies ausnutzen, um die Gateways physisch anzugreifen [28]. Des Weiteren könnte ein Angreifer gefälschte Beacons mit ungültigen Zeitstempeln senden und somit die Class-B Kommunikation des Endgeräts blockieren. Auf diese Angriffe wird jedoch in der Arbeit nicht weiter eingegangen, sie wurden hier nur der Vollständigkeit halber erwähnt.

2.2 NB-IoT

Das Narrowband-IoT LPWAN-Protokoll, kurz NB-IoT, ist eine Erweiterung des LTE-Standards, mit dem Ziel, ein Protokoll für das IoT anzubieten, welches Effizienz und Reichweite in den Vordergrund setzt. Dazu gibt es einige Änderungen, im Vergleich zum herkömmlichen LTE-Standard, welche im Folgenden aufgelistet werden [16].

1. Mit LTE Cat-NB1 kann genau ein LTE-PRB (180 kHz) verwendet werden, um Daten zu versenden.
2. Durch die Verwendung von HD-FDD wird nur eine Antenne benötigt.
3. Es wird nur QPSK als Modulationsstandard vorausgesetzt, mit vereinfachter Kanalcodierung und ohne regelbare Sendeleistung.
4. Nachrichten können akkumuliert werden, bis ein gutes Signal verfügbar ist.
5. Ein Power-Saving-Mode erlaubt Geräten für bis zu 310 h in einen Tiefschlaf-Modus zu wechseln, ohne sich danach neu im Netz anmelden zu müssen.
6. Unterstützt drei unterschiedliche Betriebsarten: In-Band, Guard Band und Stand-Alone.

Neben diesen Anpassungen profitiert NB-IoT auch von den Vorteilen, die das zugrundeliegende LTE-Protokoll bietet. Es wird ein lizenziertes Frequenzspektrum verwendet, was bedeutet, dass keine Beschränkungen in der Sendezeit vorliegen. Außerdem sind alle Datenübertragungen verschlüsselt. Geräte benötigen allerdings eine eSIM, welche das Gerät identifiziert und für die Generierung von kryptografischen Schlüssel verwendet wird.

2.3 Vergleich von LoRaWAN und NB-IoT

Als LPWAN-Technologie für die Realisierung des Smarten Briefkastens wurde LoRaWAN ausgewählt. In diesem Kapitel wird untersucht, wie sich LoRaWAN mit anderen LPWAN-Lösungen vergleichen lässt und welche Vor- und Nachteile diese bieten, wozu NB-IoT ausgewählt wurde. Im Anschluss an diesen Vergleich wurde bewertet, ob die Auswahl von LoRaWAN korrekt war, oder ob NB-IoT die bessere Technologie für den konkreten Anwendungsfall des Smarten Briefkastens gewesen wäre.

2.3.1 Allgemeiner Vergleich der Technologien

Im Folgenden werden die grundlegenden Eigenschaften der beiden Protokolle einander gegenübergestellt, um die Vor- und Nachteile herauszuarbeiten. Eine Übersicht dieser Vergleiche wird in Abbildung 10 dargestellt.

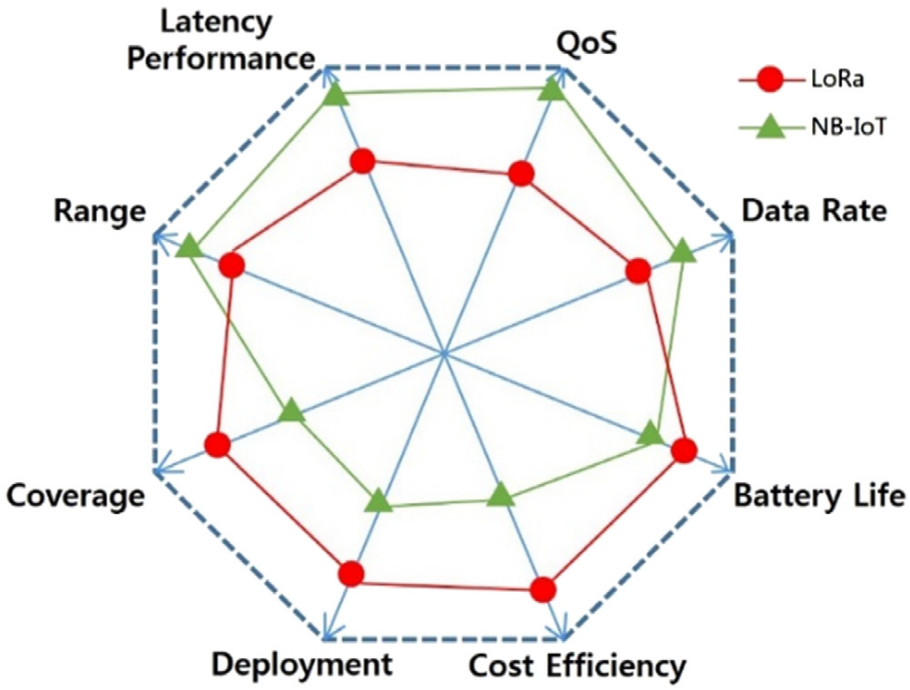

Abbildung 10: Vergleich LoRaWAN und NB-IoT [56]

Quality of Service Was die Übertragungsqualität betrifft, so ist NB-IoT LoRaWAN überlegen, da ersteres ein lizenziertes Spektrum mit synchronisierter Datenübertragung verwendet. Dieser Vorteil geht jedoch mit hohen Kosten für das Spektrum einher [56].

Stromverbrauch Was den Stromverbrauch betrifft, so ist LoRaWAN effektiver. Geräte können beliebig lang schlafen und die Modulationstechnologien verbrauchen weniger Strom. Im Ausgleich dafür hat LoRaWAN höhere Latenzen und geringere Datenübertragungsraten [56].

Reichweite und Abdeckung Während NB-IoT prinzipiell eine größere Reichweite hat, ist die Durchdringung von Gebäuden etc. schlechter. Besonders in urbanen Gebieten ist LoRaWAN also effektiver, währenddessen in offenen Bereichen NB-IoT größere Abdeckung erreicht [56].

Abdeckung NB-IoT kann die existierende LTE-Infrastruktur nutzen, wodurch Gebiete, die bereits über LTE verfügen, automatisch erfasst werden. Im Ausgleich dafür ist es teurer, die NB-IoT-Infrastruktur aufzusetzen, als dies für LoRaWAN zu tun [56].

Kosten Da LoRaWAN ein unlizenziertes Spektrum verwendet, ist die Technik deutlich günstiger als NB-IoT. Dies gilt sowohl für die Hardware, als auch für zusätzliche Einrichtungs- und Betriebskosten [56].

Kollisionen Da LoRaWAN asynchron arbeitet, besteht die Gefahr von Kollisionen während der Datenübertragung. Dies kann besonders bei großer Anzahl von Benutzern auf engem Raum für Probleme sorgen [61].

Verschlüsselung Sowohl LoRaWAN als NB-IoT verwenden symmetrische Verschlüsselung und Authentifizierung zur Absicherung von Daten. Eine genauere Analyse der verwendeten Verfahren wurde nicht durchgeführt [38] [18].

Der Vergleich der verschiedenen Punkte zeigt, dass LoRaWAN und NB-IoT sich in vielen Punkten unterscheiden und einen Fokus auf verschiedene Eigenschaften legen. So ist LoRaWAN einfacher gehalten, was sich positiv auf Stromverbrauch, Abdeckung und Kosten auswirkt, während NB-IoT verlässlichere und schnellere Datenübertragungen ermöglicht. Welche Technik besser geeignet ist, hängt von dem konkreten Anwendungsfall ab.

2.3.2 Vergleich im Kontext des Smarten Briefkastens

Während im vorherigen Kapitel die beiden Lösung allgemeinen gegenübergestellt wurden, wird im Folgenden ein Vergleich auf Basis des konkreten Anwendungsfalls des Smarten Briefkastens durchgeführt. Dabei wurden die in Tabelle 1 aufgelisteten Punkte als wichtigste Kriterien für die Datenübertragung ermittelt. Es wurde jeweils angegeben, welche der Technologien für den jeweiligen Punkt besser geeignet ist.

Kriterium	LoRaWAN	NB-IoT
Energieverbrauch	X	(X)
Verschlüsselung	X	X
Kostengünstig	X	
Reichweite	(X)	X
Abdeckung	X	(X)

Tabelle 1: Eignung von LoRaWAN und NB-IoT für den Smarten Briefkasten

Der Vergleich zeigt, dass LoRaWAN für die Verwendung in diesem konkreten Fall besser geeignet ist. Abgesehen von der Kostenfrage ist NB-IoT jedoch ebenfalls dafür qualifiziert. In dieser Arbeit wird im Folgenden ausschließlich LoRaWAN verwendet und dementsprechend nicht weiter auf NB-IoT eingegangen.

2.4 Definition Firmware und OTA-Updates

Eine Firmware wird in dem Kontext dieser Arbeit als Software-Image verstanden, welches das jeweilige IoT-Gerät steuert. Es enthält den vollständigen Code sowie alle weiteren benötigten Ressourcen. Ein Firmware-Update ist entsprechend die partielle oder vollständige Aktualisierung dieses Software-Images. Dabei wird die Firmware als opaker Datenblock angesehen; es werden also nicht nur einzelne Programme oder Bibliotheken aktualisiert, sondern immer der vollständige Code. Partielle Updates werden im Folgenden nicht weiter betrachtet.

Unter OTA-Updates wird ein Updatemechanismus verstanden, bei dem ein solches Firmware-Update automatisiert auf das IoT-Gerät transferiert und dann dort installiert wird, ohne dass eine Intervention des Nutzers prinzipiell erforderlich ist. Typischerweise geschieht eine solche Übertragung über einen drahtlosen Kanal, dies ist jedoch nicht zwingend notwendig. Im Falle von LoRaWAN erfolgt diese Übertragung über LoRa, unter Verwendung des LoRaWAN MAC Protokolls.

2.5 LDPC-Codes

Im Rahmen dieser Arbeit wird als Teil der OTA-Firmware-Updates ein Algorithmus zur Fragmentierung von Daten verwendet, der einen Fehler-Korrektur-Code nutzt. Dieser basiert auf den LDPC-Codes, weswegen deren grundlegende Eigenschaften an dieser Stelle erklärt werden. Eine ausführliche Erläuterung wird jedoch nicht gegeben. Grundlegendes Wissen bezüglich linearer Blockcodes wird vorausgesetzt. Danach wird mit höherem Detailgrad auf den davon abgeleiteten Algorithmus eingegangen, der in Unterunterabschnitt 5.3.3 verwendet wird.

2.5.1 Reguläre LDPC-Codes

Ein Low-Density-Parity-Check-Code (LDPC-Code), auch Gallager-Code genannt, ist ein linearer Blockcode mit der Eigenschaft, Fehler bei einer Übertragung korrigieren zu können [57]. Dieser Code wurde von Robert Gallager entwickelt und zuerst in dessen Dissertation beschrieben [20]. Es handelt sich bei diesem um einen Parity-Check-Code, also einen Code, bei dem an ein Datenwort Parität-Bits angehangen werden, um das Codewort zu bilden [64].

Um ein Datenwort mit einem LDPC-Code zu codieren, muss zuerst die Paritäts-Matrix C erstellt werden. Diese wird zufällig generiert, muss aber die folgenden Eigenschaften erfüllen [64].

1. Jedes Datenbit muss in gleich vielen Zeilen der Matrix vorhanden sein. Es muss also jede Spalte die gleiche Anzahl von Einsen enthalten.
2. Jede Zeile muss die gleiche Anzahl von Datenbits, also die gleiche Anzahl von Einsen, enthalten.
3. Die Dichte der Matrix hängt von der zu erreichenden Coderate R ab. Je dichter die Matrix, desto besser die Coderate.

Aus dieser Matrix C können die Kontrollmatrix $H = (C^T|I)$ und die Generatormatrix $G = (I|C)$ abgeleitet werden, mit denen die Codierung und Decodierung ermöglicht werden [57]. Dabei ist I eine Einheitsmatrix. Es gilt außerdem als zusätzliche Eigenschaft für H, dass es sich bei dieser um eine dünnbesetzte Matrix handeln muss. Diese darf also nur wenige Einsen im Vergleich zu Nullen enthalten.

Aus diesen Voraussetzungen ergeben sich einige Eigenschaften, welche die LDPC-Codes von anderen Fehler-Korrektur-Codes absetzten. Die dabei wichtigsten Eigenschaften sind:

- LDPC-Codes operieren nahe der Shannon-Grenze, also der maximalen Kapazität eines Kanals [57].
- Aufgrund der Struktur der Kontrollmatrix H ist es möglich, eine iterative Decodierung durchzuführen. Dazu wird die Kontrollmatrix H in mehrere Gruppen aufgeteilt, die parallel verarbeitet werden können [57][64].

2.5.2 Abgewandelter Algorithmus

In Unterunterabschnitt 5.3.3 wird ein Algorithmus verwendet, der auf den LDPC-Codes basiert. Dieser wird in der ”Fragmented Data Block Transport” Spezifikation beschrieben [35].

Grundidee

Die grundsätzliche Idee, die diesen Algorithmus von LDPC-Codes unterscheidet, ist die Einführung von Fragmenten. Während bei LDPC-Codes binäre Datenwörter codiert werden, werden bei diesem Algorithmus sogenannte Fragmente codiert. Ein einzelnes Fragment besteht dabei aus mehreren Bytes. Es wird weiterhin davon ausgegangen, dass bei einer fehlerbehafteten Datenübertragung Fragmente nur entweder korrekt übertragen werden oder vollständig verloren gehen. Bitfehler innerhalb eines Fragments kann dieser Algorithmus also nicht korrigieren.

Um eine Nachricht zu codieren, wird diese zuerst in m Fragmente der Länge l_f unterteilt. Jedes Fragment repräsentiert dabei ein Datenbit, die komplette fragmentierte Nachricht bildet somit das Datenwort $F = (f_1, f_2, \ldots, f_m)$, bestehend aus den Datenfragmenten. Alle Operationen, die normalerweise auf Datenbits ausgeführt werden, werden nun auf die Fragmente angewendet. Bei der Berechnung von Paritäts-Fragmenten bedeutet das, dass jeweils die kompletten Fragmente xoriert werden. Es entsteht dabei ein neues Fragment der Länge l_f.

Die Paritäts-Matrix C bleibt weiterhin eine binäre Matrix und hat die Dimension $m \times m$. Sie wird zufällig generiert, muss jedoch nicht alle Eigenschaften von regulären LDPC-Codes erfüllen. Lediglich die 2. und 3. Eigenschaft müssen eingehalten werden. Aus dieser wird eine Paritäts-Prüf-Matrix $P = (I|C^T)^T$ der Dimension $m \times n$ generiert, welche sowohl für die Codierung als auch für die Decodierung benötigt wird.

Codierung

Um eine Nachricht der Länge l_d mit diesem Algorithmus zu codieren, müssen zuerst die Fragmentlänge l_f und die Coderate R festgelegt werden. Damit berechnen sich m und n zu $m = \left\lceil \frac{l_d}{l_f} \right\rceil$ und $n = \frac{m}{R}$. Mit dieser Grundlage können die folgenden Schritte durchgeführt werden, um die Daten zu codieren. Es werden dabei aus den Datenfragmenten, welche durch die Aufspaltung der Nachricht entstanden sind, mithilfe dieser Schritte in Codefragmente umgewandelt. Die Codefragmente werden dann an den Empfänger übertragen.

1. Erstelle die Paritäts-Prüf-Matrix P. Dabei wird die Anzahl von 1 in C von der Coderate festgelegt und berechnet sich zu $m * R$.
2. Codiere jedes Codefragment c_i nach folgender Regel.

```
c_i = 0
for k=0 to m
    if P[i][k]
        c_i = c_i ^ f_k
```

3. Aus den so generierten Codefragmenten entsteht somit das vollständige Codewort $C = (c_1, c_2, \ldots, c_n)$. Dabei gilt für die ersten m Codefragmente: $c_i = f_i$.
4. Die einzelnen Codefragmente können nun nacheinander versendet werden. Dabei sollte darauf geachtet werden, dass oftmals nur ein Teil der Korrektur-Fragmente vom Empfänger benötigt wird.

Decodierung
Um die empfangenen Codefragmente decodieren zu können, muss der Empfänger ebenfalls l_f, R und P kennen. Des Weiteren muss eine Decoder-Matrix D erstellt werden. Diese hat die Dimension $m \times m$ und ist initial eine Null-Matrix. Um die Datenfragmente und somit die Original-Nachricht wiederherzustellen, müssen folgende Schritte durchgeführt werden:

1. Erstelle die Paritäts-Prüf-Matrix P und die Decoder-Matrix D.
2. Führe für jedes Codefragment c_i die folgende Berechnung durch. Dabei wird dieses mit den bereits vorhandenen Fragmenten verrechnet, um alle redundanten Informationen zu entfernen, die der Empfänger bereits kennt. Enthält das Fragment danach noch nützliche Informationen, so wird es gespeichert.

```
for k=0 to m
    if P[i][k] and D[k] != 0
        c_i = c_i ^ f_k
        P[i] = P[i] ^ D[k]
if P[i] != 0
    j = first_non_null_index(P[i])
    D[j] = P[i]
    f_j = c_i
```

3. Sobald D eine Dreiecksmatrix mit einer nur aus Einsen bestehenden Diagonalen ist, kann das Empfangen abgebrochen werden. Abhängig von der Fehlerrate des Kanals werden also weniger als n Fragmente benötigt. Falls nach dem Empfang des n-ten Fragments diese Bedingung nicht gegeben ist, wurden mehr Fragmente verloren, als der Algorithmus korrigieren kann, und es muss mit einem Fehler abgebrochen werden.
4. Im letzten Schritt werden die erhaltenen Fragmente erneut verrechnet, um aus D eine Einheitsmatrix zu erstellen. Dies wird wie folgt erreicht.

```
for k=m-1 to 0
    for l=k+1 to m
        if D[k][l]
            f_k = f_k ^ f_l
            D[k][l] = 0
```

5. Nach dieser Berechnung ist $D = I$ und alle Fragmente wurden wiederhergestellt. Das empfangene Datenwort ist somit $F = (f_1, f_2, \ldots, f_m)$.

Beispiel
Der Algorithmus soll anhand eines einfachen Beispiels erläutert werden. Dazu werden die Parameter wie folgt festgelegt.

- $d =$ "Lorem ipsum dolor si" $\equiv$ `4c6f72656d20697073756d20646f6c6f72207369`
- $l_d = 20$
- $l_f = 4$
- $R = 0,5$
- $m = 5$
- $n = 10$

Im nächsten Schritt muss die Paritäts-Prüf-Matrix P erstellt werden. Dazu werden die Zeilen $[m+1, n]$ zufällig gefüllt, mit jeweils zwei Einsen pro Zeile. Für dieses Beispiel wurde die Matrix aus Gleichung 1 generiert.

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix} \tag{1}$$

Mit dieser Matrix können nun die Codefragmente berechnet werden. Gleichung 2 zeigt die Ergebnisse der Berechnungen nach Schritt 2 der Codierung.

$$\begin{aligned} c_0 = f_0 &= \texttt{4c6f7265} \\ c_1 = f_1 &= \texttt{6d206970} \\ c_2 = f_2 &= \texttt{73756d20} \\ c_3 = f_3 &= \texttt{646f6c6f} \\ c_4 = f_4 &= \texttt{72207369} \\ c_5 = f_0 \oplus f_2 &= \texttt{3f1a1f45} \\ c_6 = f_1 \oplus f_3 &= \texttt{094f051f} \\ c_7 = f_3 \oplus f_4 &= \texttt{164f1f06} \\ c_8 = f_0 \oplus f_1 &= \texttt{214f1b15} \\ c_9 = f_2 \oplus f_4 &= \texttt{01551e49} \end{aligned} \tag{2}$$

Die so erstellten Codefragmente werden im nächsten Schritt an den Empfänger übertragen. Dabei wird in diesem Beispiel ein Paketverlust simuliert, indem die Fragmente c_1 und c_3 nicht beim Empfänger ankommen. Die Berechnung nach Schritt 2 der Decodierung ist dabei für $i < m$ trivial, da keine Berechnungen nach der ersten Schleife durchgeführt werden müssen. Der Grund dafür ist, dass für $i < m$ P nur an der Stelle P_{ii} eine Eins hat, also genau der Stelle des empfangenen Fragments, womit $D_i = 0$ gilt. Nach dem Empfang von c_0, c_2 und c_4 entspricht der Status des Empfängers Gleichung 3.

$$D = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{aligned} f'_0 = c_0 &= \texttt{4c6f7265} \\ f'_1 &= \texttt{00000000} \\ f'_2 = c_2 &= \texttt{73756d20} \\ f'_3 &= \texttt{00000000} \\ f'_4 = c_4 &= \texttt{72207369} \end{aligned} \tag{3}$$

Als Nächstes wird c_5 empfangen. $\vec{P}_5$ hat eine Eins an den Stellen 0 und 2. Bei der Verarbeitung mit Schritt 2 wird $\vec{P}'_5$ nach Gleichung 4 aktualisiert und es entsteht ein Null-Vektor. Das empfangene Codefragment enthält also keine neuen Informationen und wird verworfen.

$$\begin{aligned} \vec{P}'_5 &= \vec{P}_5 \oplus \vec{D}_0 \oplus \vec{D}_2 \\ \vec{P}'_5 &= \begin{pmatrix}1\\0\\1\\0\\0\end{pmatrix} \oplus \begin{pmatrix}1\\0\\0\\0\\0\end{pmatrix} \oplus \begin{pmatrix}0\\0\\1\\0\\0\end{pmatrix} = \vec{0} \end{aligned} \tag{4}$$

Das Codefragment c_6 wird als nächstes verarbeitet. Hier werden die gleichen Berechnungen durchgeführt. Anders als bei c_5 entsteht an dieser Stelle jedoch ein $\vec{P}'_6$, das nicht null ist, da sowohl $\vec{D}_1$ als auch $\vec{D}_3$ Null-Vektoren sind. $\vec{P}'_6$ wird somit unverändert als neues $\vec{D}_1$ verwendet, während c_6 als f'_1 gespeichert wird.

$$\begin{aligned} \vec{P}_6 &=< 0,1,0,1,0 > \\ D &= \begin{pmatrix}1&0&0&0&0\\0&1&0&1&0\\0&0&1&0&0\\0&0&0&0&0\\0&0&0&0&1\end{pmatrix} \\ f'_0 &= c_0 = \texttt{4c6f7265} \\ f'_1 &= c_6 = \texttt{094f051f} \\ f'_2 &= c_2 = \texttt{73756d20} \\ f'_3 &= \texttt{00000000} \\ f'_4 &= c_4 = \texttt{72207369} \end{aligned} \tag{5}$$

Das nächste Codefragment ist c_7. Dieses enthält ebenfalls brauchbare Informationen, da $P_{73} = 1$ ist. Bei der Anwendung von Schritt 3 wird $\vec{P}_7$ mit $\vec{D}_4$ xoriert, was die Eins an dieser Stelle entfernt. Somit ergibt sich der Status des Empfängers nach dieser Operation zu Gleichung 6.

$$\begin{aligned} \vec{P}_7 &=< 0,0,0,1,1 > \\ D &= \begin{pmatrix}1&0&0&0&0\\0&1&0&1&0\\0&0&1&0&0\\0&0&0&1&0\\0&0&0&0&1\end{pmatrix} \\ f'_0 &= c_0 = \texttt{4c6f7265} \\ f'_1 &= c_6 = \texttt{094f051f} \\ f'_2 &= c_2 = \texttt{73756d20} \\ f'_3 &= c_7 \oplus c_4 = \texttt{646f6c6f} \\ f'_4 &= c_4 = \texttt{72207369} \end{aligned} \tag{6}$$

An dieser Stelle kann die Übertragung weiterer Fragmente beendet werden, da die Voraussetzung aus Schritt 3, die Dreiecksmatrix, geschaffen wurde. Es werden nun mit Schritt 4 die Zeilen der Decoder-Matrix D miteinander verrechnet, bis eine Einheitsmatrix entsteht. Für dieses Beispiel trifft dies nur für $k = 1$ und $l = 3$ zu, was dazu führt, dass $f'_1 = f'_1 \oplus f'_3$ berechnet wird. Es entsteht somit das in Gleichung 7 zu sehende decodierte Ergebnis.

$$
\begin{aligned}
D &= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \\
f'_0 &= c_0 = \texttt{4c6f7265} = f_0 \\
f'_1 &= c_6 \oplus c_7 \oplus c_4 = \texttt{6d206970} = f_1 \\
f'_2 &= c_2 = \texttt{73756d20} = f_2 \\
f'_3 &= c_7 \oplus c_4 = \texttt{646f6c6f} = f_3 \\
f'_4 &= c_4 = \texttt{72207369} = f_4
\end{aligned}
\tag{7}
$$

Vergleicht man die so berechneten Fragmente mit denen, die ursprünglich codiert wurden, so stellt man fest, dass die Nachricht erfolgreich wiederhergestellt werden konnte: $F = f'_0|f'_1|f'_2|f'_3|f'_4 = \texttt{4c6f72656d20697073756d20646f6c6f72207369} \equiv$ ”Lorem ipsum dolor si”.

2.6 Kryptografie

In dieser Arbeit werden eine Reihe von kryptografischen Algorithmen verwendet. Diese sind im Folgenden aufgelistet, jeweils mit einer kurzen Erklärung, um was für eine Art von Algorithmus es sich handelt und wo diese in der Arbeit verwendet werden.

AES Bei dem Advanced Encryption Standard handelt es sich um einen symmetrischen Block-Verschlüsselungs-Algorithmus, der auf Blöcken von 128 Bit operiert und mit Schlüssellängen von 128, 196 oder 256 Bit verwendet werden kann [14]. Die AES-Verschlüsselung wird an mehreren Stellen in dieser Arbeit verwendet. Zum Einen ist AES der Algorithmus, den LoRaWAN für die Verschlüsselung von Daten und für die Berechnung des MICs verwendet, zum Anderen wird AES in Abschnitt 3 benutzt, um eine E2E-Verschlüsselung zu realisieren.

CTR-Modus Der CTR-Modus ist ein Verschlüsselungs-Modus, mit dem eine symmetrische Verschlüsselung betrieben werden kann. Mit diesem lassen sich Klartexte beliebiger Länge ohne Padding mit einem Block-Verschlüsselungs-Algorithmus verschlüsseln [13]. Dieser Modus wird von LoRaWAN mit AES verwendet, um Daten zwischen Endgerät und Netzwerkserver zu verschlüsseln.

GCM-Modus Der GCM-Modus ist ebenfalls ein Verschlüsselungs-Modus. Anders als der CTR-Modus ist dieser jedoch authentifizierter. Daten werden also nicht nur verschlüsselt, sondern auch mit dem GMAC gegen Manipulation abgesichert. Auch hier können Nachrichten beliebiger Länge ohne Padding verarbeitet werden. Der MAC ist 16 B lang, kann aber gekürzt werden [47]. In Abschnitt 3 wird eine Verschlüsselung verwendet, die AES im GCM-Modus verwendet.

Curve25519 Bei Curve25519 handelt es sich um ein Schlüssel-Austausch-Verfahren auf Basis von DH und der elliptischen Kurve Curve25519. Mit diesem Protokoll können

zwei Kommunikationspartner einen geheimen Schlüssel über einen unsicheren Datenkanal aushandeln [9]. Dieses wird in Abschnitt 4 verwendet, um den Schlüssel für die E2E-Verschlüsselung zu generieren.

Ed25519 Das Ed25519-Verfahren ist ein digitaler Signatur-Algorithmus auf Basis der elliptischen Kurve Curve25519. Mit diesem können Daten mit einem privaten Schlüssel signiert und mit dem zugehörigen öffentlichen Schlüssel verifiziert werden [10]. Der Algorithmus wird in Abschnitt 5 verwendet, um Updates zu signieren.

SHA-256 Es handelt sich bei SHA-256 um eine Hashsumme; also eine Prüfsumme mit zusätzlichen kryptografischen Eigenschaften. SHA-256 ist dabei eine von mehreren Varianten, die eine Prüfsumme mit einer Länge von 256 Bit generiert [15]. Als Teil des Update-Protokolls für die Signatur von Updates in Abschnitt 5 wird eine Hashsumme benötigt, wofür SHA-256 verwendet wird.

2.7 Verwendete Hard- und Software

In diesem Abschnitt soll kurz auf die verwendete Hard- und Software eingegangen werden, sowie die Gründe erläutert werden, mit denen diese ausgewählt wurden.

2.7.1 Die Arduino Plattform

Als Erstes wurde entschieden, die Software für die Endgeräte mithilfe der Arduino-Plattform zu entwickeln. Dabei gibt es zwei primäre Gründe für deren Auswahl:

Abstraktion Arduino bietet eine Reihe von Bibliotheken, die die konkret verwendete Hardware abstrahiert. So können beispielsweise die serielle Schnittstelle, SPI oder die GPIO-Pins über die Arduino-APIs angesprochen werden, unabhängig von der verwendeten Hardware.

C++-Unterstützung Arduino basiert auf C++ und kann mit diesem zusammenarbeiten. Da der Code in dieser Arbeit vorzugsweise mit C++ entwickelt werden soll, bietet Arduino eine gute Basis. Code kann mit der C++ Standard Library, unabhängig von Arduino, entwickelt werden und trotzdem noch mit der Plattform integriert werden.

Es gibt bei der Verwendung von Arduino aber auch einige Nachteile. Es verwendet sein eigenes Build-System, welches dem Entwickler nur wenig Freiraum lässt. Auch Optimierungen oder das Einbinden von Bibliotheken, die auf anderen Build-Systeme basieren, sind aufwendiger als z. B. bei CMake. Die Entwicklungsumgebung ist minimal gehalten und bietet wenig Unterstützung für den Entwickler (siehe Abbildung 11).

2.7.2 Adafruit Feather M0

Der erste Mikrocontroller, der für das Projekt verwendet wird, ist der Adafruit Feather M0 [2]. Dabei handelt es sich um ein Entwicklungsboard, welches einen ATSAMD21G18 Mikrocontroller von Atmel verbaut [8]. Dieser ist ein Arm Cortex M0+ Prozessor, der eine Taktgeschwindigkeit von bis zu 48 MHz erreichen kann, und über 32 KB statischem RAM und 256 KB Flash-Speicher verfügt.

Das Board verbaut außerdem den HopeRF RFM95 LoRa-Transceiver, auf den in Unterunterabschnitt 2.7.4 weiter eingegangen wird. Des Weiteren verfügt es über einen internen USB-Serial-Converter, wodurch eine einfache Programmierung via USB ermöglicht wird, und kann mit einem LiPo-Akku betrieben werden und diesen ebenfalls laden.

Dieses Board wurde ausgewählt, da es über einen einfachen Low-End Mikrocontroller verfügt, wodurch der Test der Funktionalität der entwickelten Software auf einem leis-

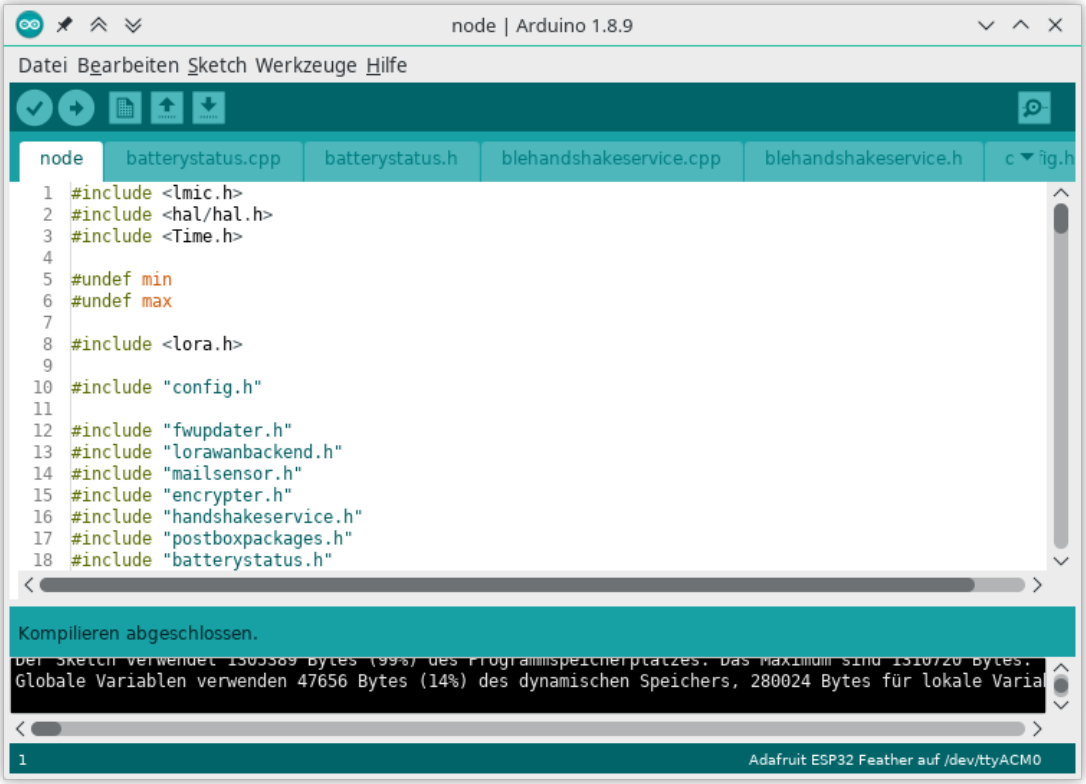

Abbildung 11: Die Arduino Entwicklungsumgebung

tungsschwachen Gerät ermöglicht wird. Darüber hinaus bietet es durch den eingebauten LoRa-Transceiver und den Akku-Anschluss ein minimales Komplettpaket, wodurch die Entwicklung vereinfacht wird.

2.7.3 Adafruit Huzzah32 - ESP32 Feather

Der zweite Mikrocontroller, der verwendet wurde, ist der Adafruit Huzzah32 ESP32 Feather [3]. Dabei handelt es sich ebenfalls um ein Entwicklungsboard, welches den leistungsstärkeren ESP32 verbaut. Dieser verfügt über eine 240 MHz Dual-Core-CPU, 520 KB RAM, und 4 MB an partitioniertem Flash-Speicher [17]. Des Weiteren verbaut das Modul über einen Bluetooth- und WLAN-Chip.

Das Board enthält jedoch keinen integrierten LoRa-Transceiver. Aus diesem Grund wird der ESP32 zusammen mit dem Adafruit LoRa-Radio verwendet, mehr dazu in Unter-unterabschnitt 2.7.4. Dieses Board wurde aufgrund seiner Unterstützung von Bluetooth ausgewählt, da dies eine Voraussetzung für die Realisierung der clientseitigen Geräte-Aktivierung aus Abschnitt 4 ist.

2.7.4 Adafruit LoRa-Radio

Für dieses Projekt wird ein LoRa-Transceiver benötigt. Es wurde entschieden, dafür auf den HopeRF RFM95 Chip zurückzugreifen [23]. Im Falle des Adafruit Feather M0s ist dieser Chip direkt in dem Entwicklungsboard verbaut; für den Adafruit Huzzah32 wurde stattdessen das Adafruit LoRa-Radio RFM95 Breakout Board verwendet, welches eben dieses Modul bereitstellt [4].

2.7.5 Arduino Bibliotheken

Neben den bereits von Arduino angebotenen Bibliotheken wurden noch einige weitere von Dritten verwendet, um bestimmte benötigte Funktionen für das Projekt bereitzustellen. Diese sind im Folgenden aufgelistet, jeweils mit dem Grund für deren Verwendung.

Arduino-LMIC library Bei dieser handelt es sich um eine Bibliothek für Arduino, welche die LoRaWAN-Spezifikation implementiert und es erlaubt, Daten über LoRaWAN mithilfe einer C-API zu versenden und zu empfangen. Diese Bibliothek wird praktisch überall im Projekt verwendet und bildet die Grundlage für die LoRaWAN-Kommunikation[1].

Arduino Cryptography Library Eine Kryptografie-Bibliothek für Arduino, welche mehrere grundlegende kryptografische Algorithmen implementiert. Genutzt werden davon die SHA-256, Curve25519, Ed25519, AES und GCM-Implementierungen an verschiedenen Stellen in der Arbeit[2].

Arduino Time Library Diese Bibliothek stellt einfache Funktionen bereit, um die internen Ticks des Prozessors in UTC-Zeit umrechnen zu lassen und erlaubt es, diese Zeit selbst zu konfigurieren. Diese Bibliothek wird für die Zeitsynchronisation, welche für Multicast-Firmware-Updates (Abschnitt 5) notwendig ist, verwendet[3].

CRC32 Eine Arduino-Bibliothek, welche den CRC32-Algorithmus bereitstellt. Diese wird nur für interne Tests der Firmware-Updates (Abschnitt 5) benötigt und ist nicht Teil der finalen Anwendung[4].

2.7.6 Qt-Framework für Client-Anwendungen

Während bisher nur auf die Entwicklung der Endgeräte eingegangen wurde, gehören zu dem Projekt auch einige Client-Anwendungen, die das Gegenstück zu denen der Endgeräte bilden. Die Client-Anwendungen werden für klassische Computer-Systeme entwickelt, unterliegen also nicht den Einschränkungen der verwendeten Hardware. Des Weiteren müssen diese nicht selbst über LoRaWAN kommunizieren, sondern stehen über MQTT und HTTP-REST mit dem Netzwerkserver in Verbindung. In dieser Arbeit werden die Anwendungen als ”Client-Anwendungen” bezeichnet, sie übernehmen aber in dem Schema von LoRaWAN die Rolle des Anwendungsservers. Dieser Unterschied wird eingesetzt, da neben den Client-Anwendungen auch noch ein generischer Anwendungsserver für Monitoring und Konfiguration verwendet wird.

Für die Entwicklung dieser Programme wurde das C++ Qt-Framework[5] ausgewählt. Die Gründe dafür sind zum einen das Vorhandensein von Bibliotheken für alle Elemente des Projekts. Qt unterstützt MQTT, HTTP-REST, BLE und Websockets. Es müssen somit keine weiteren dritten Bibliotheken für diese Aufgaben mit eingebunden werden. Zum anderen ist der Autor dieser Arbeit bereits erfahren mit dem Qt-Framework, wodurch die Entwicklungszeit für die Anwendungen reduziert wird. Da der Fokus dieser Arbeit auf den Endgeräten liegt, sind die Client-Anwendungen nur eine Notwendigkeit, weswegen die aufgeführten Gründe ausreichen, um die Verwendung zu rechtfertigen.

[1] https://github.com/mcci-catena/arduino-lmic
[2] https://rweather.github.io/arduinolibs/crypto.html
[3] https://github.com/PaulStoffregen/Time
[4] https://github.com/bakercp/CRC32
[5] https://www.qt.io/

Externe Bibliotheken und Werkzeuge

Auch für die Qt-Anwendungen wurden einige externe Bibliotheken und Werkzeuge verwendet. Diese werden im Folgenden aufgelistet.

qdep qdep ist ein Kommandozeilen-Werkzeug mit dem Abhänigkeiten verwaltet werden können. Es wird in den verschiedenen Projekten dieser Arbeit verwendet, um Bibliotheken von Dritten in die Anwendungen einzubinden. Das Werkzeug wurde vom Autor dieser Arbeit in anderen Projekten entwickelt[6].

QCliParser Bei QCliParser handelt es sich um eine Bibliothek, mit der hierarchische Kommandozeilen erstellt werden können. In Abschnitt 5 wird damit das Update-Werkzeug erstellt. Die Bibliothek wurde vom Autor dieser Arbeit in anderen Projekten entwickelt[7].

QCtrlSignals Eine Bibliothek, mit der UNIX-Signale in Qt-Signale umgewandelt werden, sodass diese einfacher verarbeitet werden können. Wird von mehreren Konsolen-Anwendungen in dieser Arbeit verwendet. Die Bibliothek wurde vom Autor dieser Arbeit in anderen Projekten entwickelt[8].

QHttp Bei QHttp handelt es sich um eine C++-Bibliothek für Qt, mit der auf einfache Weise ein HTTP-Server erstellt werden kann. Die Bibliothek wird in Abschnitt 7 benötigt, um einen solchen zu erstellen[9].

CryptoPP CryptoPP ist eine C++-Bibliothek für Kryptografie und bietet verschiedenste kryptografische Primitive an. Sie wird in Abschnitt 7 von der Client-Anwendung verwendet[10].

CryptoQQ Eine Erweiterungs-Bibliothek, welche Hilfsklassen bietet, um CryptoPP einfacher mit Qt zu integrieren. Die Bibliothek wurde vom Autor dieser Arbeit in anderen Projekten entwickelt[11].

SQLite Für die Server-Anwendung des Smarten Briefkasten wird QtSql verwendet, um auf eine SQL-Datenbank zuzugreifen. Intern wird dabei SQLite verwendet[12].

2.7.7 LoRaWAN-Server-Architekturen

Neben dem Endgerät und den Client-Anwendungen werden außerdem noch ein Gateway und ein LoRaWAN-Netzwerkserver benötigt. Für diese Aufgabe wurden in dem Projekt zwei verschiedene Varianten getestet, die im Folgenden erläutert werden.

Gateway-Hardware

Die für das Gateway verwendete Hardware ist unabhängig von der eingesetzten Lösung für die LoRaWAN-Architektur und wird daher zuerst beschrieben. Verwendet wurde ein Raspberry-Pi 2 Model B[13] mit dem WiMOD iC880A Konzentrator-Board als LoRaWAN-Schnittstelle [24]. Das Gateway wurde dabei zunächst ein GPS-Modul betrieben. Im Verlauf des Projektes wurde es jedoch notwendig, auch Class-B-Beacons zu versenden, welche ein GPS-Signal benötigen. Dafür wurde das Drotek DP0106 USB-GPS-Modul[14], welches ein U-Blox NEO-M8N verbaut [60], verwendet.

[6] https://github.com/Skycoder42/qdep
[7] https://github.com/Skycoder42/QCliParser
[8] https://github.com/Skycoder42/QCtrlSignals
[9] https://github.com/azadkuh/qhttp
[10] https://www.cryptopp.com/
[11] https://github.com/Skycoder42/CryptoQQ
[12] https://www.sqlite.org/index.html
[13] https://www.raspberrypi.org/products/raspberry-pi-2-model-b/
[14] https://store.drotek.com/DP0106

The Things Network

Bei The Things Network[15] handelt es sich um eine IoT-Cloud, die einen Netzwerk- und Anwendungsserver für LoRaWAN bereitstellt. Die Erstellung von LoRaWAN-Anwendung ist kostenlos, erfordert aber einen Account. Das Gateway musste selbst bereitgestellt werden; es wurde der TTN-Package-Forwarder[16] dafür verwendet. Es können jedoch auch bereits existierende Gateways des Netzwerks verwendeten werden, falls diese sich in Reichweite befinden.

TTN wurde bewusst ausgewählt, weil es sich bei diesem um eine IoT-Cloud handelt. Dies schafft einige Herausforderungen bezüglich der Sicherheit und verhindert die Anpassung von Anwendungs- oder Netzwerkserver, da diese nicht unter der Kontrolle des Entwicklers stehen. Im Ausgleich dafür bietet eine Cloud den Vorteil, eine höhere Abdeckung zu haben, da jedes Gateway in der Cloud von jedem Endgerät, das diese ebenfalls nutzt, verwendet werden kann. In dieser Arbeit wird nicht weiter untersucht, ob eine IoT-Cloud generell verwendet werden sollte und welche konkreten Probleme dadurch entstehen könnten. Eine Cloud wurde lediglich gewählt, um in der Arbeit die Anwendungen und Bibliotheken so entwickeln zu müssen, dass diese mit einer Cloud verwendet werden können.

LoRaServer

Als Alternative zu TTN wurde LoRaServer[17] verwendet. Dabei handelt es sich um eine Sammlung von Anwendungen, die zusammen die gleichen Rollen erfüllen wie TTN. Hier hat jedoch der Entwickler die volle Kontrolle über alle Komponenten. Es wurde ein von LoRaServer bereitgestelltes Betriebssystem-Image für den Raspberry-Pi verwendet, welches den kompletten LoRaWAN-Stack, also Gateway, Netzwerkserver und Anwendungsserver, enthält.

Der für das Projekt wichtigste Vorteil, den LoRaServer bietet, ist die Unterstützung von Class-B und Class-C Übertragungen sowie erweiterte Einstellungs-Möglichkeiten. Erstere werden nicht von TTN unterstützt, weswegen dieses nicht für die Multicast-Updates verwendet werden kann. Für Abschnitt 3 und Abschnitt 4 wurde TTN, für alle weiteren Kapitel stattdessen LoRaServer verwendet. Die Anwendungen wurden außerdem, bis auf die Schnittstelle zu TTN/LoRaServer, komplett unabhängig von diesen entwickelt.

[15] `https://www.thethingsnetwork.org/`
[16] `https://www.thethingsnetwork.org/docs/gateways/packet-forwarder/ttn.html`
[17] `https://www.loraserver.io/`

3 LoRaWAN Ende-zu-Ende-Verschlüsselung

E2E-Verschlüsselung ist ein wichtiges Element des Smarten Briefkastens. Dieser sendet regelmäßig Daten über seinen Zustand. Damit Angreifer nicht herausfinden können, ob sich Post in einem Kasten befindet, oder manipulieren können ob dies der Fall ist, müssen die Daten entsprechend verschlüsselt werden. Dabei ist das Ziel, eine Verschlüsselung über den kompletten Kommunikationspfad zwischen dem Briefkasten und der Client-Anwendung zu erreichen.

LoRaWAN legt als Protokoll bereits einen Schwerpunkt auf die Sicherheit. Dies kann unter Anderem daran festgemacht werden, dass Daten sowohl verschlüsselt als auch authentifiziert werden. Die Verschlüsselung, die bereits in Unterunterabschnitt 2.1.4 erläutert wird, ist jedoch keine vollständige E2E-Verschlüsselung. Verschlüsselt wird nur zwischen der IoT-Anwendung (dem LoRaWAN-Endgerät) und dem LoRaWAN-Netzwerkserver. Der letzte Teil der Strecke, vom Netzwerkserver zur Client-Anwendung (z. B. einem eigenen Anwendungsserver oder einer Handy-App), ist jedoch nicht Teil des Protokolls und prinzipiell unverschlüsselt. Hier liegt es in der Verantwortung des Netzwerkservers, eine sichere Übertragung zwischen diesem und der Client-Anwendung zu schaffen, wie in Abbildung 12 dargestellt.

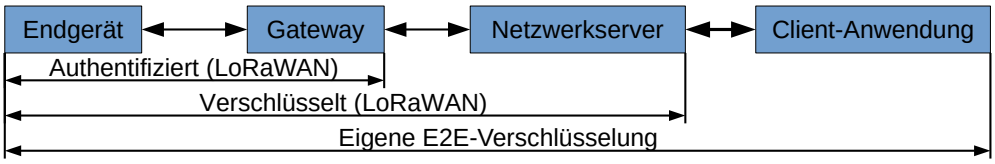

Abbildung 12: Überblick Verschlüsselung in LoRaWAN mit E2E-Verschlüsselung

Im Folgenden wird erläutert, für welche konkreten Anwendungsfälle die Verschlüsselung von LoRaWAN unzureichend ist und welche Angriffe auf diese Anwendungen ermöglicht werden. Des Weiteren wird erklärt, wie eine vollständige E2E-Verschlüsselung diese Angriffe erschweren und teilweise verhindern kann. Im Anschluss daran wird dann ein Algorithmus auf Basis des existierenden LoRaWAN-Protokolls vorgeschlagen und bezüglich Sicherheit und Effizienz untersucht.

3.1 Stand der Forschung

Nach ausgiebiger Recherche konnte festgestellt werden, dass sich in der Forschung noch nicht mit einer zusätzlichen E2E-Verschlüsselung für LoRaWAN beschäftigt wurde. Es gibt jedoch viele Arbeiten, die sich mit der Sicherheit der in LoRaWAN verwendeten Verschlüsselung beschäftigen (siehe dazu Unterunterabschnitt 2.1.6). Diese Arbeiten dienen als Basis für die hier entwickelte Verschlüsselung, sodass diese die aktuellen Schwächen von LoRaWAN ausgleichen kann.

3.2 Methodik

Um die E2E-Verschlüsselung zu entwickeln, wurde im ersten Schritt nach Angriffen gegen die existierende Verschlüsselung recherchiert. Aus diesen wurden dann Algorithmen abgeleitet, mit denen eine E2E-Verschlüsselung realisiert werden kann. Dabei wurden verschiedene Varianten ausgewählt, die alle Kriterien für eine sichere E2E-Verschlüsselung erfüllen. Untersucht werden anschließend zum einen mögliche Angriffe gegen die aus-

gewählten Varianten, zum anderen der zusätzliche Stromverbrauch, um herauszufinden, inwiefern sich zusätzliche Verschlüsselung auf den Energieverbrauch des Endgeräts auswirkt.

Für den Test der Varianten wurde zuerst ein Testprogramm erstellt, welches unabhängig von der gewählten Methode arbeitet. Dieses Programm wurde dann jeweils mit einer der Methoden ausgeführt. Dabei sendete das Programm in regelmäßigen Abständen gleichbleibende Nachrichten an die Client-Anwendung, welche mit einer identischen Nachricht antwortete. Für den Test wurden Nachrichten mit einer Länge von 10 B versendet. Dieses Programm wurde auf dem Testgerät ausgeführt und dabei der Stromverbrauch gemessen.

Diese Messergebnisse wurden dann statistisch ausgewertet und aus dem Stromverbrauch die Stromaufnahme der Endgeräte über den kompletten Sende- und Empfangsvorgang berechnet. Diese wurden für 5 verschiedene Varianten ermittelt und abschießend verglichen, um eine Aussage über die Effizienz der Algorithmen im Vergleich zu deren Nutzen treffen zu können.

3.3 Anwendungsfall E2E – Angriffs-Szenarien

Als Erstes gilt es zu ermitteln, welche Angriffs-Szenarien gegen eine solche unvollständige E2E-Verschlüsselung ermöglicht werden. Dabei gibt es mehrere verschiedene Angriffspunkte, die berücksichtigt werden müssen. Im Folgenden werden nur Angriffe gelistet, die nicht bereits durch Maßnahmen innerhalb des LoRaWAN-Protokolls verhindert oder erschwert werden. Mögliche Angriffe sind:

Angriffe gegen den Netzwerkserver Ein Angreifer kann direkt versuchen, Daten von dem Netzwerkserver, welcher diese im Klartext kennt und ggf. abspeichert, abzugreifen. Dazu bestehen viele Möglichkeiten, aber speziell betrachtet werden an dieser Stelle folgende:

- **Angriff durch einen Insider** Personen und Institutionen (im Falle der Verwendung eines Cloud-Angebots wie z. B. TTN), die Zugriff auf den Server haben, können ohne Probleme Daten auslesen, manipulieren und löschen.
- **Sicherheitslücken** Sicherheitslücken im Netzwerkserver oder dem System, auf dem dieser läuft, können es prinzipiell beliebigen Angreifern ermöglichen, Daten abzugreifen und zu manipulieren.

Man-In-The-Middle zwischen Netzwerkserver und Client-Anwendung Ein konkretes Beispiel für einen Angriff auf den Datenkanal zwischen dem Netzwerkserver und der Client-Anwendung ist der MITM-Angriff. Hier liegt es in der Verantwortung des Servers, die Verbindungen korrekt abzusichern, indem Verschlüsselung angeboten und idealerweise erzwungen wird. Ist dies nicht der Fall, werden alle möglichen Angriffe gegen diese Verbindungen möglich.

Bösartiges Gateway Besonders in Cloud-Strukturen (wie z.B. TTN) besteht oftmals keine Kontrolle über die Gateways, über die Daten an die Endgeräte gesendet werden. Die Daten sind dabei zwar verschlüsselt, jedoch nicht authentifiziert. Dies ermöglicht Bit-Flipping Angriffe, wie sie in Unterunterabschnitt 2.1.6 beschrieben werden.

Bösartige Clients Bei Clients, welche Anwendungsdaten abgreifen, wird prinzipiell keine Unterscheidung zwischen den einzelnen Clients gemacht. Es wird nur zwischen Anwendungen unterschieden. Dies bedeutet, dass beliebige Clients Daten von beliebigen Endgeräten derselben Anwendung vom Server abfragen können. Bei unzureichender Absicherung erlaubt dies einem Angreifer bei Übernahme eines einzel-

nen Clients Daten der kompletten Anwendung abzufangen und macht es möglich, Nachrichten an beliebige Endgeräte zu senden.

3.4 Auswahl des E2E Algorithmus

Um für genau diese Szenarien zusätzliche Sicherheit zu bieten, werden im Folgenden verschiedene Algorithmen diskutiert, mit denen E2E-Verschlüsselung auf Basis von LoRaWAN ermöglicht werden soll. Dabei wird im Folgenden davon ausgegangen, dass keine Kontrolle über den Netzwerkserver und das Gateway besteht. Jegliche Anpassungen zum Verbessern der Sicherheit können also nur auf dem Endgerät und in der Client-Anwendung durchgeführt werden.

Die erste Idee, die dabei aufkommt, ist eine zusätzliche Ebene der Verschlüsselung in die Anwendungen einzufügen. Dabei handelt es sich um eine grundlegende Variante, welche unabhängig von der Transport-Ebene durchgeführt werden kann. Dem gegenüber steht die Möglichkeit eines in LoRaWAN integrierten Ansatzes, der, anstatt eine zusätzliche Ebene einzuführen, die bisherige Verschlüsselung zwischen Endgerät und Netzwerkserver austauscht, ohne dass dabei Anpassungen an diesem vorgenommen werden müssen. Die theoretischen Vorteile der jeweiligen Lösungen werden in Tabelle 2 aufgelistet.

Zusätzliche Verschlüsselung	**Austausch der Verschlüsselung**
Trennung der Protokolle	Kein extra Code vonnöten
Freie Algorithmenwahl	Weniger (Energie-)Aufwand für Endgerät
Einfacher Implementiert	
Authentifizierte Verschlüsselung	

Tabelle 2: Vorteile der verschiedenen Verschlüsselungs-Varianten

Wie in Tabelle 2 zu erkennen ist, scheinen die Vorteile der zusätzlichen Ebene zu überwiegen. Der primäre Vorteil der Austausch-Variante ist dabei, dass keine zusätzlichen Ressourcen benötigt werden, weder an Speicherplatz noch an Energieverbrauch. Dessen Signifikanz wird im Evaluations-Teil (Unterabschnitt 3.7) dieses Abschnittes ermittelt.

3.4.1 Algorithmus für Austausch-Verschlüsselung

Für den Entwurf des Algorithmus wird davon ausgegangen, dass der zusätzliche AES-Schlüssel, der für die E2E-Verschlüsselung benötigt wird, über einen nicht weiter spezifizierten Kanal bereitgestellt wird. Ein möglicher Algorithmus für den Austausch von Schlüsselmaterial wird in Abschnitt 4 vorgeschlagen.

Variante 1

Die grundsätzliche Idee bei dem Algorithmus ist, dass der `AppSKey`, also der Schlüssel, der von dem Endgerät zum Verschlüsseln der Anwendungsdaten verwendet wird, mit einem eigenen, nur dem Endgerät und der Client-Anwendung bekannten, geheimen Schlüssel ausgetauscht wird. Dieser wird im Folgenden `SKey` genannt. Ansonsten bleibt die Endgerät-Implementierung unverändert. Durch diese Abänderung ergibt sich die in Abbildung 13 dargestellte Kette von Operationen und Nachrichten zwischen den einzelnen Komponenten.

Wie in Abbildung 13 zu erkennen ist, entsteht durch diese Abänderung ein Problem: Die Daten, die die Client-Anwendung vom Netzwerkserver erhält, sind sowohl mit dem neuen

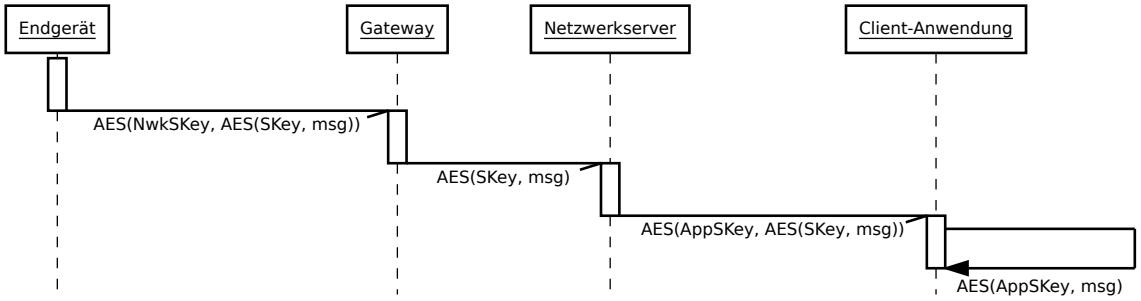

Abbildung 13: Sequenz-Diagramm Austausch-Verschlüsselung, Variante 1

`SKey`, als auch mit dem im Endgerät ausgetauschten `AppSKey`, verschlüsselt. Da die Client-Anwendung `SKey` kennt, kann sie diese Ebene der Verschlüsselung entfernen, die zweite jedoch nicht. Den `AppSKey` kennen nur das Endgerät und der Netzwerkserver, da diese den Schlüssel als Teil des Netzwerk-Beitritts aushandeln. Da es sich um AES im CTR-Modus handelt, sind die Ver- und Entschlüsselungsoperationen identisch und können in beliebiger Reihenfolge durchgeführt werden. Es ist somit kein Problem, dass die Client-Anwendung die Daten zuerst mit dem `SKey` entschlüsselt.

Der `AppSKey` trägt dabei in diesem abgewandelten Verfahren nicht zu dessen Sicherheit bei und könnte theoretisch entfernt werden. Da hier jedoch davon ausgegangen wird, dass keine Kontrolle über den Code des Netzwerkservers vorhanden ist, ist dies keine Option.

Stattdessen muss dieser Schlüssel von dem Endgerät oder dem Netzwerkserver an den Anwendungs-Client gesendet werden, um das Verfahren zu initialisieren. Falls der Netzwerkserver über solche APIs verfügt und es der Anwendung erlaubt, die aktuellen Sitzungs-Schlüssel eines Endgerätes abzufragen, kann bei Variante 1 verblieben werden. Der `AppSKey` würde in diesem Fall über einen nicht weiter spezifizierten Zweitkanal von dem Netzwerkserver an den Anwendungs-Client übertragen werden.

Variante 2

Falls der Netzwerkserver jedoch nicht über diese Möglichkeit verfügt, muss der Schlüssel von dem Endgerät an den Anwendungs-Client gesendet werden, um das Verfahren zu initialisieren. Wie in Abbildung 14 zu erkennen ist, bedeutet dies, dass das Endgerät nach jedem Beitritt in das Netzwerk eine zusätzliche Nachricht verschicken muss.

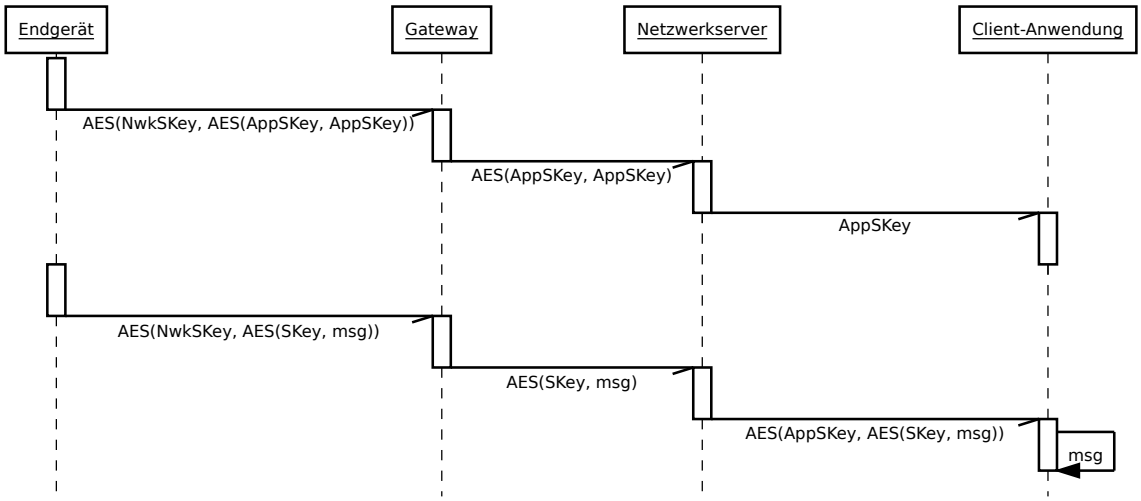

Abbildung 14: Sequenz-Diagramm Austausch-Verschlüsselung, Variante 2

Der Inhalt dieser Nachricht ist der `AppSKey`. Die Nachricht wird dabei über einen speziell dafür ausgewählten Port versendet, sodass die Client-Anwendung sofort erkennen kann, worum es sich handelt. Die Nachricht wird entsprechend nicht mit dem E2E-Verfahren verschlüsselt.

3.4.2 Wahl des IV für den Blockverschlüsselungsmodus

Für die meisten Blockverschlüsselungsmodi ist ein IV vonnöten, der mit den zu verschlüsselnden Daten verrechnet wird, sodass aus zwei identischen Quelltexten keine identischen Ciphertexte entstehen. Im Falle der Verwendung der zusätzlichen Verschlüsselung wird ein solcher IV benötigt. Für die Austausch-Verschlüsselung hingegen wird automatisch der von LoRaWAN intern verwendete IV genutzt.

Für die Verwendung in LoRaWAN ist eine der wichtigsten Anforderungen an den IV, dass dieser nicht zusätzlich zu den eigentlichen Daten versendet werden muss. Er wird stattdessen aus deterministischen Daten generiert, die sowohl das Endgerät als auch die Client-Anwendung kennen, und die sich (zumindest teilweise) für jede Nachricht verändern.

Innerhalb von LoRaWAN wird der IV aus der Adresse des Endgerätes, dem Framecounter und der Senderichtung gebildet [38]. Diese Parameter erfüllen die obigen Anforderungen und können prinzipiell somit auch für die Generierung des IVs für die zusätzliche Verschlüsselung verwendet werden. Um jedoch nicht exakt den gleichen IV zu nutzen, wurde für das erste Byte ein anderer Wert genutzt. Tabelle 3 stellt das IV-Format dar. Dabei sind `Dir` und `FCnt` Platzhalter, die in Abhängigkeit von der Senderichtung gesetzt werden.

Größe (Bytes)	1	4	1	4	4	1	1
$\mathbf{A}_i$	0x42	4 × 0x00	Dir	DevAddr	FCnt	0x00	i

Uplink:		**Downlink:**	
Dir	= 0x00	Dir	= 0x01
FCnt	= FCntUp	FCnt	= FCntDown

Tabelle 3: IV für zusätzliche Verschlüsselung

Dieser IV kann jedoch nur verwendet werden, wenn der Netzwerkserver diese Parameter an die Client-Anwendung überträgt. Besonders für `FCntDown` muss diese wissen, welcher Wert zugewiesen wird, noch bevor der Netzwerkserver dies getan hat. Solange der Client der Einzige ist, der Daten für das jeweilige Endgerät versendet, kann dieser Wert mit hoher Wahrscheinlichkeit korrekt aus dem letzten bekannten Wert, sowie der Anzahl der seitdem an den Netzwerkserver versandten Nachrichten berechnet werden. Andernfalls ist dies nicht ohne weitere Maßnahmen möglich. Wird der falsche Wert gewählt, ist die entstandene Nachricht unbrauchbar. Für `FCntUp` hingegen ist dies kein Problem, da das Endgerät selbst diesen Wert bestimmt und dieser somit bekannt ist. Es muss lediglich möglich sein, den Wert für empfangene Nachrichten vom Netzwerkserver zu erhalten.

Es gilt an dieser Stelle noch zu bemerken, dass der obige IV einige Fix-Werte enthält. Dies limitiert die Anzahl der Nachrichten, die gesendet werden können. Eine einzelne LoRaWAN-Nachricht kann maximal 256 B lang sein, i wiederholt sich also nicht. Des Weiteren dürfen nur maximal 4.294.967.296 LoRaWAN Nachrichten pro Geräte-Sitzung pro Richtung versendet werden, da sich ansonsten `FCntUp/Down` wiederholen würden.

Für GCM empfiehlt es sich, einen IV mit einer Länge von 96 Bit anstatt den sonst üblichen 128 Bit, also der Blocklänge von AES, zu verwenden. Der Grund dafür ist, dass die letzten 32 Bit als interner Zähler verwendet werden [11]. Aus diesem Grund wird im Falle der authentifizierten Verschlüsselung eine gekürzte Variante des IVs verwendet, bei der die vier 0-Bytes an der Stelle $[1;4]$ entfernt werden und das letzte Byte auf `0x00` festgelegt wird. Der IV ergibt sich somit zu `IV = 0x42 | Dir | DevAddr | FCnt | 2 × 0x00`.

3.4.3 Vorschläge für E2E-Algorithmen

Im Folgenden werden drei verschiedene Varianten für die Verschlüsselung vorgeschlagen. Diese Varianten wurden implementiert und auf deren korrekte Funktion getestet. Die Varianten sind nachfolgend dargestellt; eine vergleichende Übersicht kann in Abbildung 15 gefunden werden.

1. **Austausch-Verschlüsselung:** Der in Unterunterabschnitt 3.4.1 vorgeschlagene Algorithmus wurde verwendet, um die normale Verschlüsselung zu ersetzten.
2. **Einfache Verschlüsselung:** Eine zusätzliche Ebene der Verschlüsselung wurde mit AES-128 im CTR-Modus eingeführt, also identisch zu dem Algorithmus, der von LoRaWAN verwendet wird. Dazu wurde der IV aus Unterunterabschnitt 3.4.2 verwendet.
3. **Authentifizierte Verschlüsselung:** Eine zusätzliche Ebene der Verschlüsselung wurde eingeführt, mit AES-128 im GCM-Modus. Es handelt sich dabei um einen authentifizierten Modus; eine Manipulation der verschlüsselten Daten (z. B. durch Bit-Flipping) wird somit detektiert. Für dieses Beispiel wurde dazu der normalerweise 16 B lange Tag auf 4 B gekürzt. Auch hier wurde der IV aus Unterunterabschnitt 3.4.2 verwendet, allerdings die speziell für GCM vorgesehene 96 Bit Variante.

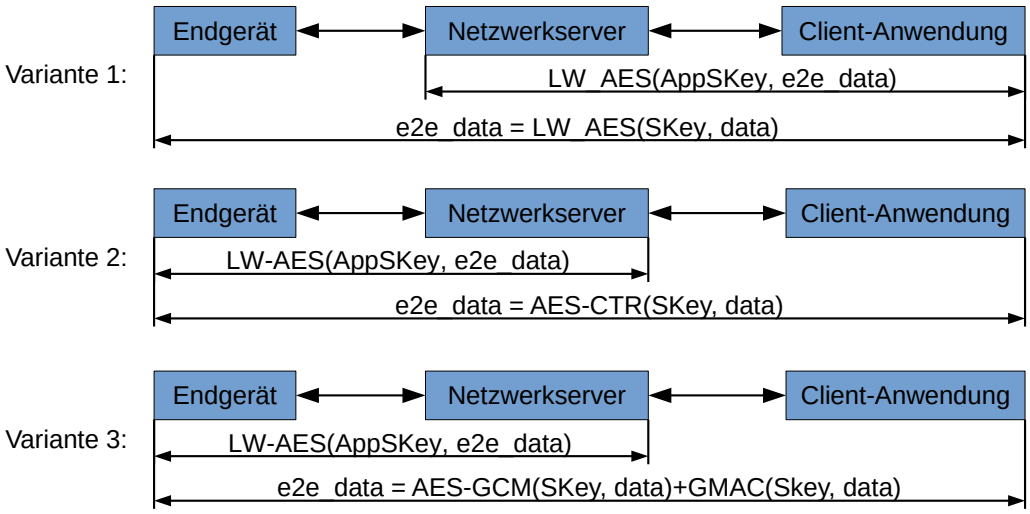

Abbildung 15: Übersicht der drei getesteten Varianten

3.4.4 Angriffe gegen die gewählten Algorithmen

In diesem Kapitel werden einige Probleme aufgelistet und diskutiert, die bei der Verwendung von zusätzlicher Verschlüsselung auftreten können. Speziell geht es dabei um mögliche Angriffe gegen diese Algorithmen. Im Folgenden werden dabei nur solche be-

trachtet, die nicht durch andere Maßnahmen in LoRaWAN selbst verhindert werden oder den Algorithmus nicht direkt angreifen. Auch generelle Angriffe gegen die unterliegenden Kryptoalgorithmen, die unabhängig von deren Einsatz sind, werden nicht betrachtet.

Bit-Flipping

Der erste Angriff, der betrachtet werden sollte, ist das Bit-Flipping. Bei diesem Angriff manipuliert der Angreifer den Ciphertext mit dem Ziel, den Klartext sinnvoll zu verändern. Für die von LoRaWAN verwendete Verschlüsselung wird der Angriff bereits in Unterunterabschnitt 2.1.6 beschrieben. Dieser ist jedoch nur bei der Verwendung des CTR-Modus (oder anderen Modi, die keine Mischung des Klartextes bei der Verschlüsselung durchführen) ohne Authentifizierung möglich.

Probleme mit kurzen Tags

Ein weiterer, wichtiger Punkt ist das Kürzen von Tags. Diese werden bei authentifizierter Verschlüsselung eingesetzt und sind meist eine Art CMAC. Bei z. B. AES im GCM-Modus entsteht auf diese Weise ein 16 B Tag, der die Integrität der Daten sichert und somit einige Angriffe, wie z. B. den Bit-Flipping Angriff, verhindert. Da bei LoRaWAN jedoch Nachrichten nur auf wenige Byte beschränkt sind und jedes Byte mehr einen höheren Energieverbrauch bedeutet, bietet es sich an, den Tag auf eine kürzere Länge, z. B. 4 B, zu reduzieren.

Dies hat den Nachteil, dass ein Angreifer schneller dazu in der Lage ist, diesen Tag zu brechen, und somit den Schutz zu umgehen. Da in diesem Anwendungsfall die Daten jedoch nur Transport-Verschlüsselt werden und es keine Wiederholung von Daten gibt, hat ein Angreifer, der keine Kontrolle über eine der Instanzen hat, nur ein begrenztes Zeitfenster, um die Daten zu manipulieren. In der Praxis sollte eine Länge gewählt werden, die genug Schutz bietet, dass ein Angreifer Nachrichten nicht ohne weiteres manipulieren kann, da andernfalls der Schutz überflüssig ist.

3.4.5 Untersuchung des Energieverbrauch der gewählten Algorithmen

Während in Unterunterabschnitt 3.4.4 die theoretischen Vor- und Nachteile der verschiedenen Algorithmen, in Bezug auf deren Sicherheit, betrachtet wurden, wurde ebenfalls der Energieverbrauch der verschiedenen Varianten untersucht. Dabei stehen zwei Fragestellungen im Fokus der Untersuchungen:

1. Bringt die Verwendung des Austausch-Algorithmus einen Vorteil im Bezug auf den Energieverbrauch im Vergleich zu einer äquivalenten extra Verschlüsselung?
2. Wie viel mehr Energie benötigen die Verwendung und der Versand von Nachrichten mit einer authentifizierten Verschlüsselung im Vergleich zu der unauthentifizierten Variante?

Diese Fragestellungen werden in den folgenden Kapiteln untersucht und beantwortet, um letztendlich einen der vorgeschlagenen Algorithmen für das Projekt zu wählen.

3.5 Implementierung

Für die Implementierung der Tests wurde zuerst ein unabhängiges Testprogramm entwickelt, welches Daten wiederholt verschlüsselt an den Netzwerkserver sendet und empfangene Daten entschlüsselt. Dafür wurde eine Schnittstelle definiert, welche die Durchführung der Verschlüsselung abstrahiert.

Anschließend wurden die drei Varianten implementiert, unter Verwendung der AES-Verschlüsselung der LMIC-Bibliothek und der AES-GCM-Verschlüsselung der Arduino Cryptography Library.

Die zugehörige Client-Anwendung wurde nach dem gleichen Prinzip entwickelt und implementiert einen einfachen Echo-Server, der alle empfangenen Nachrichten unverändert zurückschickt. Dabei werden diese zunächst entschlüsselt, um dann wieder mit einem anderen IV verschlüsselt zu werden. Zur Implementierung wurde die CryptoPP-Bibliothek verwendet.

3.6 Versuchsanordnung

Um den Stromverbrauch messen zu können, wurde das Testgerät an einen LiPo-Akku angeschlossen und eine Vorrichtung zum Messen des Stromes zwischen den beiden angeschlossen. Der Versuchsaufbau und die verwendeten Geräte werden im Folgenden beschrieben.

3.6.1 Versuchsaufbau

Der Versuchsaufbau wird in Abbildung 16 dargestellt. Das Testgerät wurde an einen gewöhnlichen 3,7V LiPo-Akku angeschlossen. Um den Stromverbrauch messen zu können wurde zwischen GND des Testgerätes und den (-)-Pol des Akkus ein Strommessgerät geschaltet. Dieses wandelt den Strom in eine messbare Spannung um, welche dann von einem Oszilloskop aufgenommen wird. Dabei wird das Oszilloskop direkt über einen der GPIO-Pins getriggert, wenn eine Übertragung beginnt oder endet. Dieser Pin wird schwebend angeschlossen, um Kurzschlüsse zu vermeiden.

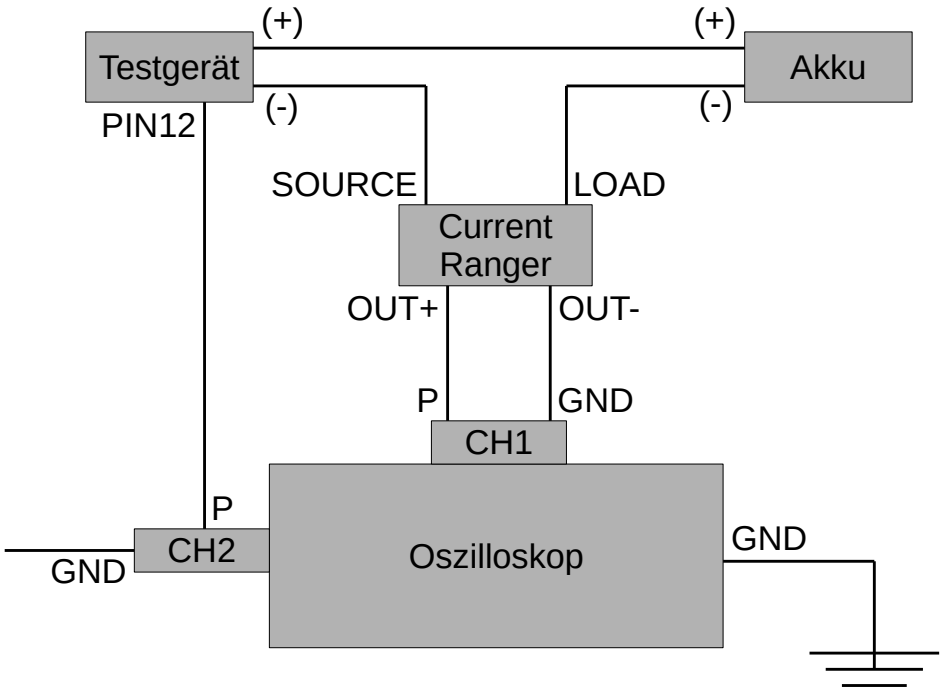

Abbildung 16: Versuchsaufbau E2E Stromverbrauchs-Messung

3.6.2 Verwendete Geräte und Konfiguration

- Testgerät: Adafruit Feather M0 LoRa-Radio
- Akku: 3,7 V LiPo Akku, 6000 mAh (PEB 126168)
- Strommessgerät: Current Ranger
  - Auflösung: mA (17 µV Bürdenspannung [43])
  - Tiefpassfilter: Aktiviert (7 kHz [41])
- Oszilloskop: Rigol DS1054
  - Kanal 1: Messdaten von Current Ranger, 20 mV, DC, 10x
  - Kanal 2: Trigger von Testgerät, 1 V, DC, 10x
  - Zeitauflösung: 200 ms
  - Trigger: Kanal 2, D = 796 ms, L = 1 V, Falling Edge
  - Optionen: High-Res, 2,5 MSa/s, 6 Mpts

3.6.3 Messung auf GND-Level

Eine Besonderheit bei diesem Versuchsaufbau ist die Tatsache, dass der Stromverbrauch im GND-Strang gemessen wird. Der Grund dafür liegt in einer Anforderung des Current Rangers: Der Eingabe- und der Ausgabe-Messkreis dürfen nicht die gleiche Erdung haben, da andernfalls ein Kurzschluss ausgelöst werden kann [42]. Dies an sich ist noch kein Problem, da der Eingabe-Schaltkreis dank des Akkus bereits von dem GND des Oszilloskops getrennt ist.

Durch das Hinzufügen des Trigger-Kanals würde jedoch diese Verbindung wiederhergestellt, weswegen der normale Aufbau (Current Ranger im (+)-Strang) nicht verwendet werden konnte. Stattdessen wurde er in den (-)-Strang gesetzt und der Trigger-Pin schwebend gemessen. Dies sorgt für ein Offset beim Trigger, was jedoch bei dieser Messung vernachlässigt werden kann, da der absolute Wert des Triggers nicht von Interesse ist.

3.6.4 Durchführung der Messungen

Es wurden insgesamt fünf Messreihen aufgenommen. Zuerst wurde der Stromverbrauch ohne extra Verschlüsselung gemessen, danach die drei Varianten aus Unterunterabschnitt 3.4.3 und anschließend der Stromverbrauch für das initiale Senden des `AppSKey` für Variante zwei der Austausch-Verschlüsselung.

Dabei wurde jeweils das Testprogramm ausgeführt, welches wiederholt Daten an die Client-Anwendung sendet und von dieser empfängt. Es wurde jeweils der Stromverbrauch vor Beginn des Sendevorgangs bis zu dessen Ende aufgenommen. Das Messergebnis besteht entsprechend aus den Verschlüsselungen, dem Sende- und Empfangs-Vorgang und der Wartezeit dazwischen. Pro Variante wurden jeweils 26 Messungen durchgeführt. Diese wurden im Multimedia-Labor der Hochschule Bonn-Rhein-Sieg durchgeführt. Eventuelle Störquellen wurden nicht berücksichtigt.

3.7 Evaluation

Im Folgenden werden die Messergebnisse ausgewertet. In einem ersten Schritt wurden dazu der Mittelwert, die Standardartabweichung, der Median, das Minimum und das Maximum für jede Messreihe berechnet. Im Anschluss wurde dann aus den Daten für jede

Messreihe der komplette Stromverbrauch berechnet und aus diesem einige statistische Werte ermittelt.

3.7.1 Auswertung der Messergebnisse

Für die Auswertung musste zuerst die am Oszilloskop gemessene Spannung in einen Strom umgewandelt werden. Diese Umrechnung erfolgt nach Gleichung 8, wie aus dem Handbuch des Current Rangers entnommen [40].

$$I = U \times 1\frac{mA}{mV} \tag{8}$$

Mithilfe dieser Information wurden die fünf Messreihen ausgewertet. Es konnte dabei festgestellt werden, dass sich diese Grafiken für alle fünf Varianten sehr ähnlich sehen. Aus diesem Grund wurde hier nur exemplarisch der Graph für die Variante der Extra-Verschlüsselung in Abbildung 17 dargestellt. Dabei wurde jeweils die momentane Stromaufnahme über die Zeit hinweg aufgetragen. Die anderen Diagramme können im Anhang gefunden werden.

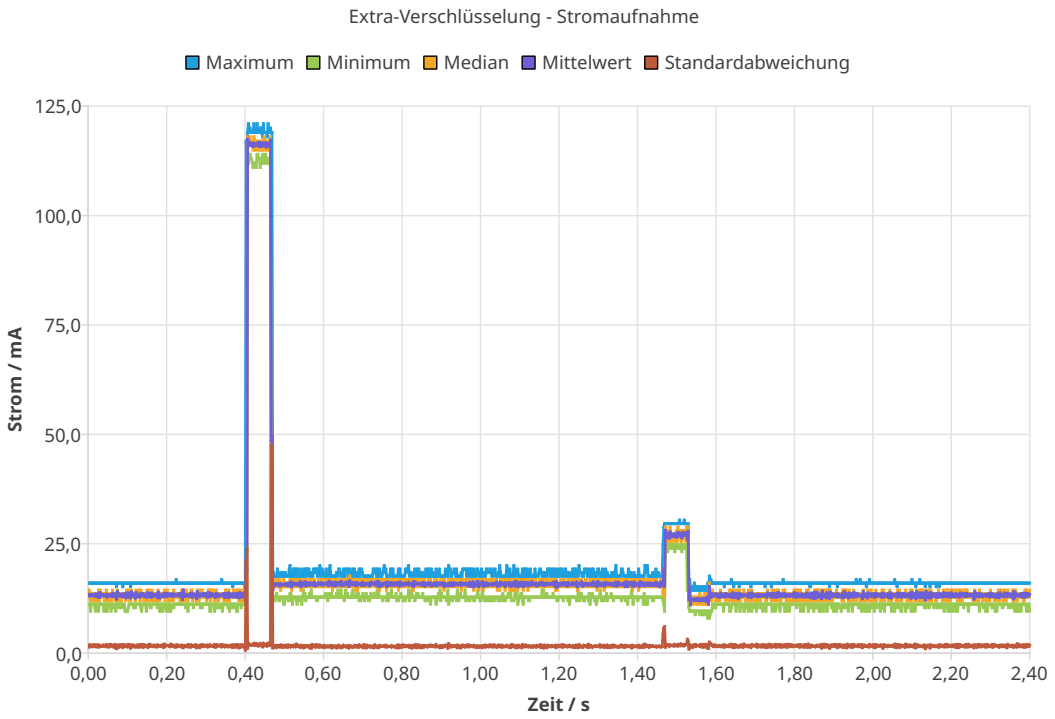

Abbildung 17: Stromverbrauch Messreihe ”Extra-Verschlüsselung”

3.7.2 Berechnung der Stromaufnahme

Um die Stromaufnahme der verschiedenen Varianten vergleichen zu können, musste diese erst berechnet werden. Es wurde vom Autor entschieden, nicht die aufgenommene Energie, sondern die aufgenommene elektrische Ladung zu berechnen. Der Grund dafür ist, dass für die Berechnung der Energie die aktuelle Spannung der Batterie benötigt würde. Da jedoch der Absolutwert für die hier durchgeführten Untersuchungen uninteressant ist und das Experiment nicht unnötig verkompliziert werden sollte, wurde stattdessen mit der Ladung gearbeitet.

Die Ladung wurde nach Gleichung 9 berechnet. Das Oszilloskop misst mit $SPS = 500$, also 500 Samples pro Sekunde. s_F ist der Index des ersten Messwertes, der vom Trigger erfasst wird, s_R der des letzten. Für die Berechnung mussten entsprechend alle Messwerte innerhalb des Intervalls $[s_F; s_R]$ gewichtet aufsummiert werden und danach mit der Zeit, welche zwischen den beiden Grenzen vergangen ist, multipliziert werden.

$$Q = I \times t$$
$$Q = \left(\sum_{k=s_F}^{s_R} \frac{I_k}{SPS} \right) \times \frac{s_R - s_F}{SPS}; SPS = 500 \tag{9}$$

Für die so berechnete Fläche wurden dann ebenfalls die statistischen Werte ermittelt und in Abbildung 18 aufgetragen. Es werden dabei in fünf Bereichen, gruppiert nach den verschiedenen Varianten, für die die Messungen durchgeführt wurden, die statistischen Werte der berechneten elektrischen Ladung dargestellt. Dabei existieren für die Austausch-Verschlüsselung zwei Gruppen. Die Linke zeigt die Stromaufnahme während des Sendevorgangs, die Rechte den für das einmalige Versenden des `AppSKey`.

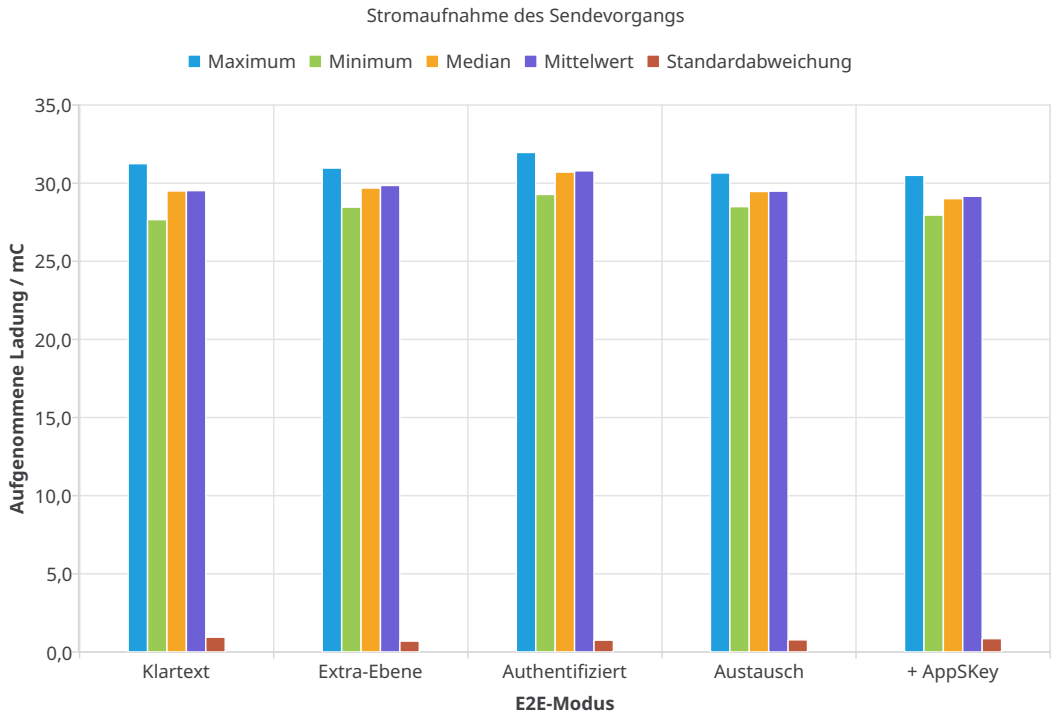

Abbildung 18: Aufgenommene elektrische Ladung aller Messreihen im Vergleich

3.8 Ergebnisse

In diesem Abschnitt werden die Ergebnisse der Messung diskutiert. Dazu wird zuerst die Qualität und Nützlichkeit der Messergebnisse ermittelt und bewertet. Danach wird auf den Unterschied im Stromverbrauch zwischen den einzelnen Algorithmen eingegangen und abschließend aus diesen Informationen der Algorithmus, welcher für den Smarten Briefkasten verwendet werden soll, ermittelt.

3.8.1 Qualität der Messergebnisse

Bei der Betrachtung der Messergebnisse für alle Messreihen fallen sofort mehrere Dinge auf. Als Erstes soll hierbei auf die Schwankungen zwischen Minimum und Maximum eingegangen werden. Diese liegen für alle Experimente bei ca. 8 mA. Ursachen können verschiedene Spannungslevel des verwendeten Akkus sowie Unterschiede im Status des Mikrocontrollers sein. Da jedoch die Standardabweichung gering ist und der Median sich nur minimal vom Mittelwert unterscheidet, handelt es sich trotzdem noch um brauchbare Messwerte.

Als Nächstes soll auf das Muster der Messungen eingegangen werden. Es sind dabei, wie in Abbildung 17 zu sehen, in jeder Messung zwei Spitzen vorhanden, welche einen deutlich höheren Stromverbrauch indizieren. Es handelt sich bei der ersten Spitze um den Sendevorgang, worauf genau 1 s später dann der Empfang von Daten folgt, was für die zweite, deutlich niedrigere, Spitze sorgt. Dies entspricht den Erwartungen, da das Senden von Daten wesentlich aufwendiger ist als das Empfangen. Dazwischen befindet sich der Mikrocontroller in einem Busy-Wait-Zustand. Dies ist von der verwendeten LMIC-Bibliothek so vorgesehen und nicht ohne eine Anpassung des Codes änderbar. Alles vor der ersten und nach der zweiten Spitze ist nicht von Interesse, da es nicht im Trigger-Bereich liegt und somit für die folgenden Berechnungen nicht verwendet wird.

3.8.2 Auswertung des Stromverbrauchs

Die endgültige Menge an Strom, die von den verschiedenen Varianten für einen Sende- und Empfangsvorgang verbraucht wird, ist in Abbildung 18 dargestellt. Dabei fällt als Erstes auf, dass alle fünf Varianten sehr ähnliche Werte haben. Vergleicht man den Stromverbrauch ohne Verschlüsselung mit dem der Extra-Ebene, so braucht die zweite Variante zwar mehr Strom, der Unterschied ist aber so gering, dass es sich dabei genauso gut um eine statistische Ungenauigkeit handeln kann. Der Unterschied der Mittelwerte beträgt 330 µC, während die Standardabweichung der unverschlüsselten Variante bei 943 µC liegt, also fast dreimal so hoch ist.

Stellt man die unverschlüsselte Variante der Austausch-Verschlüsselung gegenüber, so zeigt sich, dass die beiden nahezu identisch sind. Lediglich die Maximal- und Minimalwerte unterscheiden sich signifikant. Dies entspricht den Erwartungen, da kein Extra-Code ausgeführt wird. Dies bedeutet jedoch auch, dass im Vergleich zu der Extra-Verschlüsselung ebenfalls praktisch kein Unterschied vorhanden ist. Damit verliert die Austausch-Verschlüsselung ihren einzigen theoretischen Vorteil. Muss außerdem noch der `AppSKey` regelmäßig übertragen werden, so kann diese Variante sogar einen höheren Energieverbrauch aufweisen als die Extra-Verschlüsselung.

Die höchsten Werte für alle Auswertungen hat die authentifizierte Verschlüsselung. Vergleicht man diese mit der Extra-Verschlüsselung, so kann man einen deutlichen Unterschied erkennen. Dieser entspricht für die Mittelwerte ca. 936 µC und ist damit zwar immer noch sehr gering, aber nachweisbar. Der Grund hierfür ist jedoch nicht der höhere Rechenaufwand des Algorithmus, sondern der erhöhte Energieverbrauch durch den längeren Sendevorgang. Da GCM einen Tag von in diesem Beispiel 4 B hinzufügt, ist jede Nachricht 4 B länger als bei allen anderen Varianten.

3.8.3 Wahl eines Algorithmus

Die Messergebnisse sind eindeutig zu interpretieren. Man kann daraus folgende Schlussfolgerungen ziehen:

1. Wie erwartet ist die Austausch-Verschlüsselung genauso effizient, wie die Verwendung keiner zusätzlichen Verschlüsselung.
2. Der Stromverbrauch steigt nur minimal beim Einsatz einer Extra-Ebene der Verschlüsselung und ist somit vernachlässigbar. Die Verwendung bringt keinen Nachteil und invalidiert den einzigen Vorteil der Austausch-Verschlüsselung.
3. Die Verwendung von authentifizierter Verschlüsselung verbraucht mehr Strom, jedoch nur, weil mehr Daten gesendet werden.

Damit fallen die beiden Optionen ”Austausch-Verschlüsselung” und ”keine Verschlüsselung” sofort aus der Auswahl. Übrig bleibt also die Entscheidung, ob die Daten authentifiziert gesendet werden müssen oder nicht. Die Beantwortung dieser Fragestellung ist abhängig von der jeweiligen Anwendung und ein Trade-Off zwischen Sicherheit und Energieverbrauch. Für den Fall des Smarten Briefkastens werden regelmäßig Nachrichten gesendet, die angeben, ob Post eingeworfen wurde. Durch die Manipulation dieser Daten könnte ein Angreifer das Überfüllen des Briefkastens auslösen oder den Besitzer öfter zum Leeren auffordern als nötig. Aus diesem Grund wurde entschieden, die gesendeten Daten zu authentifizieren.

Dabei wurde eine Tag-Länge von 4 B festgelegt. Dieser Wert wurde nur exemplarisch verwendet und es wurden aus Zeitgründen keine weiteren Untersuchungen durchgeführt, ob diese Länge ausreichend ist, um Manipulationen zu verhindern.

3.8.4 Verbesserungsvorschläge

Während die Messungen für den hier vorgesehen Zweck ausreichend waren, gibt es jedoch trotzdem die Möglichkeit zur Verbesserung der Qualität der Messergebnisse. Folgende Punkte können bei zukünftigen Projekten beachtet werden, um bessere Ergebnisse zu erhalten.

- Sleep statt Busy-Wait: Der Mikrocontroller sollte in einen Schlaf-Modus wechseln, während er auf die Antwort wartet. Dies würde nicht nur den Energieverbrauch verringern, sondern auch die Messung auf die Werte beschränken, die wirklich von Relevanz sind.
- Mehr Trigger: Es könnte jeweils vor und nach der Verschlüsselung, sowie vor und nach dem Sende-/Empfangsvorgang getriggert werden. Damit könnten Verschlüsselung und Datentransfer getrennt voneinander betrachtet werden, was mehr Möglichkeiten der Analyse bedeutet.
- Variation der Parameter: Bisher wurde nur mit einer LoRa-Konfiguration gearbeitet (Spreading-Factor 7). Dies könnte variiert werden, um festzustellen, ob verschiedene Parameter einen Einfluss auf die Auswirkungen der Verschlüsselung haben.
- Messung der Akku-Spannung: Durch die Messung der Spannung des Akkus könnte die Spannung mit dem gemessenen Strom verrechnet und somit der Energieverbrauch bestimmt werden. Dies würde eventuelle Schwankungen in der Spannung des Akkus ausgleichen.
- Besseres Equipment: Bei dem Current Ranger handelt es sich um ein einfaches Messinstrument. Mit anspruchsvolleren Messgeräten, wie z. B. einem Power Analyser, lassen sich u. U. genauere Ergebnisse erzielen.

4 Clientseitige Geräte-Aktivierung

In diesem Kapitel wird sich mit einer Methode zu clientseitigen Aktivierung eines Endgerätes über einen sekundären Kanal beschäftigt. Dabei handelt es sich um ein Protokoll, mit dem einem Endgerät Sicherheitsschlüssel und Zugangsdaten für ein LoRaWAN-Netz dynamisch von einer Client-Anwendung zugewiesen werden, anstatt diese Daten bei der Herstellung in das Gerät zu programmieren[18].

Im Falle des Smarten Briefkastens ist dies eine wichtige Funktion, um eine E2E-verschlüsselte Kommunikation aufbauen zu können. Voreingestellte oder aufgedruckte Schlüssel sind für diesen problematisch, wie in den folgenden Kapiteln erläutert wird. Stattdessen wird ein sicheres Verfahren benötigt, mit dem der Briefkasten von der Client-Anwendung aus konfiguriert werden kann.

4.1 Methodik

Als erste Aufgabe für dieses Kapitel wurde analysiert, inwiefern vorprogrammierte Schlüssel für ein Endgerät von Nachteil sind, und welche Vorteile die Verwendung eines Protokolls zum Austausch dieser bringt. Aus diesen Anforderungen wurde dann ein Verfahren entwickelt, mit dem die LoRaWAN-Zugangsdaten und der geheime Schlüssel für die E2E-Verschlüsselung ausgetauscht werden können.

Um das Protokoll testen zu können, wurde eine Testanwendung entwickelt, die den Austausch von Zugangsdaten übernimmt und anschließend den generierten E2E-Schlüssel via LoRaWAN überträgt, sodass der Empfänger diesen mit seinem eigenen vergleichen kann.

Es wurde dabei mit diesem nur die korrekte Funktionalität der Implementierungen des Protokolls getestet, weitere Analysen wurden nicht durchgeführt.

4.2 Vorteile der clientseitigen Aktivierung

Die erste Frage, die sich hier stellt, ist, welchen Vorteil die Verwendung eines solchen Protokolls bringt. Dazu konnten die in den folgenden Kapiteln erläuterten Punkte ermittelt werden.

4.2.1 Angriffe gegen Hersteller-Schlüssel

Falls Schlüssel oder anderweitige Zugangsdaten nicht dynamisch zugewiesen werden, müssen diese vom Hersteller in die Firmware einprogrammiert werden. Dies hat gleich mehrere Nachteile.

Zum einen kann ein solcher Schlüssel nicht ohne weiteres geändert werden. Falls ein Angreifer in der Lage ist, einen Schlüssel auszulesen, so besteht keine Möglichkeit für den Nutzer, einen neuen Schlüssel zu generieren. An dieser Stelle müsste der Hersteller reagieren.

Zum anderen haben vom Hersteller generierte Schlüssel oftmals Sicherheitsprobleme. Fehlerhafte Zufallsgeneratoren sind ein typisches Beispiel, was dazu führt, dass viele Geräte ähnliche oder sogar identische Schlüssel besitzen [49]. Der Nutzer hat keine Kontrolle über die Qualität der Schlüssel.

[18] Im ursprünglichen Exposé war für dieses Kapitel eine Alternative zu der JOIN-Prozedur des LoRaWAN Protokolls vorgesehen. Diese Idee basierte jedoch auf der Falschannahme, dass diese nur einmalig pro Gerät, und nicht einmal pro Neu-Verbindung mit dem Netzwerk stattfindet. Aus diesem Grund wurde stattdessen ein Protokoll entwickelt, um die Zugangsdaten für diese Prozedur auszutauschen.

Beim Einsatz von echter E2E-Verschlüsselung entsteht außerdem ein weiteres Problem. Der Hersteller kennt den Schlüssel und speichert diesen möglicherweise in einer internen Datenbank. Auch hier hat der Nutzer keine Kontrolle über die Schlüssel und somit keine Garantie, dass diese sicher oder geheim sind.

4.2.2 Perfect Forward Secrecy

Bei fest einprogrammierten Schlüsseln liegt die Verantwortung über den regelmäßigen Austausch von diesen zum Einhalten von PFS in den Händen des Herstellers. Der Nutzer hat also keine Möglichkeit, selbst für dafür zu sorgen. Bei dynamischen Schlüsseln hingegen kann der Nutzer jederzeit seine eigenen austauschen.

4.2.3 Schlüssel-Verwaltung

Ohne ein solches Protokoll müsste ein zuvor vom Hersteller generierter E2E-Schlüssel dem Endgerät beigelegt werden. Dabei gibt es verschiedene Möglichkeiten, die oftmals große Nachteile haben.

So könnte zum Beispiel eine Karte mit dem Schlüssel beigelegt werden. Hier besteht das Risiko, diese zu verlieren, oder, bei der Verwaltung mehrerer Geräte, Karten durcheinanderzubringen. Eine andere Möglichkeit ist, einen Code auf das Gerät zu drucken. Dies macht jedoch Probleme, denn sobald ein Gerät eingebaut ist, sind diese Codes in vielen Fällen entweder nicht mehr einfach zu erreichen, oder öffentlich einsehbar, was ein Sicherheitsrisiko ist.

Bei der Verwendung von Curve25519 über das vorgeschlagene Protokoll hingegen, können Angreifer nicht ohne Weiteres an die ausgehandelten Schlüssel kommen. Außerdem sind die Erreichbarkeit des Gerätes sowie die Verwaltung von Schlüsseln kein Problem, da diese jederzeit einfach neu ausgehandelt werden können.

4.3 Nachteile der clientseitigen Aktivierung

Neben den zuvor genannten Vorteilen gibt es jedoch auch einige Nachteile, die der Einsatz von dynamischen Schlüsseln bringt.

Zum einen bedeutet dies, dass der Nutzer das Gerät einmalig einrichten muss. Je nach Anforderungen des Protokolls benötigt er dafür weitere Geräte. Dies kann dazu führen, dass einige Nutzer Geräte nicht aktivieren können.

Zum anderen liegt damit auch die Verantwortung über die Schlüssel beim Nutzer. Dieser muss nur die Schlüssel sicher verwalten und regelmäßig aktualisieren. Im Fall von Hersteller-Updates, die eine Aktualisierung der Schlüssel voraussetzen, muss der Nutzer aktiv werden, da sonst das Endgerät möglicherweise nicht mehr funktionsfähig ist.

4.4 Protokoll

Der Ablauf des Protokolls ist bewusst simpel gehalten und wird in Abbildung 19 dargestellt. In diesem Abschnitt wird nur das Protokoll erläutert, ohne auf die Übertragungs-Protokolle einzugehen.

Als Erstes wird dabei, nach dem Verbindungsaufbau des Sekundär-Kanals, eine ID des Endgerätes, sowie optional die öffentlichen Parameter des Curve25519-Verfahrens, an die Client-Awendung übertragen. Die Geräte-ID ist dabei eine vom Protokoll intern verwendete UUID, die unabhängig von LoRaWAN verwaltet wird und zur Identifizierung des

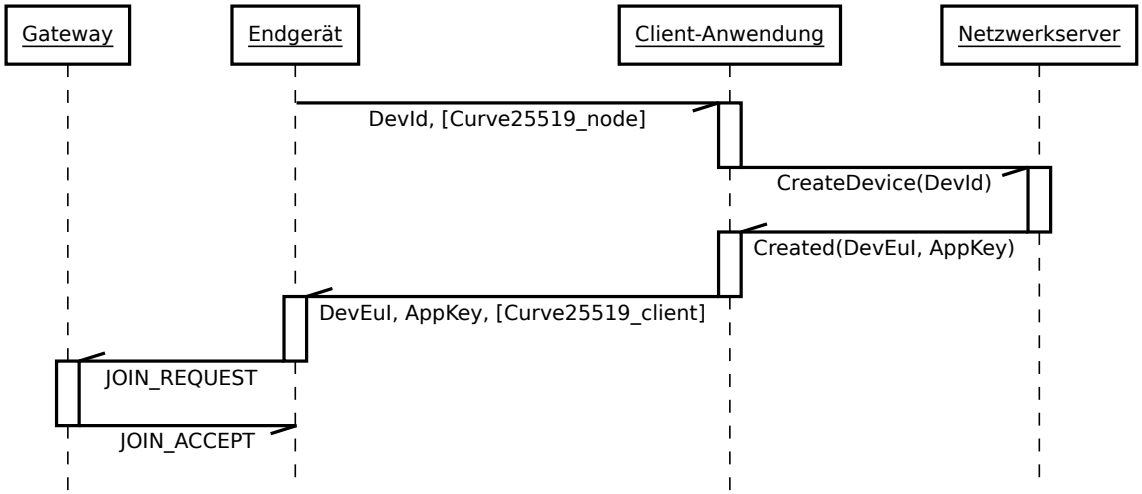

Abbildung 19: Sequenz-Diagramm Protokoll clientseitige Geräte-Aktivierung

Endgerätes dient. Optional kann hier auch noch die `AppEuI` mit übertragen werden, um das Gerät einer LoRaWAN-Anwendung zuordnen zu können.

Im nächsten Schritt meldet die Client-Anwendung das Endgerät beim Netzwerkserver an und erhält von diesem die Zugangsdaten der LoRaWAN-JOIN-Prozedur für das Endgerät. Dabei handelt es sich um die `DevEuI` und den `AppKey`. Diese werden dann, zusammen mit den Curve25519-Parametern der Client-Anwendung, an das Endgerät übertragen.

Ab diesem Zeitpunkt ist der Transfer-Part des Protokolls beendet. Beide Seiten generieren abschließend noch aus den Curve25519-Parametern das geteilte Geheimnis, welches als Schlüssel für eine optionale E2E-Verschlüsselung dient. Das Endgerät sollte nun die erhaltenen Daten permanent speichern und kann sich im LoRaWAN Netzwerk anmelden.

4.4.1 Wahl des Sekundär-Kanals

Das Protokoll ist prinzipiell unabhängig von dem gewählten Sekundär-Kanal zur Übertragung der Daten zwischen Endgerät und Client-Anwendung. Es empfiehlt sich jedoch ein Verfahren zu wählen, welches von allen Geräten, auf denen die Client-Anwendung ausgeführt werden soll, unterstützt wird und drahtlos arbeitet. Mögliche Protokolle wären beispielsweise W-LAN, Bluetooth, BLE oder NFC.

Für dieses Projekt wurde entschieden, BLE zu verwenden, da dieses Protokoll bereits von der verwendeten Hardware unterstützt wird und keinen großen Aufwand benötigt, um eine einfache Kommunikation zwischen zwei Partnern herzustellen.

4.4.2 Bluetooth Low-Energy

Für die Implementierung wird das BLE-Protokoll verwendet. Dieses soll an dieser Stelle kurz erläutert werden, um dem Leser ein grundlegendes Verständnis über dessen Struktur zu geben. Die Erläuterungen basieren auf dem Internet-Guide von Adafruit Learn [5].

BLE basiert auf dem GATT-Protokoll. Dieses stellt eine hierarchische Struktur bereit, über die Daten transferiert werden. Dabei sind die teilnehmenden Geräte in zwei Gruppen aufgeteilt, wie in Abbildung 20 dargestellt wird. Die Peripheral Geräte stellen die GATT-Definitionen bereit und bestimmen somit die Form und Interpretation der zu übertragenden Daten. Auf der anderen Seite stehen die Central Geräte, welche sich mit mehreren Peripheral Geräten verbinden können, um Daten auszutauschen.

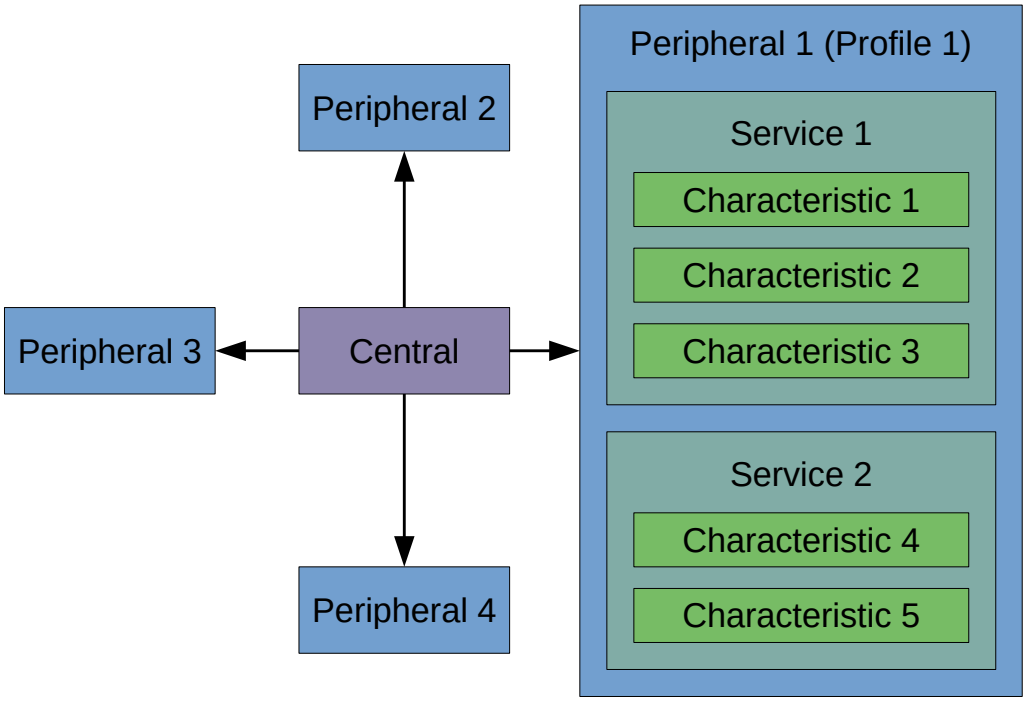

Abbildung 20: Übersicht über die Struktur von BLE und GATT

Eine GATT-Definition besteht zuerst aus einem Profil. Dieses beschreibt eine logische Zusammenstellung von mehreren Services. Diese wiederum bieten mehrere logisch zusammenhängende Datenpunkte an, welche durch Charakteristiken repräsentiert werden.

Bei der Datenübertragung verbindet sich das Central Gerät mit einem Peripheral Gerät und ermittelt die angebotenen Services. Danach kann es deren Charakteristiken abfragen, um herauszufinden, welche Daten angeboten werden. Ab diesem Zeitpunkt sind beide Seiten bereit zur Datenübertragung. Dazu lesen und schreiben diese die beiden Werte der Charakteristiken, um sie auszutauschen.

4.5 Implementierung

Es wurden sowohl der Code für das Endgerät als auch für die Client-Anwendung implementiert, um einen Austausch von Daten zu ermöglichen. Dabei wurde für die Implementierung des Endgerätes auf die BLE-Implementierung des ESP32s und die Arduino Cryptography Library für Curve25519 zurückgegriffen. Für die Implementierung der Client-Anwendung wurden QtBluetooth- und die CryptoPP-Bibliothek verwendet.

Für den Austausch der Daten via BLE wurde ein BLE-Profil definiert. Dieses besteht aus nur einem Service, welcher die folgenden vier Charakteristiken anbietet. Dabei steht `svc` für ”BLE-Service” und `char` für ”BLE-Charakteristik”.

svc BLE-LoRaWAN-Join (`b3e0734d-ee3d-4d8c-9be1-45d9445402aa`)

- **char** DevId (`e8f7c400-fc6d-4063-b21e-07292d81ef0f`, nur lesbar)
- **char** ECDHNodeKey (`58a90d66-d45a-409a-b0a9-b981827ce9b1`, nur lesbar)
- **char** LoRaWAN-Params (`0db9490e-ec02-4bf9-b503-c3943e5d7d92`, nur schreibbar)
- **char** ECDHClientKey (`b42fba6e-093b-4815-8434-d841c2447d2d`, nur schreibbar)

Die ersten beiden Charakteristiken werden von der Client-Anwendung ausgelesen, um die entsprechenden Daten vom Gerät zu erhalten. Nachdem das Gerät im LoRaWAN-Netz angemeldet wurde, können dann die Parameter, die das Gerät für die Verbindung mit diesem benötigt, über die anderen beiden Charakteristiken an das Gerät gesendet werden. Dabei ist LoRaWAN-Params definiert als: `LoRaWAN-Params = DevEuI | AppKey`.

4.6 Evaluation

Als Teil der Evaluation wurde zuerst eine abgewandelte Version des Protokolls getestet. Bei dieser wurden `DevEuI` und `AppKey` mit jeweils einer eigenen Charakteristik bereitgestellt, was insgesamt fünf Charakteristiken ergab. Dies führte dazu, dass es nicht mehr möglich war, überhaupt irgendwelche Daten auszutauschen. Während die Anzahl der Charakteristiken als Auslöser für diesen Fehler ausgemacht werden konnte, wurde nicht weiter untersucht, wo genau der Fehler lag. Stattdessen wurden die beiden Charakteristiken zu einer kombiniert.

Diese zweite Variante funktionierte dann jedoch wie erwartet und es konnte ein Gerät erfolgreich einem LoRaWAN-Netz hinzugefügt werden. Auch der Austausch der Curve25519-Parameter und die Generierung des geteilten Geheimnisses konnte erwartungsgemäß durchgeführt werden.

4.7 Ergebnisse

Abschließend lässt sich über das Protokoll sagen, dass es die geplanten Funktionen erfüllt und somit für den Einsatz als clientseitige Geräte-Aktivierung geeignet ist. Des Weiteren wurde entschieden, diese Funktion auch für den Smarten Briefkasten zu verwenden, um eine sichere und einfache Methode zur Verfügung zu haben, um diesen aufzusetzen und den Schlüssel für die E2E-Verschlüsselung auszutauschen.

Was in dieser Arbeit nicht untersucht wurde, ist die Verwaltung von mehreren parallelen Sitzungen sowie die praktische Umsetzung der Aktivierung und Deaktivierung des Bluetooth-Kanals durch Nutzer-Interaktion. Dafür sollte eine Lösung eingesetzt werden, die den Kanal standardmäßig geschlossen hält, und nur auf Anforderung des Nutzers öffnet. Ist diese fehlerhaft, so kann ein Angreifer u. U. das Schlüsselmaterial gegen eigenes austauschen. Auch in diesem Fall kann der Angreifer jedoch kein altes Schlüsselmaterial erhalten, er kann lediglich neues festlegen.

5 Firmware-Updates via LoRaWAN

Firmware-Updates sind ein integraler Bestandteil von sicheren IoT-Anwendungen. Dies gilt nicht anders für den Smarten Briefkasten. Im Falle von Funktionsupdates, Fehlerbehebungen oder dem Schließen von Sicherheitslücken ist es unabdingbar, dass der Briefkasten Updates durchführen kann. Dies muss im Hintergrund und ohne Interaktion des Nutzers erfolgen. Außerdem müssen Updates gegen Manipulation gesichert sein.

In diesem Kapitel wird ein Protokoll zur sicheren Übertragung von Firmware-Updates über das LoRaWAN-Protokoll entwickelt, implementiert und untersucht. Dabei wird nur die eigentliche Übertragung des Updates an das Endgerät genauer betrachtet. Die konkrete Speicherung der Update-Daten sowie die Durchführung des Updates werden hier nicht betrachtet.

5.1 Stand der Forschung

Im Folgenden wird herausgestellt, welche besonderen Herausforderungen an OTA-Updates durch den Einsatz via LoRaWAN gestellt werden, und welche Bemühungen in diesem Bereich bereits von anderer Seite erfolgt sind. Als Teil der Recherche wurde zuerst versucht, existierende Protokolle für OTA-Updates zu finden und zu prüfen, ob diese auch via LoRaWAN angewendet werden können. Dazu wurden zuerst klassische, IP-basierte Protokolle betrachtet und danach konkret nach Protokollen für LoRaWAN recherchiert.

5.1.1 Herausforderungen an OTA-Updates via LoRaWAN

Bei dem Design und der Implementierung eines OTA-Updatemechanismus mit LoRaWAN als Protokoll zur Datenübertragung entstehen einige besondere Herausforderungen, die bewältigt werden müssen. Auf diese wird im Folgenden eingegangen und es wird erläutert, warum sie problematisch sind, sowie mögliche Lösungsansätze vorgeschlagen.

Transport von großen Daten
Die erste Herausforderung, die LoRaWAN an ein solches Protokoll stellt, ist das Nicht-Vorhandensein eines standardisierten Transport-Protokolls (OSI-Layer 4). LoRaWAN selbst ist ein MAC-Protokoll (OSI-Layer 2) mit einigen Elementen eines Netzwerk-Protokolls (OSI-Layer 3) und basiert auf LoRa, einem physikalischen Protokoll (OSI-Layer 1) [50][51]. LoRaWAN bietet entsprechend keine Funktionen zur Verwaltung von Datenströmen, die mehr als nur ein einzelnes Datenpaket benötigen.

Die Größe eines LoRaWAN-Paketes hängt von dem gewählten Spreading-Faktor ab. Es können Datenpakete mit einer Größe von, im schlechtesten Fall, maximal 51 B versendet werden; unter optimalen Bedingungen können es maximal 240 B pro Datenpaket werden [36]. Dies ist weit unter der Größe, die Firmware-Updates typischerweise benötigen. Ein einfaches Code-Beispiel, welches nur eine LED blinken lässt, benötigt beispielsweise auf dem Adafruit Feather M0 ca. 10 KB.

Des Weiteren unterstützt LoRaWAN keine Wiederholung von Paketen. Während der Übertragung verloren gegangene oder beschädigte Pakete werden nicht detektiert, sondern nur still verworfen, und somit auch nicht behandelt. Es wird also ein Algorithmus benötigt, welcher dazu in der Lage ist, große Dateien zuverlässig über mehrere LoRaWAN-Pakete verteilt zu übertragen, selbst wenn dabei einige dieser verloren gehen.

An dieser Stelle gilt es zu entscheiden, welche Eigenschaften ein solcher Algorithmus besitzen muss. Prinzipiell gibt es zwei Möglichkeiten, paketierte Daten verlustfrei über einen verlustbehafteten Kanal zu übertragen: Fehlererkennung in Kombination mit der Wiederholung von fehlerhaften Paketen, oder die Verwendung eines Algorithmus mit der

Möglichkeit, Fehler zu korrigieren. Auf die Wahl des Algorithmus wird in Unterunterabschnitt 5.3.3 eingegangen. Allerdings sollte hier bereits erwähnt werden, dass der Algorithmus in der Lage sein muss, vollständige Pakete zu korrigieren, da Pakete mit fehlerhaften Bits bereits von der LoRaWAN-Schicht aussortiert werden.

Sende-Beschränkungen aufgrund der relativen Frequenzbelegungsdauer

Eine weitere Beschränkung, die beachtet werden muss, ist die relative Frequenzbelegungsdauer (auch ”Duty-Cycle” genannt), die für die von LoRa in der EU verwendeten Frequenzbänder gilt. Diese legt fest, wie lange einzelne Geräte eine bestimmte Frequenz belegen dürfen und wird von der Bundesnetzagentur festgelegt [12].

Werden beispielsweise Updates über ein Band mit einem Duty-Cycle von 1% übertragen, dann kann maximal 36 s/h gesendet werden. Je nach der gewählten Datenrate der Übertragung kann LoRaWAN zwischen 250 und 11.000 Bit/s transferieren [36]. Bleibt man bei dem Beispiel des 10 KB Updates, dann benötigt ein solches Update bei minimaler Übertragungsgeschwindigkeit 9 h, von denen nur 320 s Daten übertragen werden.

Dies ist für die Wahl des Update-Algorithmus insofern relevant, dass ein Algorithmus gewählt werden sollte, der die von den Gateway versendeten Datenpakete minimiert. Dazu sollte die Wiederholung von Paketen vermieden und vorzugsweise eine Multicast Datenübertragung verwendet werden, sodass die Daten nicht an jedes Gerät einzeln versendet werden müssen.

Wahl des Daten-Übertragungs-Modus

LoRaWAN stellt drei verschiedene Methoden zur Datenübertragung zur Verfügung (Siehe Unterunterabschnitt 2.1.3). Da die Übertragung eines Updates größtenteils nur in Richtung Downlink geschieht, empfiehlt es sich, Geräte temporär in den Class-B oder Class-C Modus zu versetzen. Die Übertragung via Class-A sollte aber trotzdem noch unterstützt werden, da nicht alle Gateways und Endgeräte die Anforderungen erfüllen, um in einem der anderen Modi zu arbeiten.

Die dabei signifikanteste Anforderung ist die, dass Gateways, die Class-B Daten verwenden wollen, über ein GPS-Modul verfügen müssen, um gewisse Datenpakete (genannt Beacons) zeitgenau versenden zu können [38].

Multicast in LoRaWAN

Das LoRaWAN Protokoll sieht eine Verschlüsselung von Daten zwischen dem Endgerät und dem Anwendungsserver vor und ist primär auf die Übertragung von Daten im Unicast ausgelegt. Für Übertragungen im Class-B oder im Class-C Modus ist jedoch die Verwendung von Multicast-Paketen möglich [38]. Viele der Aspekte, die dazu vonnöten sind, werden jedoch nicht von der LoRaWAN-Spefizikation festgelegt, sondern bewusst an die Anwendungs-Ebene weitergereicht. Somit bleiben entsprechend einige Aufgaben übrig, die von den Anwendungen selbstständig erfüllt werden müssen. Die wichtigsten Aufgaben, die auf jeden Fall erfüllt werden müssen, sind dabei:

1. Erstellung der Multicast-Gruppe: Geräte müssen dynamisch zu der Gruppe hinzugefügt und entfernt werden können. LoRaWAN-Parameter, wie die Multicast-Adresse, Frequenz, Datenrate und AES-Schlüssel, müssen mit den Geräten ausgetauscht werden.
2. Zeit-Synchronisation: Damit alle Geräte gleichzeitig in einen Modus wechseln, in dem sie in der Lage sind, Multicast-Nachrichten zu empfangen, müssen die Uhren der Geräte mit der des Netzwerkservers synchronisiert werden.

3. Übertragungsmodus: Alle Geräte in der Gruppe müssen zum richtigen Zeitpunkt entweder in Class-B oder Class-C wechseln. Damit ein echter Multicast stattfinden kann, müssen alle Geräte in den gleichen Modus wechseln.

5.1.2 Anwendbarkeit von klassischen OTA-Updates

Bei der Recherche nach klassischen OTA-Verfahren konnte sehr schnell festgestellt werden, dass ein TCP/IP-basiertes Update-Protokoll nicht ohne weiteres auf LoRaWAN portiert werden kann. Die Gründe dafür wurden bereits in Unterunterabschnitt 5.1.1 aufgezählt, aber zusammenfassend kann erneut gesagt werden, dass das Fehlen einer Transportschicht in Kombination mit einigen anderen Faktoren diese Protokolle inkompatibel macht. Es wurde an dieser Stelle entschieden, nicht weiter nach klassischen Verfahren zu suchen, und stattdessen speziell nach solchen für LoRa oder LoRaWAN zu suchen.

5.1.3 Protokolle für LoRaWAN

Konkret für LoRaWAN wurde dabei nur ein Update-Mechanismus ausfindig gemacht. Dieser stammt von mbedOS und beschreibt ein Update-Protokoll, welches Daten verlässlich und sicher via Multicast übertragen kann [25]. In dem Blogpost wird auf eine zugehörige Referenz-Implementierung auf Basis von mbedOS verwiesen sowie ein Paper verlinkt, das das gleiche Protokoll noch einmal formaler beschreibt [26]. Beide Quellen behandeln jedoch nur die grundsätzliche Idee, kein konkretes Protokoll. Die Implementierung wurde untersucht, um festzustellen, ob diese als Basis für dieses Projekt dienen kann. Es wurde dabei festgestellt, dass die Implementierung sehr stark von mbedOS abhängt, was eine Portierung eher schwierig macht. Stattdessen wurde entschieden, die Grundidee des Ansatzes wiederzuverwenden und daraus ein eigenes Protokoll zu entwickeln.

Auf Basis des zuvor beschriebenen Protokolls wurden von der LoRa Alliance außerdem drei Spezifikationen veröffentlicht, die einige der Grundlagen des Update-Verfahrens vereinheitlichen, um somit eine standardisierte Basis bieten zu können [33]. Diese werden in Unterabschnitt 5.3 untersucht, um festzustellen, ob sie als Basis für die eigene Implementierung verwendet werden können.

5.2 Methodik

Um ein verlässliches und sicheres Update-Protokoll zu erstellen, wurden als erster Schritt die bereits existierenden Protokolle, welche hilfreich bei der Entwicklung eines solchen sein können, analysiert. Dabei wurde untersucht, welche Funktionen die Protokolle bieten und inwiefern diese sich eignen, um OTA-Updates zu verteilen.

Im Anschluss wurde aus diesen Protokollen ein vollständiges Update-Protokoll entworfen, welches alle Aufgaben zum Transferieren und Validieren der Updates über LoRaWAN übernimmt. Dabei wurde unterschieden, welche Teile des Protokolls allgemein benötigt werden, und welche nur für den Transfer von Updates via Multicast relevant sind.

Der nächste Schritt war dann die Implementierung des Protokolls, sowohl für Unicast, als auch für Multicast. Dafür wurden zuerst die existierenden Protokolle implementiert, unabhängig von deren Verwendung für das Update-Protokoll. Erst danach wurde aus diesen dann das Update-Protokoll implementiert. Anschließend wurde noch ein Kommandozeilen-Werkzeug entwickelt, mit dem ein Client Updates für Endgeräte bereitstellen kann.

Der Test des Update-Protokolls fand in mehreren Schritten statt. Zuerst wurden die unterschiedlichen Basis-Protokolle getrennt voneinander implementiert und getestet. Dazu wurden jeweils Testprogramme erstellt, um die korrekte Funktionsweise der Protokolle zu

verifizieren. Anschließend wurde das Update-Protokoll implementiert und dazu ebenfalls ein Testprogramm entwickelt. Dabei wurde zuerst der Unicast-Transfer getestet und danach der Multicast-Transfer. Abschließend wurde das Update nochmal unter Verwendung des Kommandozeilen-Updaters übertragen, um auch dessen korrekte Funktion sicherzustellen.

Nachdem die Funktionalität auf diese Weise gewährleistet wurde, konnten weitere Tests durchgeführt werden, um folgende Eigenschaften zu ermitteln. Dabei wurde das Update-Testprogramm verwendet, mit kleinen Anpassungen, falls für den jeweiligen Test notwendig. Diese wurden sowohl praktisch ermittelt als auch zuvor theoretisch berechnet.

1. Übertragungsdauer: Es wird betrachtet, wie viel Zeit für die Vorbereitung der Datenübertragung benötigt wird, sowie die Zeit, die für ein vollständiges Update erforderlich ist.
2. Transfer-Geschwindigkeit: Es wird mehrfach ein Firmware-Update unter normalen Bedingungen übertragen und aus der Übertragungsdauer die Übertragungszeit per Fragment berechnet.
3. Transfer-Raten: Es wird mehrfach ein Firmware-Update unter normalen Bedingungen übertragen und dabei die Anzahl der empfangenen, benötigten und verlorenen Pakete gemessen. Aus diesen werden die folgenden Raten ermittelt
   - Paket-Verlust-Rate: $r_{pl} = \frac{\text{Anzahl verlorene Pakete}}{\text{Anzahl versendete Pakete}}$
   - Verlust-Ausgleichs-Rate: $r_{lc} = \frac{\text{Anzahl verlorene Pakete}}{\text{Anzahl verwendete Korrektur-Pakete}}$
   - Ausgleich-Nutz-Rate: $r_{ce} = \frac{\text{Anzahl verwendete Korrektur-Pakete}}{\text{Anzahl versendete Korrektur-Pakete}}$
   - Extra-Transfer-Rate: $r_{te} = \frac{\text{Anzahl Daten-Pakete}}{\text{Anzahl versendete Pakete}}$
4. Übertragungs-Parameter (theoretische Untersuchung): Es werden verschiedene Parameter der Übertragung variiert, um deren Einfluss auf die Übertragungsgeschwindigkeit zu berechnen. Die Parameter sind:
   - Spreading-Factor (Datenrate)
   - Fragment-Größe
   - Duty-Cycle
   - Ping-Slot-Periodizität (nur Class-B)

5.2.1 Kriterien der Bewertung

1. Funktionalität
   a) Verlässliche Übertragung eines Updates
   b) Unicast-Übertragungen
   c) Multicast-Übertragungen
   d) Integritäts- und Authentizitäts-Prüfung
2. Leistungsfähigkeit
   a) Optimale Übertragungsdauer unter Einhaltung des Duty-Cycles
   b) Minimaler Zusatzaufwand durch Fehlerkorrektur
   c) Unterstützung für leistungsschwache Mikrocontroller

5.3 LoRaWAN Application-Layer Spezifikationen

Mit vielen der in Unterunterabschnitt 5.1.1 aufgezählten Probleme hat sich bereits die ”FUOTA Working Group” der LoRa Alliance beschäftigt. Diese haben dazu drei Erweiterungen zu der LoRaWAN-Spezifikation vorgeschlagen, die einige dieser Probleme auf der Anwendungs-Ebene lösen. Im Folgenden werden diese im Einzelnen betrachtet und untersucht, welche konkreten Probleme sie lösen und bewertet, ob sie dafür jeweils geeignet sind. Die Spezifikationen sind dabei unabhängig von den Firmware-Updates einsetzbar, wurden aber mit dem Einsatz für die Übertragung als Hintergedanke entwickelt.

5.3.1 Application Layer Clock Synchronization

Das Ziel der ”Application Layer Clock Synchronization” Spezifikation ist es, die Uhr des Endgerätes mit der der Client-Anwendung mit Sekunden-Genauigkeit zu synchronisieren [34]. Das hier definierte Protokoll kann verwendet werden, um die Zeiten zu synchronisieren, falls keine besseren Quellen von genauer Zeit vorliegen. Eine Synchronisation der Zeit wird benötigt, um Multicast-Nachrichten empfangen zu können.

Die Synchronisation selbst erfolgt mithilfe der Zeitstempel von LoRaWAN-Paketen. Dazu sendet das Endgerät seine aktuelle GPS-Zeit an die Client-Anwendung. Diese berechnet dann aus den Daten und dem Zeitpunkt, zu dem diese von dem Gateway empfangen wurden, den Zeitunterschied. Dieser wird anschließend als Korrekturwert an das Endgerät zurückgesendet. Der Zeitunterschied zwischen dem Senden und dem Empfangen wird dabei nicht extra behandelt.

Der Algorithmus ist nicht besonders genau; da jedoch nur Sekunden-Genauigkeit als Ziel gesetzt ist, kann der Zeitverlust durch Verarbeitung und Datenübertragung vernachlässigt werden. Die resultierende Synchronisation ist keinesfalls genau genug, um konkrete Datenpakete zu synchronisieren, reicht aber aus, damit Geräte rechtzeitig einer Multicast-Gruppe beitreten. Das Protokoll ist somit für diesen Zweck geeignet und kann verwendet werden.

5.3.2 Remote Multicast Setup

Mit der ”Remote Multicast Setup” Spezifikation sollen Daten zur Erstellung, Verwaltung und Aktivierung von Multicast-Gruppen übertragen werden [37]. Dazu werden einige Nachrichten spezifiziert, welche diese Informationen enthalten, und entsprechend auf den Geräten ausgewertet und für die Aktivierung von Multicast-Übertragungen verwendet werden.

Das Protokoll verwaltet Multicast-Gruppen prinzipiell unabhängig von der gewählten Übertragungsart. Adresse, Schlüssel und weitere Parameter können einmalig an das Gerät übertragen und dann für mehrere Übertragungen zu verschiedenen Zeiten verwendet werden. Für die Aktivierung werden sowohl Class-B als auch Class-C Übertragungen unterstützt.

Das Protokoll stellt dabei fast alle Funktionen, die zu der Verwaltung von Multicast-Gruppen benötigt werden, bereit. Die einzige Ausnahme ist, dass Geräte nicht mitteilen können, ob sie Class-B oder Class-C unterstützten. Diese Information muss also entweder getrennt übertragen oder vorher festgelegt werden. Ansonsten ist das Protokoll geeignet und kann verwendet werden.

5.3.3 Fragmented Data Block Transport

Als letzte der drei übernimmt die ”Fragmented Data Block Transport” Spezifikation die Übertragung von großen Daten über LoRaWAN [35]. Dazu wird zum einen ein generelles Übertragungs-Protokoll definiert, das die Verwaltung und Datenübertragung der einzelnen Datenblöcke (hier ”Fragmente” genannt) übernimmt, sowie ein davon prinzipiell unabhängiger Algorithmus zur Codierung der Daten, um in der Lage sein zu können, den Verlust von Paketen auszugleichen.

Datenübertragungsprotokoll

Das Protokoll geht davon aus, dass der gewählte Fragmentierungs-Algorithmus die Daten in einzelne Fragmente mit festgelegter Größe aufteilt. Eine Übertragung wird zuvor angekündigt und dabei Informationen über das Datenpaket übertragen. Dazu gehören u. A. die Größe und Anzahl der Fragmente, sowie ein Padding. Aus diesen Daten kann das Endgerät die Größe der eigentlichen Datei berechnen.

Nach dem Setup können dann Daten übertragen werden. Der Empfang von Paketen wird nicht von dem Endgerät quittiert, was es möglich macht, diese Daten auch im Multicast zu übertragen. Dabei ist es erlaubt, mehr Fragmente als zuvor angekündigt zu versenden, um Fehlerkorrektur möglich zu machen. Während dem Datentransfer wird außerdem regelmäßig der Status der Geräte abgefragt, damit die Client-Anwendung berechnen kann, wie viele Pakete noch gesendet werden müssen.

Was jedoch von dem Protokoll aktuell noch nicht unterstützt wird, sind die Prüfung der Integrität oder die Authentifizierung der Daten. Diese müssen entweder von dem Fragmentierungs-Algorithmus oder der Anwendung selbst übernommen werden. Davon abgesehen eignet sich das Protokoll jedoch sehr gut für die Datenübertragung, solange ein Fragmentierungs-Algorithmus verwendet wird, der gut mit dem Verhalten des Protokolls zusammenspielt.

Vorgeschlagener Fragmentierungs-Algorithmus

Von der Spezifikation wird ebenfalls ein Algorithmus zur Fragmentierung vorgeschlagen [35]. Es handelt sich dabei um einen von LDPC-Codes abgeleiteten Code. Bei diesem werden zuerst die originalen Daten und danach eine Reihe von Korrektur-Paketen versendet, die mit den bereits empfangenen Paketen verrechnet werden können, um fehlende Pakete wiederherzustellen. Der Algorithmus wird im Detail in Unterabschnitt 2.5 erklärt.

Ein solcher Algorithmus ist gut geeignet für den Zweck der Übertragung von großen Daten via LoRaWAN. Dadurch, dass zuerst nur die eigentlichen Daten verschickt werden, müssen im Idealfall (kein Paketverlust) keine Extra-Bytes übertragen werden und durch die Verwendung entsteht kein Nachteil. Im Fall des Verlustes von Daten müssen Korrekturpakete gesendet werden; aber auch diese müssen nur so lang gesendet werden, bis alle Empfänger genug Informationen zur Wiederherstellung der fehlenden Daten erhalten haben. Korrekturpakete sind unabhängig von den verlorenen Paketen, können also ebenfalls im Multicast versendet werden.

Der einzige Nachteil des Protokolls sind die hohen Anforderungen an den Arbeitsspeicher. Dies liegt daran, dass der Empfänger eine Code-Matrix der Größe $m \times m$ (mit $m =$ Anzahl der Fragmente) in seinem Arbeitsspeicher vorhalten muss. Da LoRaWAN nur wenige Bytes pro Paket erlaubt, entstehen entsprechend viele Fragmente bei der Fragmentierung von Paketen, sodass auf Mikrocontrollern mit oftmals nur wenigen Kilobyte Arbeitsspeicher Probleme entstehen können. Bei einer beispielhaften Fragmentgröße von 49 B und 10 KB an Daten werden 205 Fragmente benötigt. Die zugehörige

Bit-Matrix würde entsprechend 5,26 KB an Arbeitsspeicher benötigen. Um dies auszugleichen, müssen entsprechend optimierte Datenstrukturen verwendet werden. Dazu mehr in Unterunterabschnitt 5.6.2.

5.4 Update-Protokoll

Unter der Voraussetzung, dass die zuvor genannten Spezifikationen verwendet werden, bleiben nur noch einige wenige Punkte übrig, die von einem Firmware-Update-Protokoll übernommen werden müssen. Diese sind:

1. Übertragung von Metadaten zu der Firmware. Dies sind der Geräte-Typ, die Version und Wichtigkeit des Updates, sowie die Prüfsumme der Firmware.
2. Authentifizierung des Updates. Oben genannte Daten müssen signiert werden, um zu verhindern, dass Angreifer falsche Updates verteilen können.
3. Integration mit den anderen Protokollen.
4. Aktivierung des Updates. Nach der erfolgreichen Übertragung muss das Gerät auf das OK des Servers warten, um das Update durchzuführen.

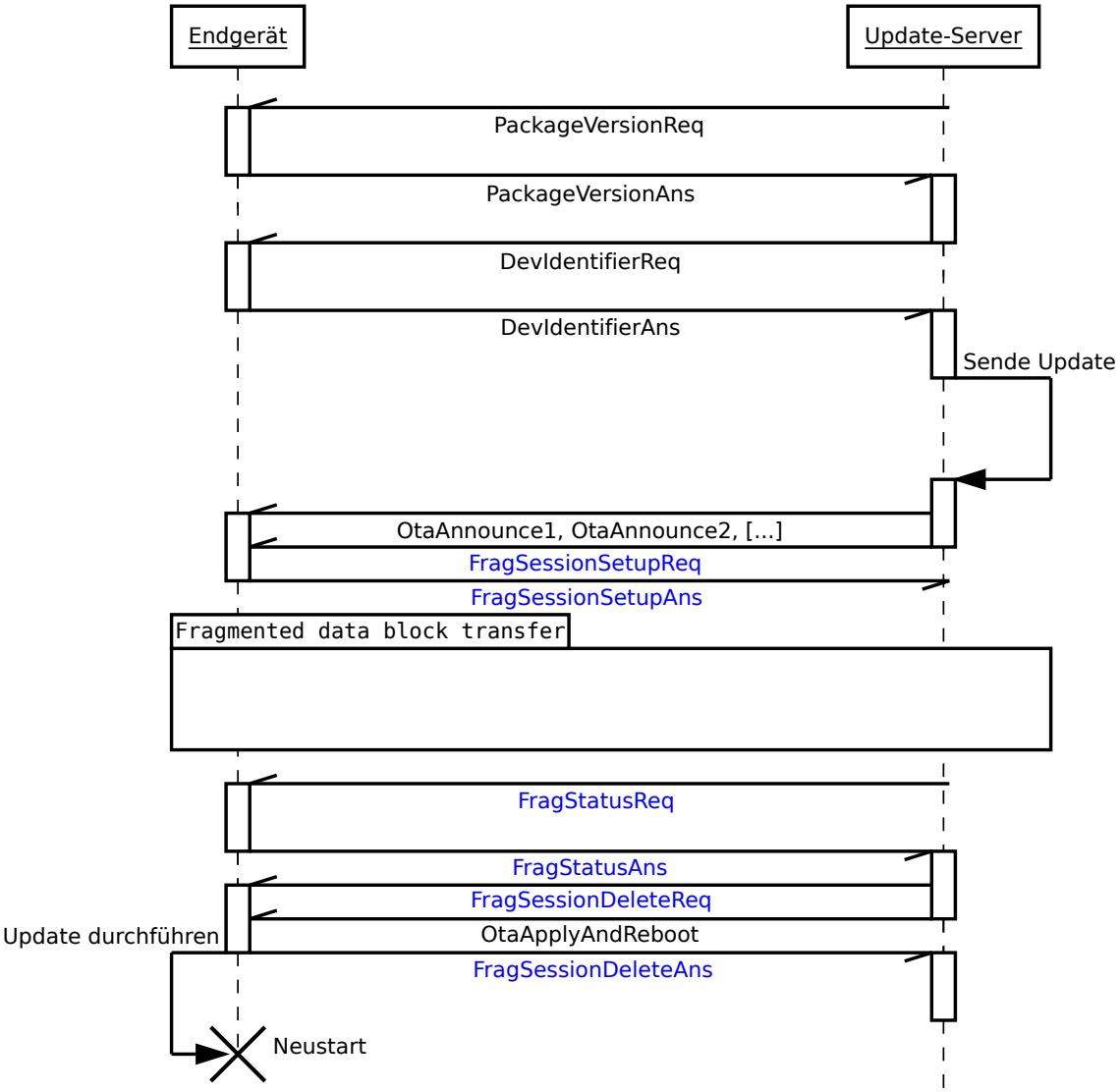

Abbildung 21: Sequenz-Diagramm Update-Protokoll

Aus diesen Anforderungen wurde das folgende Protokoll entwickelt. Abbildung 21 zeigt den Ablauf des Verfahrens. Dabei sind Nachrichten blau markiert, die Informationen des Update-Protokolls enthalten oder anderweitig für den Ablauf relevant sind, aber streng genommen nicht dazu gehören.

Das Design orientiert sich am Stil der anderen drei existierenden Protokolle. Es wird also ein Port für das Protokoll reserviert (Port 210) und alle Nachrichten in dem Format `<Code>[<Nutzdaten>][...]` versendet. Die Nutzdaten sind dabei optional und es können mehrere Nachrichten aneinandergereiht werden.

Das Protokoll beginnt mit der PackageVersionReq Nachricht (Code `0x00`, keine Payload) zum Abfragen der Version des Protokolls. Das Gerät muss dann mit der PackageVersionAns Nachricht (Code `0x00`, Tabelle 4) antworten. Dabei ist der PackageIdentifier auf `11` festgelegt und PackageVersion muss `1` sein.

Feld	PackageIdentifier	PackageVersion
Größe (Bytes)	1	1

Tabelle 4: PackageVersionAns Payload

Wenn das Gerät die korrekte Antwort gibt, wird als Nächstes vom Update-Server eine DevIdentifierReq Nachricht (Code `0x01`, keine Payload) versendet. Diese dient dazu, den Typ von Gerät zu ermitteln, um entsprechende Firmware-Updates zuordnen zu können. Das Gerät antwortet mit der Geräte-Kategorie und dem Geräte-Typen in Form einer DevIdentifierAns Nachricht (Code `0x01`, Tabelle 5).

Feld	DeviceInfo
Größe (Bytes)	1

Tabelle 5: DevIdentifierAns Payload

DeviceInfo Felder	DeviceType	DeviceCategory
Größe (Bits)	5	3

Tabelle 6: DevIdentifierAns DeviceInfo Feld

Neben diesem Setup müssen außerdem parallel dazu alle drei Erweiterungs-Protokolle initialisiert werden. Des Weiteren kann die Zeit-Synchronisation bereits im Hintergrund durchgeführt werden. Soll dann ein Firmware-Update veröffentlicht werden, müssen mehrere Schritte durchgeführt werden:

1. Eine Prüfsumme (Hash) des Updates muss erstellt werden. Aktuell wird eine Länge von 256 Bit vorausgesetzt, wie Beispielsweise SHA-256.
2. Ein 32 Bit Update-Deskriptor (Tabelle 7) wird aus der Version des Updates sowie einigen anderen Informationen erstellt.
3. Der Deskriptor und die Prüfsumme werden signiert. Aktuell wird eine Ed25519-Signatur vorausgesetzt (mehr dazu im nächsten Paragrafen). Die zugehörige Signatur hat eine Länge von 64 B. Der öffentliche Schlüssel muss auf dem Endgerät vorhanden sein. Die Signatur berechnet sich zu: `Signatur = Ed25519(PrivKey, Deskriptor|Prüfsumme)`

Deskriptor Feld	Magic	Version			Important
		Major	Minor	Patch	
Größe (Bits)	5	6	10	10	1

Tabelle 7: Update-Deskriptor

Der erstellte Deskriptor wird dann mit der FragSessionSetupReq Nachricht versendet, um den Transfer der Update-Datei zu starten. Davor werden noch die Prüfsumme und die Signatur mit den OtaAnnounce1 (Code `0x02`, Tabelle 8) und OtaAnnounce2 Nachrichten (Code `0x03`, Tabelle 9) an das Gerät verschickt. Die beiden Informationen wurden in zwei Nachrichten aufgeteilt, um nicht die 51 B Grenze zu überschreiten. Dabei wird die Signatur in zwei Sektionen aufgeteilt. Falls mehr Bytes versendet werden dürfen, können die beiden Nachrichten aneinandergereiht und in einem LoRaWAN-Paket verschickt werden. Im Falle einer Weiterentwicklung des Protokolls wurden weitere Codes reserviert, sodass zusätzliche OtaAnnounce Nachrichten verwendet werden können, falls mehr Daten versendet werden müssen.

Feld	Shasum	SignatureSection1
Größe (Bytes)	32	16

SignatureSection1 = Signature[0:15]

Tabelle 8: OtaAnnounce1 Payload

Feld	SignatureSection2
Größe (Bytes)	48

SignatureSection2 = Signature[16:63]

Tabelle 9: OtaAnnounce2 Payload

Das Gerät kombiniert dann den Deskriptor wieder mit der Prüfsumme und verifiziert die Signatur. Ist diese fehlerhaft oder wird das Update aus einem anderen Grund nicht akzeptiert, so wird in der FragSessionSetupAns die `errDescriptor`-Flag gesetzt. Andernfalls wird die normale Verarbeitung der Anfrage fortgesetzt.

Ab diesem Zeitpunkt wird die Update-Datei mit den anderen Protokollen übertragen. Ein Transfer via Multicast ist ebenfalls möglich, solange die entsprechende Sitzung zuvor erstellt wurde. Nach Abschluss der Übertragung prüft das Gerät die erhaltene Datei, indem die Prüfsumme berechnet und mit der zuvor erhaltenen verglichen wird. Stimmen sie überein kann das Gerät die erhaltene Firmware aktivieren, andernfalls wird diese verworfen.

Dieser Schritt erfolgt jedoch erst, nachdem das Gerät das OK vom Server in Form der OtaApplyAndReboot Nachricht (Code `0x04`, keine Payload) erhalten hat. Der Grund dafür ist, dass der Server zuerst vom Gerät die FragStatusAns Nachricht erhalten muss, um zu erkennen, dass dieses alle Daten empfangen hat. Dies ist notwendig, damit der Server die Geräte korrekt verwalten kann und weiß, ob noch welche Daten benötigen oder nicht. Erhält das Gerät die Nachricht, so wird die zuvor erhaltene Firmware zur aktiven gemacht und es startet neu, um diese zu aktivieren.

Wahl des Signatur-Algorithmus

Für das Signatur-Protokoll wurde der Ed25519 Algorithmus gewählt. Diese Auswahl wurde auf Basis der folgenden Kriterien getroffen.

1. Asymmetrische Signatur: Für Signaturen ist asymmetrische Kryptografie generell besser geeignet als symmetrische, da bei letzterer ein Schlüssel pro Endgerät verwendet werden müsste und mit dem Schlüssel auf dem Gerät selbst Signaturen erstellt werden könnten. Aus diesen Gründen sollte idealerweise ein asymmetrischer Algorithmus verwendet werden.
2. Standardisiert: Nur Algorithmen basierend auf öffentlichen Standards und ohne bekannte, praxisrelevante Sicherheitsprobleme stehen zur Auswahl.
3. Kurze Signatur-Länge: Die Signatur sollte so kurz wie möglich sein, um diese als Teil eines normalen LoRaWAN-Paketes verschicken zu können, ohne dabei die Sicherheit dieser aufzugeben.
4. Effizienter Algorithmus: Der Signatur-Algorithmus sollte möglichst effizient sein, besonders bei der Verwendung auf einem Mikrocontroller.
5. Existierende Implementierung: Idealerweise sollte ein Algorithmus gewählt werden, für den bereits eine (gute) Implementierung mit Unterstützung für Mikrocontroller existiert.

Durch diese Anforderungen können einige Kandidaten direkt ausgeschlossen werden. HMAC/CMAC Algorithmen basieren auf symmetrischen Schlüsseln, DSA basiert auf RSA, benötigt somit Signatur-Größen von minimal 2084 Bit, um sicher zu sein. Somit bleiben von den bekannteren Verfahren im Grunde nur auf elliptischen Kurven basierende übrig. Die Wahl des verwendeten Verfahrens hingegen erfolgte primär auf Basis der minimal möglichen Länge und der Verfügbarkeit von Implementierungen. Beide Kriterien werden von der Ed25519-Signatur erfüllt, weswegen die endgültige Wahl auf diese fiel.

Hierbei gilt es jedoch noch zu bemerken, dass diese Auswahl nicht zwingend erforderlich ist. Prinzipiell wird lediglich die Verwendung einer Signatur vorausgesetzt, die konkrete Wahl dieser kann bei Bedarf variiert werden.

5.5 Wahl der Basis-Bibliotheken

Um die in Unterabschnitt 5.3 und Unterabschnitt 5.4 aufgezählten Protokolle zu implementieren, werden einige Anforderungen an die Geräte und Basis-Bibliotheken gestellt. Die wichtigste ist dabei die Unterstützung von Endgeräten der Class-B oder Class-C.

Die bisher verwendete LMIC-Bibliothek für die Endgeräte kommt mit experimenteller Unterstützung für den Betrieb als Class-B-Gerät. Class-C wird nicht unterstützt. Des Weiteren erlaubt die Bibliothek keine parallelen Sitzungen, das Gerät muss also im Falle eines Multicasts temporär die Unicast-Übertragung deaktivieren.

Es wurde ebenfalls nach Alternativen für die LMIC-Bibliothek gesucht. Dabei konnten zwei Implementierungen gefunden werden, die zwar die verwendete Hardware nicht unterstützten, aber mit etwas Aufwand auf diese portiert werden könnten. Es handelt sich dabei um:

LoRaMac-node Die Referenz-Implementierung der LoRa-Alliance, verfügbar auf GitHub[19]. Unterstützt sowohl Class-B als auch Class-C, aber keines der verwendeten Boards. Ein Portierung ist möglich, aber, da CMake als Build-System verwendet wird, aufwendig.

[19] `https://github.com/Lora-net/LoRaMac-node`

LoRa Basic MAC Ein Fork der LMIC-Bibliothek, ebenfalls auf GitHub verfügbar[20]. Sie erweitert LMIC um vollständige Class-B und Class-C Unterstützung. Eine Portierung ist ebenfalls möglich; direkt unterstützt werden die verwendeten Boards jedoch nicht. Es wurde der Versuch unternommen, die Bibliothek zu portieren. Dieser scheiterte jedoch; Details dazu in Unterunterabschnitt 5.7.4.

Für den Anwendungsserver musste jedoch die Software gewechselt werden. Die bisher verwendete TTN-Cloud unterstützt weder Class-B noch Class-C Geräte. Als Alternative wurde LoRaServer gewählt. Es handelt sich dabei um eine Open-Source-Implementierung eines vollständigen LoRaWAN-Stacks, welche dementsprechend sowohl das Gateway als auch den Netzwerkserver enthält. Es werden Class-B und Class-C Geräte unterstützt und es gibt die Möglichkeit, Multicast-Gruppen zu erstellen [39].

5.6 Unicast Implementierung

Für die Unterstützung von Geräten im Unicast wurden zunächst nur die ”Fragmented Data Block Transport” Spezifikation sowie das Update-Protokoll implementiert. Im Folgenden werden der Verlauf sowie auftretende Probleme und deren Lösung aufgezählt.

5.6.1 Fragmented Data Block Transport Protokoll

Für dieses Protokoll musste zunächst ein Fragmentierungs-Algorithmus implementiert werden. Dabei wurde entschieden, den in der Spezifikation vorgeschlagenen Algorithmus zu verwenden. Sowohl die Codierung als auch die Decodierung werden in der Spezifikation vollständig beschrieben [35]. Was jedoch nicht vorgeschlagen wird, sind konkrete Datenstrukturen. Aus diesem Grund wurde in der initialen Implementierung eine einfache Bitmatrix gewählt; diese benötigt Arbeitsspeicher entsprechend Gleichung 10, mit $m = n =$ Anzahl Fragmente. Der Algorithmus selbst hingegen ist simpel umzusetzen.

$$RAM_B(m, n) = \left\lceil \frac{nm}{8} \right\rceil \tag{10}$$

Neben dem Algorithmus selbst wurde auch das Protokoll zum Transfer der Daten implementiert. Dazu wurde zunächst eine Basis-Bibliothek erstellt, welche die generelle Integration mit den verwendeten Bibliotheken übernimmt und allgemeine Funktionen und Interfaces bereitstellt, die von allen Implementierungen benötigt werden. Danach wurde dann das eigentliche Protokoll implementiert. Da dieses in der Spezifikation gut dokumentiert ist, gab es an dieser Stelle keine weiteren Komplikationen.

Während die Implementierung größtenteils generisch (und somit unabhängig von der konkret verwendeten Plattform) ist, wurden für den ESP32 und den Adafruit Feather M0 Backends realisiert, welche in der Lage sind, die erhaltenen Fragmente direkt aus dem Flash-Speicher zu lesen und dorthin zu schreiben. Für andere Plattformen existiert lediglich ein Interface, dass implementiert werden muss, um diese Funktionalität zu bieten.

5.6.2 Optimierte Bitmatrix

Wie bereits in Unterunterabschnitt 5.6.1 erwähnt, wurde zunächst eine einfache Bitmatrix verwendet. Im Test ergab sich jedoch, dass deren Anforderungen an den Arbeitsspeicher zu groß sind, um auf den eingesetzten Endgeräten verwendet werden zu können. Aus diesem Grund wurde eine speziell für den Anwendungsfall des LDPC-Codes entwickelte Datenstruktur vorgeschlagen, die je nach Situation bis zu 99,9% weniger Arbeitsspeicher

[20] `https://github.com/lorabasics/basicmac`

benötigt. Im Folgenden werden nur die Datenstruktur und deren Implementierung erläutert. Die Auswertung der Effizienz ist in Unterunterabschnitt 5.9.3 zu finden. Abbildung 22 zeigt dabei ein Beispiel für die Matrix-Struktur, die im Folgenden erläutert wird.

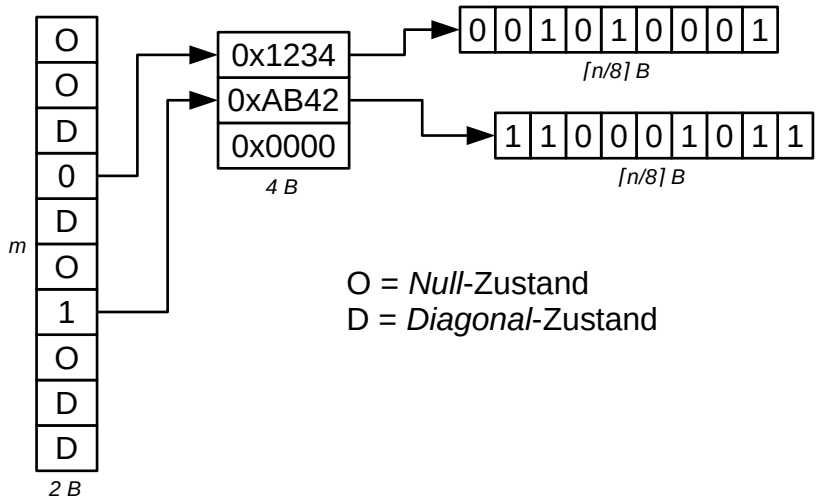

Abbildung 22: Veranschaulichung der optimierten Bitmatrix-Struktur

Die Matrix unterscheidet dabei zwischen drei Zuständen einer Zeile: `Null`, `Diagonal` und `Bitvector`. Der erste stellt eine Zeile dar, in der alle Bits null sind. Der zweite beschreibt eine Zeile, in der nur das diagonale Bit (also das Bit in Spalte n der Zeile n) eins ist. Der letzte schließt alle sonstigen Zustände ein. Diese Kategorisierung wurde verwendet, da die meisten Zeilen der Matrix nur entweder aus Nullen oder der diagonalen Eins bestehen. Andere Zustände nehmen die Zeilen nur im Fall von Paketverlusten ein.

Um dies ausnutzen zu können, verwendet die $m \times n$-Matrix zwei Listen. Die erste enthält genau m Zeilen, in der je 2-Byte-Indices gespeichert werden. Diese zeigen auf Positionen in der zweiten Liste, welche Adressen zu Bit-Vektoren enthält, die genau $\lceil \frac{n}{8} \rceil$ Bytes lang sind. Dabei werden jedoch zwei spezielle Indices festgelegt, die nicht auf ein Element in der zweiten Liste zeigen, sondern einen der anderen Zustände repräsentieren.

Enthält eine Zeile der ersten Liste den Wert $USHRT_MAX - 1$, dann steht dies für den `Null`-Zustand; es wird kein Bit-Vektor in der zweiten Liste für diese alloziert. Enthält eine Zeile in der ersten Liste den Wert $USHRT_MAX - 2$, dann steht dies für den `Diagonal` -Zustand. Es wird ebenfalls kein Bit-Vektor in der zweiten Liste für diese alloziert, da der Wert jeder Spalte aus der Zeilennummer berechnet werden kann. Für alle anderen Werte befindet sich die Zeile im `Bitvector`-Zustand; der Wert dieser ist ein Index in der zweiten Liste. In dieser wiederum befindet sich an der entsprechenden Stelle die Adresse zu einem dynamisch allozierten Bit-Vektor, der jede Spalte mit einem Bit repräsentiert.

5.6.3 Update-Protokoll

Für die Implementierung dieses Protokolls wurde die in Unterunterabschnitt 5.6.1 erwähnte Basis-Bibliothek verwendet. Des Weiteren wurde die Arduino Cryptography Library zum Berechnen und Überprüfen der SHA-256 Hashsumme der Firmware und zum Überprüfen der Ed25519-Signatur verwendet.

Die eigentliche Implementierung verlief problemlos; es ließen sich alle Funktionen wie gewünscht umsetzen. Auch die Integration mit den genutzten Bibliotheken auf verschie-

denen Plattformen konnte wie erwartet realisiert werden. Neben der Implementierung der generischen Hauptfunktionen der Bibliothek, wurden außerdem Backends für den ESP32 und den Adafruit Feather M0 erstellt. Diese sind in der Lage, Firmware-Partitionen zum Beschreiben auszuwählen und, nach erfolgreichem Download und Überprüfung des Updates, diese zu aktivieren. Für andere Plattformen existiert ein Interface, welches implementiert werden muss, um diese Funktionen zu bieten.

5.7 Multicast Implementierung

Um das zuvor realisierte Protokoll Mutlicast-fähig zu machen, musste es um mehrere Elemente erweitert werden. Zuerst musste das ”Clock Synchronization” Protokoll implementiert werden, um die Uhren der Geräte zu synchronisieren. Dieser Schritt ist optional, da auch andere Methoden (wie z. B. GPS) verwendet werden können, um eine genaue Zeit zu erhalten. Des Weiteren musste auch das ”Remote Multicast Setup” Protokoll implementiert werden, damit Multicast-Gruppen für den Transport erstellt werden können.

An dem Update-Protokoll selbst hingegen sind praktisch keinerlei Änderungen vonnöten. Es musste lediglich die Möglichkeit hinzugefügt werden, ein Multicast-Setup an den Mechanismus zu übergeben, sodass dieser an den internen Transport-Mechanismus weitergegeben werden kann. Alles Weitere wird bereits von den zuvor erwähnten Protokollen übernommen.

5.7.1 Clock Synchronization Protokoll

Die Synchronisierung basiert auf der GPS-Zeit. Es musste also eine GPS-Konforme Systemzeit implementiert werden. Dabei wurden die normalerweise bei GPS-Zeit ignorierten Schaltsekunden nicht beachtet. Bei Schaltsekunden handelt es sich um zusätzliche Sekunden, die in unregelmäßigen Abständen zwischen den Jahren eingefügt werden, um Abweichungen in der Erdrotation auszugleichen [58]. GPS-Zeit ignoriert diese Schaltsekunden, weswegen diese und die UTC-Zeit um mehrere Sekunden zueinander versetzt sind.

Dies bedeutet, dass die aktuelle Implementierung keine echte GPS-Zeit synchronisiert. Solange sowohl der Server als auch das Endgerät Schaltsekunden nicht beachten ist dies kein Problem, da nur die Zeit-Differenz zwischen den beiden Teilnehmern synchronisiert wird. Die fehlenden Sekunden können also bei Bedarf hinzuaddiert werden. Wenn diese Implementierung jedoch mit anderen Projekten kombiniert verwendet werden soll, muss beachtet werden, dass beide Seiten Schaltsekunden gleichermaßen behandeln.

Für die Synchronisation ist außerdem wichtig zu wissen, wann die Nachricht vom Gateway empfangen wurde. Die aktuelle Implementierung ist in der Lage, diese Information von TTN und LoRaServer zu extrahieren. Für andere Fälle wird die Zeit der Verarbeitung der Nachricht verwendet.

Aktuell basiert die Implementierung auf der Arduino Time Library, allerdings wird diese nur verwendet, um ein Interface zu implementieren, andere Zeit-Quellen können also ebenfalls verwendet werden.

5.7.2 Remote Multicast Setup Protokoll

Die Realisierung dieses Protokolls verlief ohne nennenswerte Probleme. Es musste jedoch, wie bereits in Unterabschnitt 5.5 erwähnt, die Gateway-Software gewechselt werden. Der Code stellt Interfaces bereit, um Class-B/C Sitzungen zu starten und zu beenden, implementiert wurden jedoch nur Class-B Sitzungen auf Basis der LMIC-Bibliothek.

Hier gilt es außerdem zu erwähnen, dass die LMIC-Bibliothek keine parallelen Sitzungen unterstützt. Es kann also immer nur entweder im Unicast oder im Multicast gesendet und empfangen werden. Entsprechende Sitzungsdaten müssen also zwischengespeichert werden, um den Wechsel vollziehen zu können. Aktuell unterstützt LMIC kein Class-C, weswegen dieser Teil des Interfaces nicht implementiert werden konnte.

Während die Implementierung des eigentlichen Protokolls keine Probleme bereitete, gab es gravierende Probleme mit der Realisierung der Kommunikation via Multicast. Dieses Problem ist jedoch unabhängig vom Remote Multicast Setup Protokoll und wird entsprechend hier nicht weiter erläutert. Mehr Details dazu finden sich in Unterunterabschnitt 5.9.2.

5.7.3 Anpassungen des Fragmented Data Block Transport Protokolls

Um mit den anderen beiden Protokollen zusammenarbeiten zu können, musste die serverseitige Implementierung des ”Fragmented Data Block Transport” Protokolls noch angepasst werden. Dabei wurde lediglich der Ablauf des Transfers erweitert, um das Setup der Multicast-Gruppe mit einzuschließen. An dem clientseitigen Code für das Endgerät mussten hingegen keinerlei Änderungen vorgenommen werden, da die Protokolle unabhängig voneinander arbeiten. Es musste lediglich sichergestellt werden, dass auch empfangene Multicast-Nachrichten verarbeitet werden.

5.7.4 Portierung der LoRa Basic MAC Bibliothek

Um in der Lage sein zu können, Class-C zu testen, und um eine vollständige Class-B Implementierung zur Verfügung zu haben, wurde versucht, die LoRa Basic MAC Bibliothek auf die verwendete Arduino-Plattform zu portieren. Es wurde dabei diese gewählt, da es sich um einen Fork der LMIC-Bibliothek handelt und diese somit als Basis für die Portierung verwendet werden kann.

Es wurde dazu die HAL-Ebene der Bibliothek implementiert. Es konnten dabei alle benötigten Funktionen bereitgestellt werden. Allerdings zeigte sich im Test, dass entweder die Portierung oder die Bibliothek selbst fehlerhaft sind. Grund dafür ist, dass zum einen das Testprogramm kurz nach dem Start abstürzt, zum anderen die gesendete JOIN-Request vom Netzwerkserver (ohne Fehler) verworfen wird, obwohl die Payload der Nachricht korrekt zu sein scheint. Es wurden an dieser Stelle aus Zeitgründen keine weiteren Versuche unternommen, diese Probleme zu beheben.

5.8 Updater-Client-Anwendung

Im ersten Schritt wurde jeweils der serverseitige Teil der Spezifikationen und des Update-Protokolls in Form von wiederverwendbaren Bibliotheken implementiert. Dies erlaubt die einfache Integration in eigene Anwendungen und vereinfacht das Testen der jeweiligen Implementierungen. Im nächsten Schritt wurde dann eine Kommandozeilen-Anwendung implementiert, welche diese Bibliotheken nutzt, um ein einfaches Tool zum Erstellen und Verteilen von Updates bereitzustellen.

5.8.1 Kommandozeile

In diesem Abschnitt wird der Aufbau und die Funktionen des Werkzeugs anhand der bereitgestellten Kommandozeilen-API erläutert. Dabei bietet das Tool 4 Grundfunktionen: `keygen`, `sign`, `server`, `publish`.

Die `keygen`-Funktion
Mit dieser Funktion können öffentliche und private Schlüssel für die Signatur und Validierung von Firmware-Images erstellt werden. Es handelt sich um Schlüssel des Ed25519-Verfahrens, welche im DER-Format vorliegen. Die zugehörigen Argumente sind:

--**passphrase** <**password**> Mit dieser Option kann das Passwort festgelegt werden, mit dem der private Schlüssel verschlüsselt werden soll oder wurde. Dazu wird mithilfe der PKCS #5 PBKDF-2 ein kryptografischer Schlüssel abgeleitet und mit diesem via AES-GCM die Verschlüsselung durchgeführt [27].

--**base64** Der Schlüssel wird im base64-Format, anstatt in einem binären Format, codiert.

--**raw** Für den öffentlichen Schlüssel werden nur die Schlüssel-Bytes gespeichert. Wenn diese Option fehlt, wird stattdessen die DER-Codierung verwendet.

private Erstellt einen privaten Schlüssel für das Signieren von Updates.

--**no-public** Erstellt keinen öffentlichen Schlüssel, nur den Privaten.

--**pub-path** <**path**> Gibt den Pfad zu dem zu erstellenden, öffentlichen Schlüssel an. Kann weggelassen werden, um diesen aus dem Pfad des privaten Schlüssels zu generieren.

<**path**> Der Pfad zu dem privaten Schlüssel, der erstellt werden soll. Falls die Datei bereits existiert, wird sie ersetzt.

public Erstellt einen öffentlichen Schlüssel aus einem existierenden privaten Schlüssel.

<**private-key-path**> Der Pfad zu dem privaten Schlüssel, aus dem der Öffentliche abgeleitet werden soll.

[<**public-key-path**>**]** Der Pfad zu dem öffentlichen Schlüssel, der generiert werden soll. Kann weggelassen werden, um den Pfad aus dem des privaten Schlüssels zu generieren.

Die `sign`-Funktion
Mit dieser Funktion können Metadaten zu einem Firmware-Image erstellt und mit einem zuvor erstellten privaten Schlüssel signiert werden. Dabei stehen folgende Optionen und Argumente zur Verfügung.

--**key-path** <**path**> Der Pfad zu einem privaten Schlüssel, mit dem die Metadaten signiert werden sollen. Wenn diese Option fehlt, werden stattdessen unsignierte Metadaten erstellt.

--**passphrase** <**password**> Mit dieser Option kann das Passwort angegeben werden, mit dem der private Schlüssel verschlüsselt wurde.

--**base64** Es wird angenommen, dass der Schlüssel im base64-Format, anstatt in einem binären Format codiert wurde.

--**meta** Gibt an, dass `image-path` den Pfad zu einer Metadaten-JSON-Datei, anstatt eines Firmware-Images, enthält.

<**image-path**> Der Pfad zu dem Firmware-Image (oder den Metadaten), zu denen (signierte) Metadaten erstellt werden sollen.

[<**signature-path**>**]** Der Pfad zu der Signatur- oder Metadaten-JSON-Datei, die erstellt werden soll. Kann weggelassen werden, um den Namen aus dem Pfad des Firmware-Images zu generieren.

Das Format der Metadaten ist in Listing 1 zu sehen. Dabei gibt `fwType` an, für welchen Gerätetyp das Update bestimmt ist, mit `magic` als Prüfwert für das Endgerät beim Empfang

des Updates. Mit dem `version`-Feld wird die Versions-Nummer des Updates angegeben und `important` spezifiziert, ob das Update wichtig ist oder nicht. In dem `sha256sum`-Feld wird die base64-codierte SHA-256 Hashsumme des Firmware-Images gespeichert und `signature` enthält die, ebenfalls base64-codierte, Signatur dieser Metadaten, wie sie vom Endgerät erwartet wird. Im Fall von unsignierten Metadaten ist dieses Feld stattdessen `null`.

```
{
    "fwType": {
        "category": 1,
        "type": 1
    },
    "magic": 11,
    "version": "1.1.0",
    "important": true,
    "sha256sum": "/xGu4BbqrFJgfwWTbIkFQcf1Dl5vMyIkkK+TH6kwdPo=",
    "signature": "vkmSB+pqlL6Vm47KH1PlXBADLxoSncjy2zqrDHQWux/YCK+6
       EmjkHw=="
}
```

Listing 1: Beispiel einer signierten Metadaten-JSON-Datei

Die `server`-Funktion
Mit dieser Funktion kann ein lokaler Server gestartet werden, der sich mit einem Netzwerkserver verbindet, um die Funktionen des Update-Protokolls bereitzustellen. Mit dem Start des Servers werden neu hinzugefügte Geräte bezüglich deren Protokoll-Versionen abgefragt. Falls nötig wird außerdem deren Zeit mit der Zeit des Systems, auf dem diese Anwendung ausgeführt wird, synchronisiert. Ein Update-Transfer wird noch nicht gestartet. Für diese Funktion stehen folgende Optionen und Argumente zur Verfügung.

--**url** <**url**> Die URL, unter der der Server lokal erreichbar sein soll. Es handelt sich dabei um eine Qt Remote Objects URL. Falls diese Option ausgelassen wird, wird eine einfache, lokale Instanz erstellt.

--**access-key** <**key**> Ein Password oder anderweitiger Schlüssel, der benötigt wird, um sich mit dem Netzwerkserver zu verbinden.

--**dev-db** <**path**> Pfad zu einer JSON-Datei, welche die von dem ”Remote Multicast Setup” Protokoll benötigten `GenAppKey`-Schlüssel pro Endgerät enthält.

--**jwt** <**jwt**> Ein JSON-Web-Token, wie es von LoRaServer benötigt wird, um die HTTP-Schnittstelle für den Versand von Multicast-Nachrichten nutzen zu können.

ttn|**loraserver** Der Typ von Netzwerkserver, der verwendet werden soll. Aktuell werden nur The Things Network (`ttn`) und LoRaServer (`loraserver`) unterstützt. Dabei ist Multicast-Transfer nur mit LoRaServer möglich.

<**app-id**> Die ID der LoRaWAN-Anwendung, für die Updates verteilt werden sollen. Das konkrete Format hängt von dem verwendeten Netzwerkserver ab.

Die `publish`-Funktion
Die letzte der Funktionen. Mit dieser können Firmware-Updates veröffentlicht werden. Dazu wird eine laufende `server`-Instanz benötigt, mit der sich das Werkzeug zum Transferieren des Updates verbinden kann. Aktuell muss diese auch auf dem gleichen Host-System und unter dem gleichen Benutzer ausgeführt werden. Um ein Update transferie-

ren zu können, wird ein signiertes Firmware-Image benötigt. Es wird sowohl Unicast- als auch Multicast-Transfer unterstützt.

--**server** <**url**> Die URL, unter der die lokal laufende Server-Instanz erreicht werden kann. Es handelt sich dabei um eine Qt Remote Objects URL. Falls diese Option ausgelassen wird, wird die gleiche URL angenommen, die auch der Server verwendetet, wenn dort die `--url`-Option ausgelassen wird.

--**meta** <**path**> Der Pfad zu den signierten Metadaten des Firmware-Images. Kann weggelassen werden, um diesen Pfad aus dem des Firmware-Images zu generieren.

unicast Startet einen Unicast-Transfer des Updates.

<**image-path**> Der Pfad zu dem Firmware-Image, welches an die Endgeräte transferiert werden soll.

multicast Startet einen Multicast-Transfer des Updates.

--**start-delay** <**delay**> Bestimmt die Wartezeit zwischen dem Start der erstellten Multicast-Gruppe und dem Versand der Nachricht, welche diese ankündigt. Falls die Option ausgelassen wird, wird eine Wartezeit von einer Stunde angesetzt.

<**image-path**> Der Pfad zu dem Firmware-Image, welches an die Endgeräte transferiert werden soll.

[<**rmc-config-path**>] Der Pfad zu einer JSON-Datei, welche die Konfiguration der Multicast-Gruppe enthält, über die die Daten transferiert werden sollen. Kann weggelassen werden, um diese Informationen stattdessen von der Standard-Eingabe abzufragen.

Falls das `rmc-config-path`-Argument angegeben wird, muss die JSON-Datei einem bestimmten Format folgen. Ein Beispiel dieses Formats ist in Listing 2 dargestellt. Dabei ist `devId` eine UUID, mit der die Multicast-Gruppe in LoRaServer angesprochen wird. `mcAddress` ist die LoRaWAN-Adresse der Gruppe und `key` der Master-Key, aus dem der `AppSKey` und `NetSKey` berechnet werden. Beide werden als hexadezimal codierte Strings angegeben. Diese drei Parameter sind dabei Pflicht. Mit `fCnt` kann optional ein Bereich von gültigen Downlink-Frame-Nummern angegeben werden. Wird dieser weggelassen, werden alle erlaubt.

```
{
    "devId": "31e077ec-16b6-4a3c-bb4a-f760561c2e4a",
    "mcAddress": "fc13b1ae",
    "key": "130863cb99d1d1496b232b24c27e4fbb",
    "fCnt": {
        "begin": 0,
        "end": 3846
    }
}
```

Listing 2: Beispiel einer RMC-Konfigurationsdatei

5.8.2 Server-Publish-Architektur

Die `server`-Funktion und die `publish`-Funktion arbeiten eng zusammen. Mit ersterer wird eine Server-Instanz gestartet, die die Updates verwaltet und an den Netzwerkserver transferiert, während letztere sich mit der Server-Instanz verbindet, um ein Update zu star-

ten und den Status zu verfolgen. Der Grund für diesen Aufbau ist, dass um ein Update überhaupt erst starten zu können, einige Informationen über die Endgeräte bekannt sein müssen. Diese sind unabhängig von einer einzelnen Update-Übertragung und werden typischerweise nur einmalig beim Beitritt eines Gerätes in das LoRaWAN-Netzwerk an den Server übertragen. Im Gegensatz dazu können mehrere Updates gleichzeitig und unabhängig voneinander an verschiedene Geräte-Gruppen transferiert werden.

Aus oben genannten Gründen wurde entschieden, diese beiden Aufgaben zu trennen. Die Server-Instanz wird einmalig gestartet und ist idealerweise permanent aktiv, ähnlich zu einem gewöhnlichen Anwendungsserver. Sie verwaltet im Hintergrund alle bekannten Geräte und sorgt dafür, dass deren Uhren synchronisiert bleiben. Gibt es ein neues Update, so wird eine temporäre Publish-Instanz gestartet, die die Server-Instanz nutzt, um das Update an alle relevanten Geräte zu übertragen.

5.9 Evaluation

Im Folgenden werden die Ergebnisse der Evaluation beschrieben. Dazu wird zuerst auf die Funktionstests der einzelnen Komponenten sowie des eigentlichen Update-Protokolls eingegangen. In diesen wird nur die Korrektheit der vorgesehenen Funktionen überprüft, aber keinerlei Aussagen bezüglich der Leistungsfähigkeit getroffen. Diese wird anschließend mit den Leistungstests untersucht und ausgewertet.

5.9.1 Funktionstest Clock Synchronization Protokoll

Um das Protokoll zu testen, wurde ein einfaches Testprogramm erstellt, welches jede Minute die lokale Zeit des Endgerätes an die Client-Anwendung sendet und parallel dazu die Zeit-Synchronisation durchführt. Die zugehörigen Tests verliefen erfolgreich und ohne Probleme. Um die Korrektheit der Synchronisation zu prüfen, wurde dieses Programm dreißig-mal ausgeführt und jeweils der Zeitunterschied nach der Synchronisation (unter Beachtung der Verarbeitungs-Zeit) zwischen dem Endgerät und der Client-Anwendung gemessen.

Mittelwert	Standardabw.	Median	Minimum	Maximum
0,296 s	0,219 s	0,247 s	0,020 s	0,761 s

Tabelle 10: Messergebnisse: Delta Zeit-Synchronisation

In Tabelle 10 sind die zugehörigen Ergebnisse dargestellt. Der Unterschied schwankt stark, liegt aber innerhalb des erlaubten Intervalls von $[0; 1)$ s. Ursachen für die Ungenauigkeiten sind die Verarbeitungs- und Sendezeiten, sowie die Tatsache, dass die intern verwendete GPS-Zeit nur Sekunden-Auflösung hat.

5.9.2 Funktionstest Remote Multicast Setup Protokoll

Auch hier wurde ein Testprogramm implementiert, dass zuerst eine Multicast-Sitzung erstellt und danach Daten über diese überträgt. Dabei funktioniert das Protokoll an sich wie erwartet; Sitzungen können erstellt und gestartet werden. Die eigentliche Datenübertragung im Multicast hingegen konnte jedoch für lange Zeit nicht erfolgreich durchgeführt werden. Die Gründe dafür sind zahlreich und werden im Folgenden erläutert. Es wurde dabei der ESP32 als Testgerät verwendet, da dieser verlässlicher als der Feather M0 arbeitet (weniger Paket-Verluste).

GPS-Simulation für Gateway-Beacons

Um Daten im Class-B Modus transferieren zu können, muss das Gateway den Endgeräten mitteilen, wann die Geräte Daten erwarten können. Dies geschieht mithilfe von Beacons, speziellen Datenpaketen, die regelmäßig als Broadcast gesendet werden (Siehe Abbildung 23). Als Voraussetzung für diese muss das Gateway jedoch über ein GPS-Modul verfügen, da die exakte GPS-Zeit als Basis für die Zeitberechnungen zum Senden der Beacons verwendet wird. Dabei unterstützt das Gateway jedoch nur ein einzelnes Modul, das U-Blox LEA-6[21], ein Modul, welches bereits nicht mehr hergestellt wird. Neuere Modelle sind vorhanden und laut Hersteller kompatibel, allerdings konnte nicht herausgefunden werden, ob jemand mit diesen bereits das Gateway erfolgreich mit GPS ausstatten konnte. Des Weiteren muss das Modul direkt auf das Gateway aufgelötet werden. Anstatt ein solches Modul einzubauen wurde entschieden, dieses stattdessen zu simulieren.

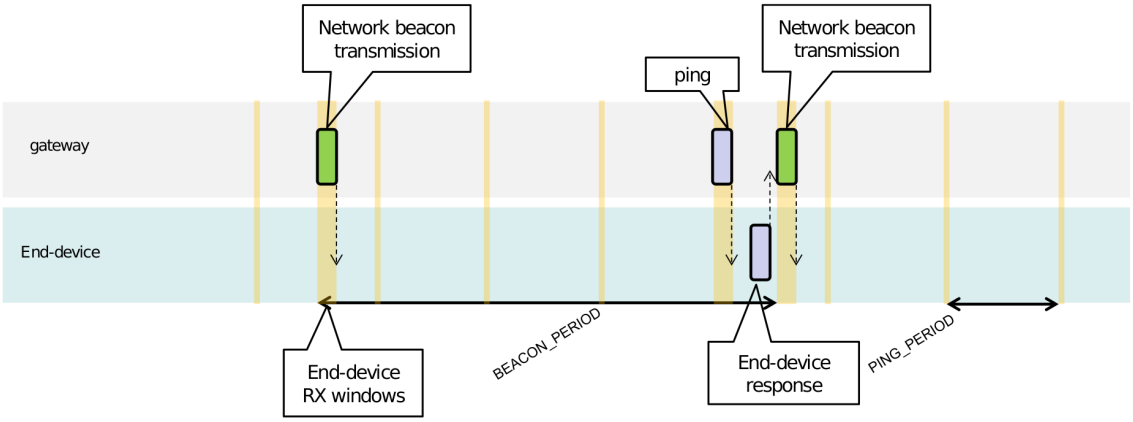

Abbildung 23: LoRaWAN Beacon und Ping-Slots [38]

Die Simulation erfolgte mit einem Python-Script, welches die Nachrichten, die das GPS-Modul sendet, vortäuscht, und somit beliebige Zeiten und Positionen versenden kann. Hier tat sich das erste größere Problem auf: Trotz der nachweisbar korrekten Verhaltensweise des Simulations-Scripts konnten vom Endgerät keine Beacons empfangen werden, obwohl die Log-Ausgaben des Programms eindeutig angaben, dass das Beacon an das LoRa-Board übertragen wurde. Es wurde ein Spektrums-Analysator zurate gezogen, um herauszufinden, an welcher Stelle das Problem liegt. Es zeigte sich, dass tatsächlich keine Beacons vom Gateway gesendet wurden.

Um dieses Problem zu lösen, wurde der Quellcode des Gateways untersucht. Dabei konnte festgestellt werden, dass das LoRa-Board ein PPS-Signal, welches normalerweise von dem GPS-Modul gesendet wird, erwartet, um das Beacon so exakt wie möglich senden zu können. Der Code wurde an dieser Stelle angepasst, um ohne das Signal auszukommen. Bei einem weiteren Test konnte nach dieser Änderung der korrekte Versand des Beacons verifiziert werden.

Fehlerhafte Bibliothek

Obwohl das Gateway nun Beacons sendete, konnten diese jedoch immer noch nicht von dem Endgerät empfangen werden. Da die Unterstützung für Class-B in LMIC nur experimentell ist, wurde auch hier die Implementierung gesichtet und mit dem LoRaWAN-Standard verglichen [38]. Dabei konnten drei Fehler gefunden und korrigiert werden, die den Empfang des Beacons und der Ping-Slots verhinderten.

Der erste Fehler war das Überschreiben des Beacon-Kanals. Dieser ist in LMIC einer festen Nummer zugewiesen, direkt nach den Kanälen, welche für die jeweiligen regio-

[21] https://www.u-blox.com/en/product/lea-6-series

nalen Parameter festgelegt sind. Als Teil des Beitritts in das LoRaWAN-Netz empfängt das Endgerät jedoch weitere, zusätzliche Kanäle vom Gateway. Diese überschreiben die Konfiguration für Beacons. Zur Lösung des Problems wurde der Beacon-Kanal mit einer höheren Nummer versehen und somit verhindert, dass dieser überschrieben wird.

Doch selbst nach dieser Änderung konnten noch keine Beacons empfangen werden. LoRaWAN-Daten-Nachrichten vom Gateway zu Endgeräten werden invertiert gesendet, sodass der Empfänger bereits beim Erhalt des Präambels zwischen Nachrichten von Endgeräten und von Gateways unterscheiden kann. Die einzige Ausnahme ist dabei das Beacon selbst. Dies wurde von LMIC nicht beachtet und es wurde versucht, das Beacon ebenfalls invertiert zu empfangen. Der Code wurde auch hier entsprechend angepasst und nun konnten Beacons korrekt verarbeitet werden.

Während der Empfang von Beacons funktionierte, konnten noch keine Daten-Nachrichten empfangen werden. Dabei war das Problem, dass das Gateway überhaupt nicht versuchte, Daten in den Ping-Slots zu versenden. Hier lag das Problem daran, dass das Endgerät eigentlich dem Gateway mitteilen muss, dass es in den Class-B-Modus gewechselt ist. Dies wurde von LMIC nicht gemacht und musste ebenfalls berichtigt werden. Außerdem war die Datenrate der Ping-Slots im Gateway falsch eingestellt und musste korrigiert werden. Dieser Fehler wurde erst spät entdeckt, da diese Einstellungen nicht dokumentiert sind und somit geraten werden musste, in welchem Format die Datenrate eingegeben werden muss.

Desynchronisierter Datentransfer

Nach all diesen Änderungen konnten Daten jedoch nur extrem unzuverlässig empfangen werden. Nur in ca. 1 von 20 Beacon-Perioden (Zeit zwischen zwei Beacons) konnten Daten empfangen werden. Außerdem war es dem Endgerät nicht möglich, weitere Beacons zu empfangen.

Für das zweite Problem konnte als Grund das Gateway ausgemacht werden. Bei der Untersuchung der Log-Datein konnte festgestellt werden, dass dieses die Beacons zu komplett falschen Zeiten (Unterschiede von >10s) sendete. Nach weiterer Analyse des Quellcodes konnte herausgefunden werden, dass das LoRa-Board das PPS-Signal ebenfalls für die interne Zeitverwaltung benötigt, weswegen die Umrechnung von der GPS-Zeit zu der internen Zeit fehlerhaft war. An dieser Stelle wurde entschieden, anstatt der Simulation ein echtes GPS-Modul zu verwenden, um ein gültiges PPS-Signal zu erhalten.

Abbildung 24: Das Drotek DP0106 USB-GPS-Modul

Dazu wurde das Drotek DP0106 USB-GPS-Modul[22], welches einen U-Blox NEO-M8N verbaut, verwendet (Siehe Abbildung 24). Bei diesem wurde außerdem der PPS-Pin herausgeführt, um diesen per GPIO-Pin an das LoRa-Board anschließen zu können. Es waren dabei einige minimale Anpassungen bezüglich der Konfiguration an dem Code zur Kommunikation mit dem Modul vonnöten, da das verwendete Modul ein neueres Modell ist, welches noch nicht vom Gateway unterstützt wird, und da es via USB angeschlossen wurde. Durch die Verwendung des Moduls ist das Endgerät nun in der Lage, Beacons über lange Zeiträume zu empfangen, wodurch dieses Problem gelöst wurde.

Damit blieb nur noch ein letztes Problem, welches immer noch bestand: Daten, welche in den Ping-Slots gesendet wurden, konnten nicht empfangen werden. Auch hier wurden die Log-Dateien zurate gezogen, um die Ursache des Problems zu finden. Es stellte sich heraus, dass die Daten zu anderen Zeiten gesendet werden, als sie empfangen werden. Dieser Unterschied wird in Abbildung 25 gezeigt. Das Gateway versendet die Daten, während das Endgerät regelmäßig auf Empfang schaltet, um diese zu empfangen. Der Abstand zwischen den einzelnen Ereignissen ist dabei für beide Seiten konstant; es liegt also eine Parallelverschiebung vor.

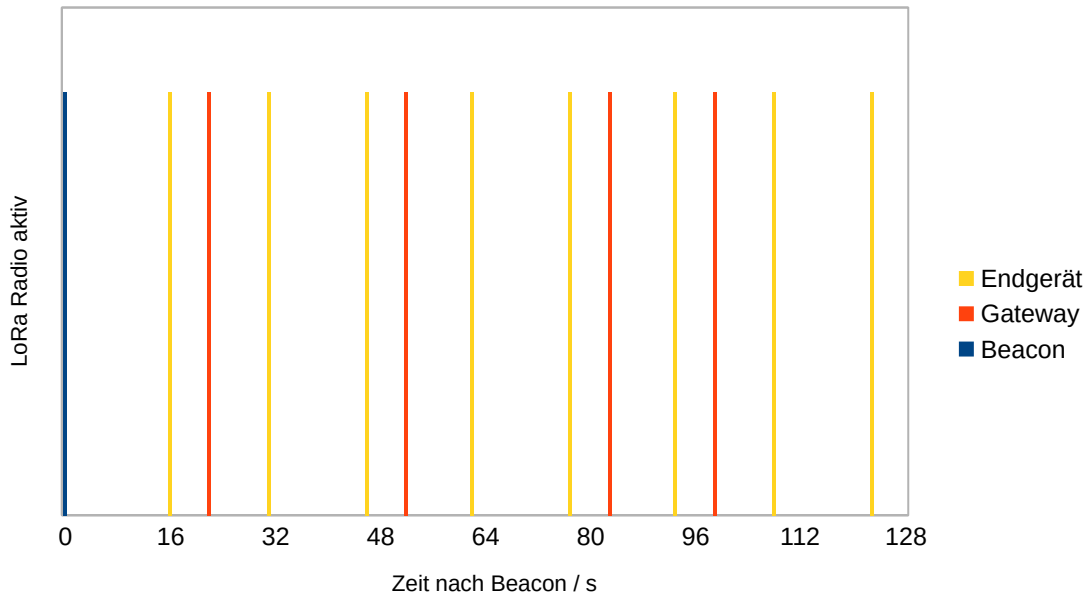

Abbildung 25: Versatz zwischen Versandt und Empfang von Ping-Slots

Es wurde auch hierfür die LoRaWAN-Spezifikation konsultiert und festgestellt, dass der Zeitpunkt für den jeweils ersten Ping-Slot nach einem Beacon mit einem Pseudo-Zufalls-Algorithmus berechnet wird. Bei einer Untersuchung des Codes von sowohl der LMIC-Bibliothek als auch dem Gateway, konnten jedoch keine Fehler oder Unterschiede in den zugehörigen Implementierungen gefunden werden.

[22] https://store.drotek.com/DP0106

Inkompatibles Beacon-Format
Es wurden also stattdessen die Parameter untersucht, die in die Berechnung dieses Offsets einfließen. Über mehrere Tests konnte dabei ein Parameter gefunden werden, der falsch zu sein schien: die Zeit, zu der das Beacon versandt wurde. Bei genauerer Untersuchung des Beacons konnte dann festgestellt werden, dass nicht nur diese, sondern auch einige andere Parameter ungewöhnliche Werte aufwiesen. Es wurden die Implementierungen des Gateways und der LMIC-Bibliothek verglichen und festgestellt, dass das Beacon anders generiert wird als es von der Bibliothek interpretiert wird. Nach erneutem Vergleich mit der Spezifikation konnte der Grund gefunden werden: Es gibt zwei verschiedene Beacon-Formate, wie in Tabelle 11 und Tabelle 12 dargestellt. Ersteres Format wird in der LoRaWAN-Spezifikation [38] definiert und von LMIC verwendet. Zweiteres ist hingegen Teil der regionalen Parameter [36] und wird vom Gateway versandt.

Feld	NetID	Time	CRC1	GwSpecific	CRC2
Größe (Bytes)	3	4	1	7	2

Tabelle 11: BCNPayload Feld der LoRaWAN-Spezifikation [38]

Feld	RFU	Time	CRC1	GwSpecific	CRC2
Größe (Bytes)	2	4	2	7	2

Tabelle 12: BCNPayload Feld der Regionalen Parameter [36]

Die Tatsache, dass in den beiden Formaten das ”Time”-Feld jeweils versetzt ist, führte dazu, dass eine falsche Zeit für die Berechnung verwendet wurde. Die LMIC-Bibliothek wurde angepasst, um stattdessen die Variante zu erwarten, die vom Gateway versendet wird. Damit konnte das Problem vollständig gelöst werden und es ist nun möglich, Daten im Class-B Modus zu empfangen.

Bis zu diesem Zeitpunkt wurde kein echter Multicast, sondern nur Unicast über Class-B in den Tests verwendet. Nachdem dieser funktionierte, wurde dann eine Multicast-Adresse erstellt und dieser Transfer ebenfalls erfolgreich getestet. Somit konnte nach langer Fehlersuche nun auch die Funktionalität des Multicast-Transfers erreicht werden.

Als letzter Test wurde dann noch der Multicast-Transfer auf dem Adafruit Feather M0 durchgeführt. Dieser war jedoch deutlich anfälliger für Fehler. Nur ca. 50% der versandten Daten konnten empfangen werden. Nach einer Analyse konnte festgestellt werden, dass dies nicht an dem Protokoll, sondern an der Ungenauigkeit der internen Uhr liegt. Um das Problem zu lösen, kann entweder ein externer Zeitgeber verwendet, oder versucht werden, früher und dafür länger auf Empfang zu schalten. Es wurde an dieser Stelle keine der Varianten getestet, da entschieden wurde, stattdessen den ESP32 für Multicast zu verwenden.

5.9.3 Funktionstest Fragmented Data Block Transport Protokoll

In diesem Abschnitt wird auf die Ergebnisse des Funktionstests des ”Fragmented Data Block Transport” Protokolls eingegangen. Dazu wird zuerst die effektive Einsparung an Arbeitsspeicher durch die optimierte Bitmatrix-Struktur untersucht. Anschließend wird der Fragmentierungs-Algorithmus selbst untersucht und abschließend das komplette Protokoll, unabhängig von dem verwendeten Algorithmus.

Optimierte Bitmatrix

Aus den Angaben in Unterunterabschnitt 5.6.2 können die Anforderungen der Datenstruktur an den Arbeitsspeicher ermittelt werden. Diese können mit Gleichung 11 berechnet werden. Dabei sind m und n die Anzahl der Zeilen und Spalten einer $m \times n$-Matrix. x ist die prozentuale Anzahl von Zeilen, die sich im `Bitvector`-Zustand befinden. W_{ptr} ist die Größe (in Bytes) von Adressen auf dem Zielsystem. Für den ESP32 wäre dies beispielsweise $W_{ptr} = 4$. Des Weiteren kann für die folgenden Auswertungen von $n = m$ ausgegangen werden, da der Fragmentierungs-Algorithmus nur mit quadratischen Matrizen arbeitet.

$$RAM_L(m,n,x) = 2W_{ptr} + 2m + xm\left(W_{ptr} + \left\lceil \frac{n}{8} \right\rceil\right); x \in [0;1] \tag{11}$$

Um die Wirksamkeit dieser optimierten Datenstruktur prüfen zu können, wurde der Arbeitsspeicher-Verbrauch für verschiedene m und x berechnet und mit den Anforderungen einer klassischen Bitmatrix (Siehe Gleichung 10) verglichen. Die Ergebnisse sind in Abbildung 26 dargestellt. Die effektive Einsparung wurde nach Gleichung 12 berechnet.

$$EEP(m,x) = 1 - \frac{RAM_L(m,m,x)}{RAM_B(m,m)}; x \in [0;1] \tag{12}$$

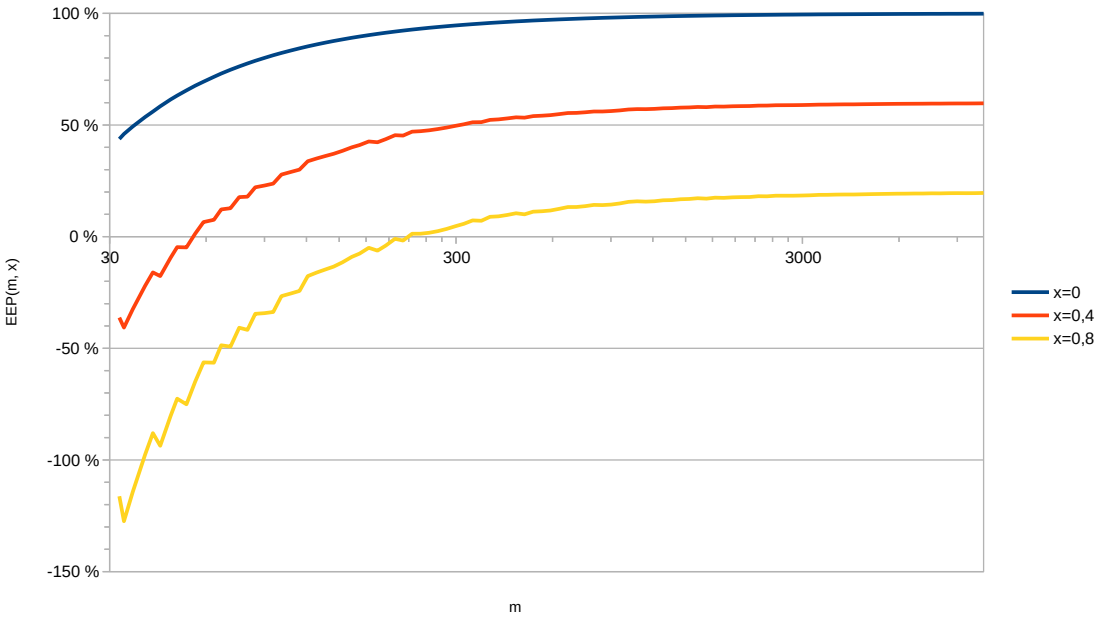

Abbildung 26: Einsparung an Arbeitsspeicher, abhängig von m und x

Aus Abbildung 26 lassen sich mehrere Eigenschaften der optimierten Matrix im Vergleich zu der Bitmatrix ausmachen:

1. Die Einsparung ist logarithmisch abhängig von m
2. Die Einsparung ist linear abhängig von x
3. Es gilt $\lim_{m->\infty} EPP(m,x) = 1 - x$
4. Für kleine m und große x finden sich Schwankungen in der Einsparung. Dies liegt an dem Vorhandensein des $\left\lceil \frac{nm}{8} \right\rceil$ Teils in RAM_B und des $\left\lceil \frac{n}{8} \right\rceil$ Teils in RAM_L.
5. Für kleine m und große x ist die Einsparung der optimierten Matrix negativ. Das maximale x, in Abhängigkeit von m, für das sich der Einsatz dieser Struktur noch lohnt,

berechnet sich nach Gleichung 13. Dabei wurde bei der Berechnung der Formel die normalerweise vorhandene Aufrundung vernachlässigt, weswegen die Formel nur einen Richtwert berechnet.

$$\begin{aligned} RAM_L(m,m,x) &\leq RAM_B(m,m) \\ 2W_{ptr} + 2m + xm\left(W_{ptr} + \frac{m}{8}\right) - \frac{m^2}{8} &\leq 0 \\ \frac{16W_{ptr} + 16m + 8W_{ptr}xm + xm^2}{8} - \frac{m^2}{8} &\leq 0 \\ (m^2 + 8W_{ptr}m)x - m^2 + 16m + 16W_{ptr} &\leq 0 \\ x &\leq \frac{m^2 - 16m - 16W_{ptr}}{m^2 + 8W_{ptr}m} \end{aligned} \tag{13}$$

Fragmentierungs-Algorithmus
Um den Algorithmus zu testen, wurde ein Programm entwickelt, welches zufällige Daten erzeugt, diese mit dem Algorithmus codiert, decodiert und dann mit den Original-Daten vergleicht. Dabei wurden verschiedene Parameter variiert, um sicherzustellen, dass der Algorithmus korrekt arbeitet. Diese Parameter sind:

- Fragmentgröße (Byte): $\{250; 122; 58; 26\}$
- Coderate: $\{0,5; 0,25\}$
- Datenlänge (Byte): $\{5000; 25000; 50000; 100000\}$
- Fehlerrate (% verlorene Pakete): $\{0; 33,3; 66,6; 100\} \times (1 -$ <Coderate>$)$

Im Test arbeitet der Algorithmus komplett zuverlässig und kann selbst bei der maximal zulässigen Fehlerrate immer noch die Original-Daten rekonstruieren.

Transport-Protokoll
Als letzter Teil wurde dann das Protokoll selbst getestet. Dazu wurde ein Testprogramm erstellt, welches zuerst eine Transport-Sitzung erstellt und dann Daten mithilfe des zuvor getesteten Algorithmus an das Endgerät überträgt. Dabei wurde die CRC32-Prüfsumme der Daten als Deskriptor verwendet, um die Integrität der empfangenen Daten prüfen zu können.

Die Übertragung wurde zuerst im Unicast durchgeführt, um das Protokoll unabhängig von den anderen testen zu können. Es wurde dabei einmal ohne Übertragungsfehler und einmal mit einer Fehlerrate von 20% gesendet, um sicherzustellen, dass das Protokoll für beide Fälle korrekt abläuft. Die Fehlerrate wurde dabei willkürlich gewählt, da nur das Verhalten des Protokolls, nicht das des Algorithmus, untersucht wurde. Die Tests verliefen erfolgreich; es konnten in allen Varianten die übertragenen Daten fehlerfrei wiederhergestellt werden.

Nachdem alle drei Protokolle unabhängig voneinander erfolgreich getestet werden konnten, wurde anschließend noch ein kombinierter Test durchgeführt, bei dem der fragmentierte Transfer von Daten über einen Multicast-Kanal geprüft wurde. Auch hier konnte die korrekte Funktion für beide Fehlerraten verifiziert werden.

5.9.4 Funktionstests Update-Protokoll

Nachdem die Funktionalität aller Basis-Protokolle verifiziert werden konnte, wurde anschließend der Update-Mechanismus selbst getestet. Dazu wurde ein Testprogramm ge-

schrieben, welches ein zuvor generiertes Firmware-Image signiert und anschließend mithilfe des Protokolls an die Endgeräte überträgt.

Unicast

Das Protokoll wurde zuerst nur im Unicast-Modus (Class-A) getestet. Das Endgerät sendet dazu wieder in regelmäßigen Abständen die interne Zeit, damit das Gateway die Möglichkeit zum Antworten bekommt. Dabei ist wichtig, dass das Gateway das `FPending`-Bit setzt, damit das Endgerät weiterhin leere Nachrichten an dieses sendet, auf das das Gateway mit den eigentlichen Datenpaketen antworten kann. Andernfalls würde die Übertragung des Updates via Class-A extrem lange dauern.

Mit diesen Voraussetzungen wurde dann ein Firmware-Update für den Adafruit Feather M0 versendet. Es wurde dieser Mikrocontroller gewählt, weil eine minimale Firmware nur eine Größe von 11 KB hat, während eine äquivalente Firmware für den ESP32 auf 192,2 KB kommt. Es müssen somit bei einer Fragmentgröße von 96 B mindestens 115 Fragmente versendet werden, um das Update zu übertragen. Um die Signatur-Prüfung in den Test miteinbeziehen zu können, wurde ein Testschlüssel generiert und von dem Testprogramm verwendet, um die Firmware zu signieren.

Das Testprogramm wurde dabei zehnmal durchgeführt und jedes Mal konnte die Firmware erfolgreich aktualisiert werden. Dabei ging in 8 von 10 Fällen mindestens 1 Fragment verloren. Die Korrektur-Funktion konnte somit auch erfolgreich getestet werden.

Multicast

Abschließend wurde das Update im Multicast getestet. Es wurde dabei der ESP32 als Testgerät eingesetzt, da wie in Unterunterabschnitt 5.9.2 erwähnt Übertragungen via Multicast auf dem Feather nicht zuverlässig sind. Der Transport wurde so konfiguriert, dass ein Class-B Multicast Kanal erstellt und anschließend für den Transfer verwendet wird. Dabei bleibt das Programm während der Übertragung von Nachrichten des Update-Protokolls im Unicast, weswegen sich bezüglich dieses Protokolls nichts im Testablauf änderte.

Auch dieser Test konnte erfolgreich durchgeführt werden; die Updates konnten im Multicast an mehrere Endgerät übertragen werden. Die Firmware des ESP32 ist dabei 192,2 KB groß und benötigt somit 1922 Fragmente bei einer Fragment-Größe von 100 B, der maximal zulässigen Fragment-Größe für Class-B. Aufgrund des dadurch entstandenen höheren Zeitaufwands beim Transfer wurde dieses Update nur dreimal getestet.

5.9.5 Funktionstest Kommandozeilen-Updater

Nachdem die korrekte Funktionsweise des Update-Protokolls mit den Testanwendungen verifiziert werden konnte, wurde abschließend noch die korrekte Funktion des Werkzeugs zum Verteilen von Updates sichergestellt. Dazu wurde die Testanwendung des Update-Protokolls auf dem Endgerät ausgeführt und auf dem Testrechner ein Update mit dem Werkzeug signiert und anschließend übertragen.

Der Transfer verlief dabei ohne Probleme und konnte auf beiden Seiten erfolgreich nachvollzogen werden. Das Werkzeug funktioniert also ebenfalls wie erwartet und kann in der Praxis eingesetzt werden, um Updates an Endgeräte zu verteilen.

5.9.6 Leistungstests Update-Protokoll

Im Anschluss an die Funktionstests wurde die Leistungsfähigkeit des Protokolls untersucht. Die Ergebnisse dieser Untersuchungen werden im Folgenden präsentiert. Dabei

wurden zuerst einige theoretische Überlegungen bezüglich der Übertragungsdauer getroffen und anschließend praktische Messungen zum Vergleich durchgeführt.

Theoretische Auswertungen
Für die theoretische Auswertung wurde berechnet, wie lange die Übertragung eines Firmware Updates in der Theorie dauert, abhängig von verschiedenen Parametern. Als Erstes galt es dafür zu ermitteln, welche Parameter zu beachten sind. Dazu wurden die LoRaWAN Spezifikation [38] und die regionalen Parameter [36] zurate gezogen und folgende Parameter ermittelt:

- Relative Frequenzbelegungsdauer (Duty-Cycle, DC)
- Datenrate (Spreading-Faktor, SF)
- Coderate (CR)
- Bandbreite (BW)
- Fragment-Größe (l_f)
- Update-Größe (l_u)
- Ping-Slot-Rate (nur für Class-B, r_p)

Aus diesen Parametern lässt sich Gleichung 14 ableiten, welche die Übertragungsdauer in Abhängigkeit von den angegebenen Parametern ermittelt. Die dargestellten Formeln berechnen dabei die Zeit, die der Transfer einer Update-Datei benötigt. Die erste Formel berechnet den Transfer unabhängig von dem gewählten Modus. Die zweite Formel bezieht die Ping-Slot-Rate für Class-B Übertragung mit in die Berechnung ein. Dabei wurde in beiden Fällen von der optimalen Übertragungsdauer ausgegangen, also dem Fall, bei dem kein Fragment bei der Übertragung verloren geht. Ein solcher Fall kann äquivalent zu einer Übertragung mit einer Fragment-Anzahl gleich der insgesamt versendeten Fragmente betrachtet werden.

$$\begin{aligned} T_{update} &= T_{packet}(l_{lw} + l_f) \left\lceil \frac{l_u}{l_f} \right\rceil \frac{1}{DC} \\ T_B &= 2^{r_p} \left\lceil \frac{l_u}{l_f} \right\rceil \\ T_{update_B} &= max(T_{update}, T_B) \end{aligned} \tag{14}$$

Der wichtigste Teil ist dabei die T_{packet} Funktion, welche die Zeit, die ein LoRa-Paket zum Versand braucht, berechnet. Die zugehörigen Formeln sind in Gleichung 15 dargestellt und wurden dem "Semtech LoRaModem Designers Guide" entnommen [53]. Dabei ist $n_{preamble} = 8$, wie in den regionalen Parametern für LoRaWAN angegeben [36].

$$\begin{aligned} T_{sym} &= \frac{2^{SF}}{BW} \\ T_{preamble} &= (n_{preamble} + 4,25) \times T_{sym} \\ payloadSymbNb(PL) &= 8 + max\left(\left\lceil \frac{8PL - 4SF + 28 + 16}{4SF} \right\rceil (CR + 4), 0 \right) \\ T_{payload}(PL) &= payloadSymbNb(PL) \times T_{sym} \\ T_{packet}(PL) &= T_{preamble} + T_{payload}(PL) \end{aligned} \tag{15}$$

Es wird dabei der l_{lw} Parameter verwendet, welcher angibt, wie viele zusätzliche Bytes für ein LoRaWAN-Paket benötigt werden. Dabei wurden der Aufbau und die Längen der einzelnen Elemente der LoRaWAN-Spezifikation entnommen [38]. Es wird von dem optimalen Fall ausgegangen, dass $FOptsLen = 0$ B ist und somit bei der Berechnung der $FOpts$ Teil entfällt. Des Weiteren wurde $FRMPayload = 2$ B festgelegt, da das "Fragmented Data Block Transport" Protokoll für jedes Fragment 2 B an Metadaten mit versendet. Die Zusammenstellung des Pakets wird in Gleichung 16 dargestellt und dessen Länge berechnet sich zu $l_{lw} = len(PHYPayload) = 15$ B.

$$\begin{aligned} PHYPayload &= MHDR + MACPayload + MIC \\ MACPayload &= FHDR + FPort + FRMPayload \\ FHDR &= DevAddr + FCtrl + FCnt + FOpts \end{aligned} \tag{16}$$

Mit dieser Formel kann nun die benötigte Zeit für verschiedene Szenarien berechnet werden. Dabei wurden im Folgenden für einige der Parameter Werte festgelegt, da diese in der Praxis nur in seltenen Fällen variiert werden. Es wurde $CR = 1$ festgelegt, um eine Coderate von $4/5$ zu erhalten. Des Weiteren sind SF, BW und DC von dem gewählten Kanal und Modus abhängig und somit nicht frei wählbar. Die möglichen Kombinationen sind in Tabelle 13 aufgelistet. Dabei sind nur Kanal 1–3 von der Spezifikation festgelegt. Alle weiteren Kanäle können vom Gateway dynamisch zugewiesen werden, müssen sich aber an die Richtlinien bezüglich der Spezifikation und der relativen Frequenzbelegungsdauer halten.

Kanal	Frequenz	Bandbreite	Datenraten	Duty-Cycle
Class-A, Erstes Empfangs-Fenster				
1	868.10 MHz	125 kHz	SF12-SF7	1%
2	868.30 MHz	125 kHz	SF12-SF7	1%
3	868.50 MHz	125 kHz	SF12-SF7	1%
4-8	863.0 – 868.0 MHz	125 kHz	SF12-SF7	1%
4-8	868.0 – 868.6 MHz	125 kHz	SF12-SF7	1%
4-8	868.7 – 869.2 MHz	125 kHz	SF12-SF7	0,1%
4-8	869.4 – 869.65 MHz	125 kHz	SF12-SF7	10%
4-8	869.7 – 870.0 MHz	125 kHz	SF12-SF7	1%
Class-A, Zweites Empfangs-Fenster				
-/-	869.525 MHz	125 kHz	SF12	10%
Class-B				
-/-	869.525 MHz	125 kHz	SF9	10%
Class-C				
-/-	869.525 MHz	125 kHz	SF12	10%

Tabelle 13: LoRaWAN-Kanäle für die EU [38][36]

Mit dieser Basis wurden im Folgenden einige beispielhafte Übertragungen berechnet. Dazu wurden verschiedene Werte für l_f sowie unterschiedliche Übertragungs-Modi und -Kanäle gewählt und die jeweilige Transfer-Dauer berechnet. Es wurde dafür $l_u = 10960$ B festgelegt, da l_u sich nur linear auf die Übertragung auswirkt (erhöht die Anzahl der zu

übertragenden Fragmente), während die anderen Parameter komplexere Auswirkungen haben.

Im ersten Schritt wurden $DC = 1\%$, $BW = 125$ kHz festgelegt und $SF \in [7, 12]$ variiert, um die zugehörige Update-Zeit zu erhalten. Dies wurde einmal mit $l_f = 36$ B, der maximal möglichen Fragmentgröße bei $SF = 12$ und einmal mit $l_f = l_{f_{max}}(SF)$, der maximal möglichen Fragmentgröße, in Abhängigkeit vom gewählten Spreading-Faktor, berechnet. Das Ergebnis wird in Abbildung 27 dargestellt.

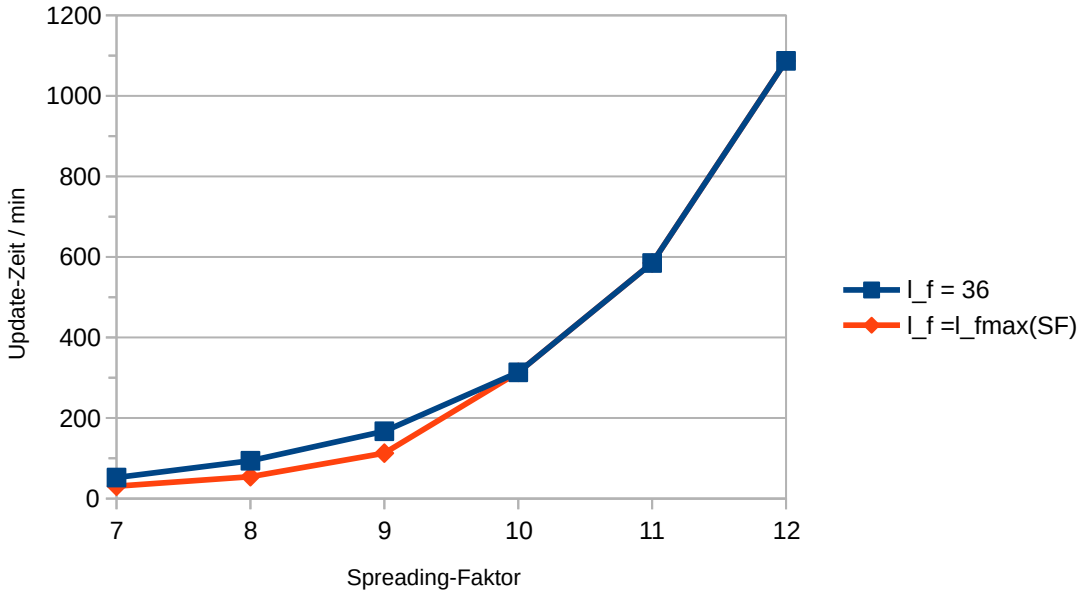

Abbildung 27: Update-Zeit in Abhängigkeit vom Spreading-Faktor SF

In einem zweiten Schritt wurden dann $DC = 1\%$, $BW = 125$ kHz und $SF = 7$ festgelegt und $l_f \in [36; 207]$ B variiert. Dieses Intervall berechnet sich aus der maximal erlaubten Fragmentgröße für $SF = 12$ und $SF = 7$. Größere Werte sind nicht möglich, kleine Werte sinnlos. Abbildung 28 zeigt die zugehörigen Graphen.

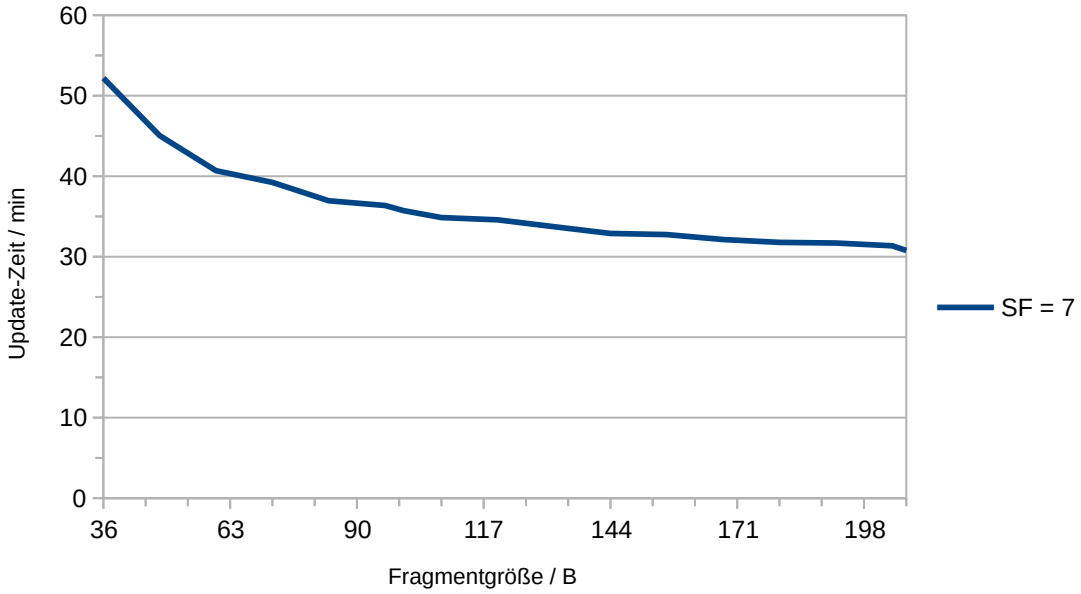

Abbildung 28: Update-Zeit in Abhängigkeit von der Fragmentgröße l_f

Der letzte Parameter, der noch variiert wurde, war der Duty-Cycle. Dieser wurde auf die für die EU möglichen Werte beschränkt, also $DC \in \{0,1\%; 1\%; 10\%\}$. Es besteht dabei eine Antiproportionalität zwischen dem Duty-Cycle und der Update-Zeit, $DC = 0,1\%$ braucht also zehnmal so lange wie $DC = 1\%$ etc. Der Grund dafür ist, dass dieser Wert nichts an der Übertragung an sich verändert, sondern nur die Pausen zwischen zwei Datenpaketen beeinflusst. In Tabelle 14 wurden beispielsweise die für T_{update} berechneten Zeiten in Abhängigkeit von DC mit $BW = 125$ kHz, $SF = 7$ und $l_f = 96$ B aufgetragen. Der $DC = 100\%$ Wert dient dabei dem Vergleich, um zu sehen, wie lange das Update bei permanentem Senden brauchen würde.

Duty-Cycle DC	100%	10%	1%	0,1%
Update-Zeit T_{update}	21,82 s	3,64 min	36,36 min	6,06 h

Tabelle 14: Update-Zeit in Abhängigkeit vom Duty-Cycle

Anschließend wurde noch untersucht, welchen Einfluss die Ping-Slot-Periodizität bei einem Class-B-Transfer auf die Übertragungsdauer hat. Dabei konnte festgestellt werden, dass bereits ab einer Periode von 3 (8 Sekunden), selbst bei der für Class-B maximalen Fragmentgröße von $l_f = 100$ B, Daten seltener gesendet werden als vom Duty-Cycle erlaubt. Die Werte sind zum Vergleich in Tabelle 15 dargestellt. Ab dieser Periode können also alle verfügbaren Ping-Slots für die Übertragung genutzt werden. Bei geringeren Perioden müssten einige ausgelassen werden, um die entsprechenden Richtwerte einzuhalten. Dabei handelt es sich bei T_{update_B} um den theoretischen Idealwert. Durch die Variation des Offsets des ersten Ping-Slots, in Kombination mit den sich alle 128 s wiederholenden Beacons, ist die Zeit zwischen dem letzten Ping-Slot vor und dem ersten Ping-Slot nach einem Beacon oftmals größer als die Periode, wodurch Updates länger brauchen.

<table>
<tr><th colspan="2">Ping-Slot-Periodizität</th><th colspan="3">Update-Zeit</th></tr>
<tr><th>Faktor</th><th>Sekunden</th><th>T_{update}</th><th>T_B</th><th>T_{update_B}</th></tr>
<tr><td>0</td><td>1 s</td><td rowspan="8">11,28 min</td><td>1,83 min</td><td rowspan="3">T_{update}</td></tr>
<tr><td>1</td><td>2 s</td><td>3,67 min</td></tr>
<tr><td>2</td><td>4 s</td><td>7,33 min</td></tr>
<tr><td>3</td><td>8 s</td><td>14,67 min</td><td rowspan="5">T_B</td></tr>
<tr><td>4</td><td>16 s</td><td>29,33 min</td></tr>
<tr><td>5</td><td>32 s</td><td>58,67 min</td></tr>
<tr><td>6</td><td>64 s</td><td>117,33 min</td></tr>
<tr><td>7</td><td>128 s</td><td>234,67 min</td></tr>
</table>

Tabelle 15: Vergleich Update-Zeit Duty-Cycle mit Class-B Ping-Slot-Periodizität

Die letzte Abhängigkeit, die untersucht wurde, war die zwischen der Größe eines Updates und der Zeit, welche für dieses benötigt wird. Abbildung 29 zeigt diese Abhängigkeit für verschiedene Parameter. Festgelegt wurden $BW = 125$ kHz, $SF = 9$ und $l_f = 100$ B. Aufgetragen wurde zum einen die Update-Zeit für die verschiedenen Duty-Cycle DC, ohne Beachtung der Ping-Slot-Periode. Für diese wurden zusätzlich die verschiedenen Perioden für $r_p \geq 3$ aufgetragen, jeweils mit einem festgelegten $DC = 10\%$. Kleinere

Werte für r_p wurden nicht dargestellt, da diese bereits durch den Duty-Cycle begrenzt werden und somit alle identisch zum $DC = 10\%$-Fall sind.

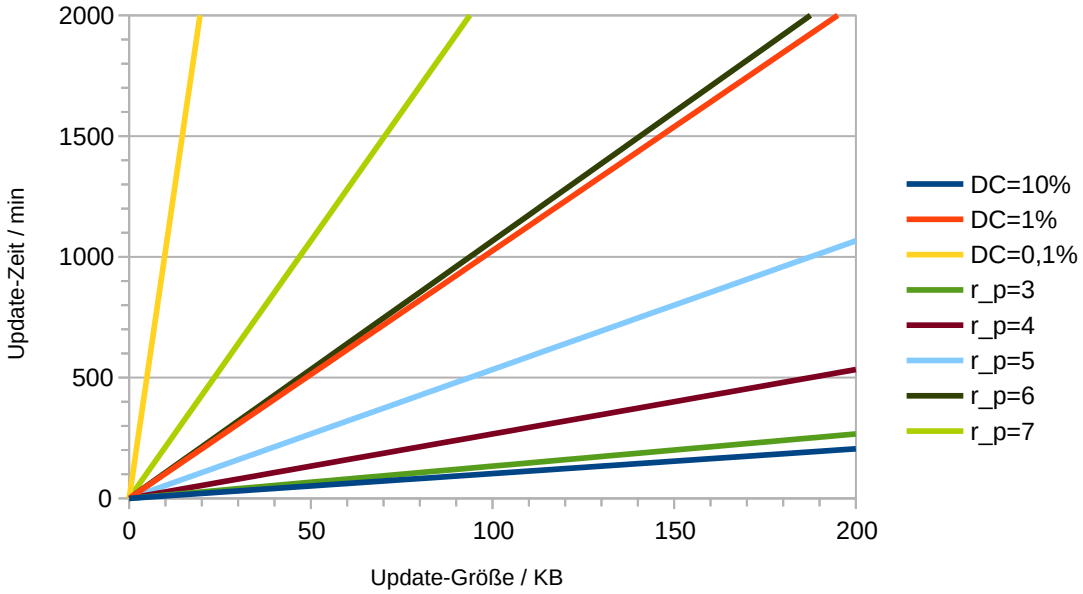

Abbildung 29: Update-Zeit in Abhängigkeit von der Update-Größe l_u

Auswertung der Unicast-Messergebnisse

Neben den theoretischen Auswertungen wurden auch praktische Untersuchungen durchgeführt, um diese mit den theoretischen Ergebnissen vergleichen zu können. Dazu wurde das Firmware-Update zehnmal unter gleichen Bedingungen ausgeführt und die komplette Übertragung aufgezeichnet. Es wurden dabei $CR = 1$ (4/5), $SF = 7$, $BW = 125$ kHz, $l_f = 96$ B und $l_u = 10960$ B als Übertragungs-Parameter festgelegt. Die Übertragungen wurden ohne Beachtung des Duty-Cycles durchgeführt.

Um vergleichbare Werte zu erhalten, wurden bei der Durchführung zuerst alle Fragmente an den Netzwerkserver übertragen, sodass das Endgerät ohne Pause Daten empfangen kann. Die Fragmente wurden im Unicast via Class-A Modus übertragen da hier die echte Transfergeschwindigkeit gemessen werden kann. Das Endgerät fragt also wiederholt mit leeren Datenpaketen beim Gateway nach, um Daten zu empfangen, bis alle Fragmente abgearbeitet wurden.

Dabei wurden mehrere Werte ermittelt, die im Folgenden dargestellt werden. Der erste, interessante Wert ist dabei die Übertragungsdauer. Diese wird dabei in zwei Bereiche aufgeteilt. Der erste ist Setup-Zeit t_s, also die Zeit, die das Protokoll benötigt, um die eigentliche Datenübertragung vorzubereiten. Dazu gehören die Initialisierungen der Protokolle sowie die Ankündigung und der Start des Datentransfers. Der zweite Teil ist die Transfer-Zeit t_t, also die Zeit, die das Protokoll für die eigentliche Datenübertragung, einschließlich der Status-Übertragungen, benötigt. Die statistische Auswertung dieser Zeiten wird in Abbildung 30 dargestellt.

Neben diesen Zeiten wurde parallel auch die Zeit des Transfers eines einzelnen Pakets (t_F) untersucht, sowie der Zeitverlust durch verlorene Pakete (t_l). Diese legen einen Minimalwert fest; eine schnellere Übertragung ist nicht möglich. Abbildung 31 stellt diese dar. Dabei ist t_F in Sekunden und t_l in Minuten aufgetragen.

Die letzte Gruppe von Werten, die betrachtet wurde, beschäftigt sich mit verschiedenen Raten. Dazu wurden die Paket-Verlust-Rate r_{pl}, die Verlust-Ausgleichs-Rate r_{lc}, die

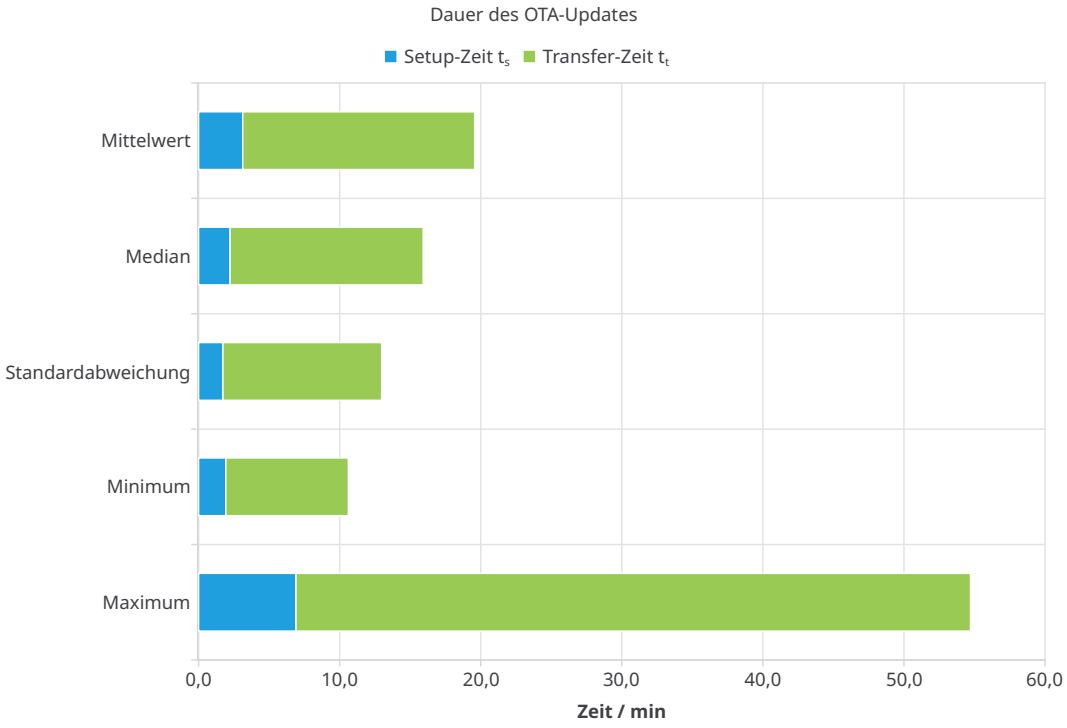

Abbildung 30: Auswertung der Messergebnisse der Update-Zeit

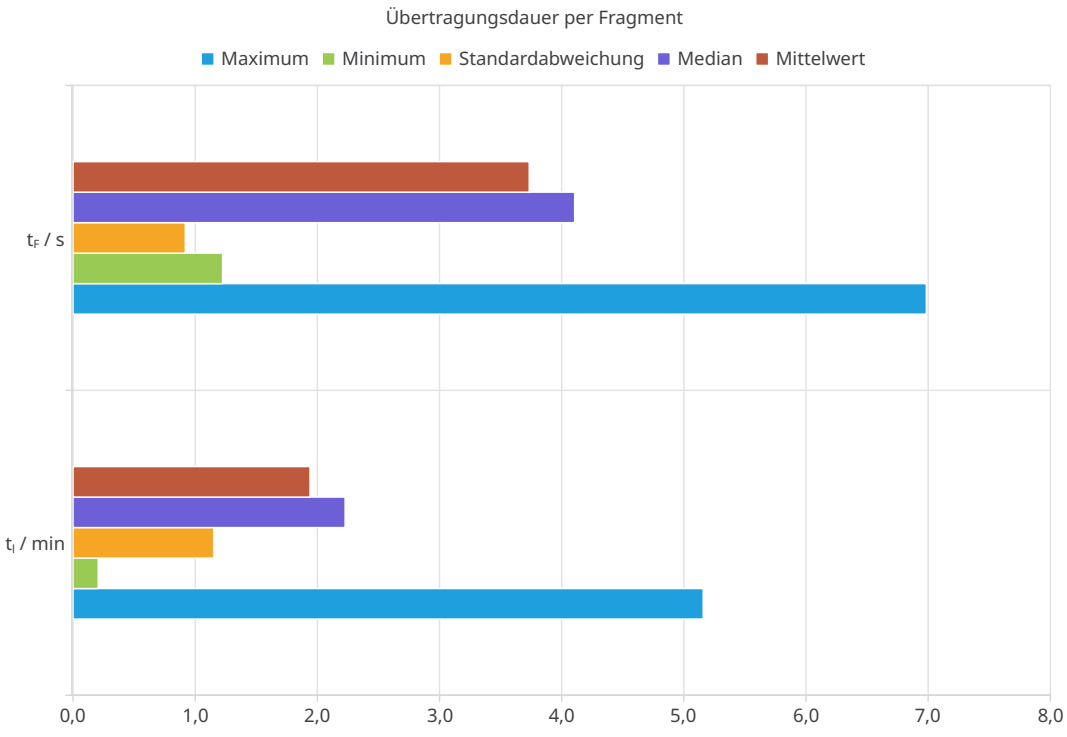

Abbildung 31: Auswertung der Messergebnisse der Transferzeit-Zeit pro Fragment

Ausgleich-Nutz-Rate l_{ce} und die Extra-Transfer-Rate r_{te} aus den Messdaten berechnet und in Abbildung 32 abgebildet. Die Bedeutung dieser Raten wird in Unterabschnitt 5.2 erläutert.

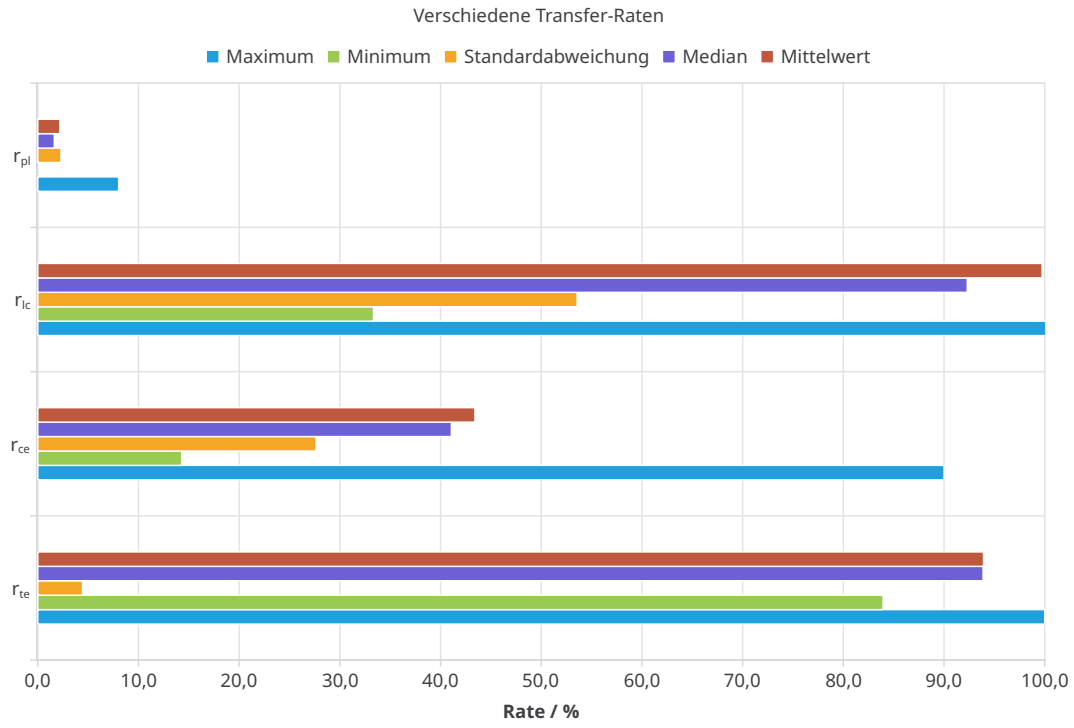

Abbildung 32: Auswertung der Messergebnisse der verschiedenen Raten

Auswertung der Multicast-Messergebnisse
Der Multicast-Test wurde mit einer Ping-Slot-Rate von $r_p = 3$, also einer Periode von 8 Sekunden, durchgeführt. Des Weiteren wurde eine Wartezeit von 6 Minuten zwischen der Ankündigung und dem Start des Multicast-Transfers gewählt, damit alle teilnehmenden Geräte rechtzeitig den Multicast aktivieren können. Da der Transfer der Daten via Class-B erfolgt, sind alle anderen Parameter festgelegt auf $CR = 1$ (4/5), $SF = 9$ und $BW = 125$ kHz. Es wurde mit der maximalen Fragmentgröße $l_f = 100$ B ein Update der Größe $l_u = 192192$ B übertragen. Der Duty-Cycle liegt bei $DC = 10\%$, da jedoch $r_p = 3$ gewählt wurde, wird dieser nicht komplett ausgenutzt und muss nicht weiter beachtet werden.

Da es sich um eine Multicast-Übertragung handelt, muss das Endgerät nicht permanent Pakete senden, um selbst welche empfangen zu können. Stattdessen muss es rechtzeitig beginnen, das Beacon zu verfolgen, um die Ping-Slots öffnen zu können. Auch hier wurden wieder alle Fragmente direkt an den Netzwerkserver übertragen, sodass die maximale Übertragungsgeschwindigkeit gemessen werden konnte.

Die Übertragung selbst benötigte ca. 4,5 h pro Update, einschließlich der Wartezeiten für die Multicast-Übertragungen. Dabei wurde im Schnitt alle 7,99 s ein Fragment übertragen, passend zu der Ping-Slot Periode. Es gingen von den vom Gateway versandten Paketen ca. 1% verloren und mussten korrigiert werden; eine ähnliche Menge von Fragmenten wurde zur Korrektur benötigt. Die zugehörigen Diagramme können im Anhang gefunden werden.

5.10 Ergebnisse

In diesem Abschnitt werden die Ergebnisse der Evaluation diskutiert. Dazu soll zuerst das Update-Protokoll im Ganzen bewertet werden. Danach wird der theoretisch berechnete Zeitaufwand mit den praktisch gemessenen Werten verglichen. Anschließend werden einige mögliche Verbesserungen für das Protokoll vorgeschlagen und diskutiert.

5.10.1 Bewertung auf Basis der Untersuchungs-Kriterien

Zuerst soll an dieser Stelle geprüft werden, ob alle Bewertungskriterien aus Unterunterabschnitt 5.2.1 erfüllt wurden. Dabei können besonders die funktionalen Kriterien schnell bestätigt werden. Updates können erfolgreich über eine verlustbehaftete LoRa-Datenverbindung übertragen werden, sowohl im Unicast als auch im Multicast. Dabei führt das Protokoll eine Integritätsprüfung der Firmware durch und verifiziert die Authentizität der Daten mithilfe von signierten Metadaten.

Während schlussendlich alle funktionalen Kriterien erfüllt werden konnten, muss an dieser Stelle hervorgehoben werden, dass es einige aufwendige Hürden auf dem Weg dorthin zu überwinden galt. Die wohl signifikantesten Probleme waren dabei zum einen die Implementierung einer Speicher-Effizienten Bit-Matrix für den Fragmentierungs-Algorithmus, zum anderen der Weg zur korrekten Class-B Übertragung, welche für Multicast-Updates benötigt wird.

Bei der Entwicklung der Bitmatrix war dabei die Herausforderung, einen effizienten Algorithmus zu finden. Wie bereits in Unterunterabschnitt 5.6.2 erwähnt, wurden dazu zuerst einige existierende Varianten betrachtet und letztendlich entschieden, einen eigenen Algorithmus zu entwickeln. Dieser ist besonders effizient für den Fall, dass kaum Fehler bei der Übertragung entstehen. In den praktischen Untersuchungen konnten Fehlerraten von ca. 2% ermittelt werden, womit diese Voraussetzung gegeben ist. Auch in den praktischen Tests konnten keine Fehler aufgrund von unzureichendem Arbeitsspeicher ausgemacht werden. Besonders für große Updates lohnt sich also der Einsatz dieser Struktur.

Das größte Problem bei der Implementierung war jedoch der in Unterunterabschnitt 5.9.2 beschriebene, langwierige Prozess, Class-B Übertragungen funktionsfähig zu bekommen. Die primären Gründe für dessen Schwierigkeit lagen in der Tatsache, dass Class-B und auch Class-C bisher nur selten eingesetzt werden. Aus diesem Grund gibt es nur wenig Dokumentation bezüglich dieser und viele Implementierungen unterstützten die Modi nur experimentell. Dieses Problem sorgt wiederum für Fehler in den Implementierungen und Inkompatibilitäten zwischen den verschiedenen Varianten. Auch hier wird die Lösung durch schlechte Dokumentation erschwert. Was an dieser Stelle jedoch hervorgehoben werden muss, ist, dass die Spezifikation von LoRaWAN hingegen gut strukturiert und dokumentiert ist, sodass durch den Vergleich von Implementierung und dem Standard Fehler oftmals eindeutig identifiziert und behoben werden konnten.

Tatsache ist jedoch, dass dieses Problem unabhängig von dem Update-Protokoll existiert und behoben wurde, da das Verfahren unabhängig von der verwendeten MAC-Implementierung entwickelt wurde. Das Update-Protokoll an sich hingegen entspricht den Anforderungen und erfüllt die gestellten Kriterien.

5.10.2 Vergleich zwischen theoretischen und praktischen Werten

Um die Leistungsfähigkeit des Protokolls bewerten zu können, werden im Folgenden die praktisch ermittelten Messwerte mit den theoretischen Berechnungen verglichen. Im Fokus steht dabei die Übertragungsdauer. Dies ist der wichtigste metrische Wert, da für diese Zeit der Transfer jeglicher anderweitiger Daten, neben dem Update, blockiert wird.

Theoretische Ergebnisse

Betrachtet man zuerst nur die theoretischen Untersuchungen, so fällt als Erstes auf, dass der Spreading-Faktor der signifikanteste Parameter ist. Bei konstanter Fragment-Größe wirkt sich dieser exponentiell auf die Übertragungsdauer aus. Das erste Ziel bei OTA-Updates sollte also immer sein, einen möglichst niedrigen Spreading-Faktor zu erreichen, selbst wenn dafür ggf. eine höhere Paket-Verlust-Rate in Kauf genommen werden muss. Die Länge der Fragmente hingegen hat eine deutlich geringere Signifikanz. In den meisten Fällen ist es jedoch sinnvoll, die für den Spreading-Faktor maximal zulässige Fragmentlänge zu wählen, da diese indirekt beeinflusst, wie viel Arbeitsspeicher der Fragmentierungs-Algorithmus benötigt.

Praktische Ergebnisse

Das Erste, was dabei festgestellt werden kann, ist, dass das Setup eines Transfers nicht vernachlässigt werden sollte. Dies sind im Schnitt 3 Minuten pro Gerät, in denen Daten zur Vorbereitung des Transfers benötigt werden. Dabei kann bei einem Multicast-Transfer die Wartezeit für den Start von Mutlicast-Gruppen vernachlässigt werden, da in dieser Zeit ganz normal Daten übertragen werden können; das Gerät wird nicht blockiert.

Werden dann Daten transferiert, so werden pro Fragment im Schnitt 3,75 s im Unicast benötigt. Der größte Teil ist davon eine kurze Pause seitens des Endgerätes zwischen dem Empfang von zwei Paketen, um den Kanal nicht permanent zu blockieren. Die dabei ermittelten Raten zeigen, dass der verwendete Fragmentierungs-Algorithmus effizient arbeitet. Bei einer Paket-Verlust-Rate von ca. 2% werden von den meisten Empfängern genauso viele Zusatz-Pakete benötigt, wie verloren wurden. Ca. 43% der versandten Korrekturpakete pro Welle werden tatsächlich von den Endgeräten verwendet, was zu durchschnittlich 6% Extra-Paketen führt. Was also an dieser Stelle optimiert werden könnte, ist die Anzahl der Korrekturpakete pro Welle.

Für den Multicast zeigen sich ähnliche Ergebnisse. Die Zeit zwischen zwei Fragmenten liegt nun bei 8 s, aber alle anderen Werte sind ähnlich zu denen des Unicasts. Die Verwendung einer Multicast-Übertragung bringt also keine Vor- oder Nachteile bezüglich der Qualität der Übertragung. Es bleibt jedoch der Vorteil, dass Updates an mehrere Geräte parallel übertragen werden können. Falls möglich, ist also ein Multicast-Transfer immer zu bevorzugen.

Vergleich

Für das 11 KB große Testupdate wurde im praktischen Test eine Gesamtdauer von durchschnittlich 20 min (einschließlich des Setups) erzielt. Vorgegeben wäre für den zugehörigen Kanal ein Duty-Cycle von 1%. Den theoretischen Berechnungen zufolge muss ein solches Update über einen Zeitraum von 36 min übertragen werden. Es ist also möglich, selbst im Unicast die verfügbare Sendezeit komplett auszuschöpfen.

Im Multicast hingegen stimmen theoretische und praktische Werte fast komplett überein, da in beiden Fällen 8 s pro Fragment benötigt werden. Insgesamt dauert das Update dabei ca. 12 min länger als in der Theorie, also genau die Zeit, welche als Wartezeit für die Multicast-Gruppen festgelegt wurde. Diese muss zweimal verrechnet werden, da durch den Verlust von Fragmenten zwischendurch diese Gruppe kurzfristig beendet und danach wieder gestartet werden muss.

5.10.3 Abschließende Bewertung

Zusammenfassend kann gesagt werden, dass Firmware-Updates via LoRaWAN ein praxisrelevanter Anwendungsfall sind. Das Ergebnis erfüllt alle Erwartungen und liefert einen brauchbaren Mechanismus, mit dem verlässlich und, im Rahmen der Bedingungen, die

LoRaWAN stellt, auch effizient Updates übertragen werden können. Um dieses Ziel zu erreichen, musste jedoch viel Zeit in dessen Realisierung investiert werden.

Die größte Herausforderung ist beim Einsatz von Updates dabei die durch die Sende-Beschränkungen und langsame Datenübertragungen gegebene hohe Sendezeit. Besonders bei leistungsstärkeren Systemen wie dem ESP32 dauern Updates aufgrund der größeren Firmware mehrere Stunden, in denen das Gerät bezüglich des Datentransfers blockiert wird. Alle anderen Probleme und Herausforderungen, die durch den Versand von Updates via LoRaWAN gegeben sind, können mit dem implementierten Protokoll bewältigt werden.

Ein weiteres Problem, das jedoch durch die lange Dauer der Updates entsteht, ist der hohe Stromverbrauch. Anders als im normalen Betrieb, in dem Endgeräte typischerweise lange Phasen in einem Schlaf-Modus verbringen können, muss das Endgerät für die Dauer der Übertragung des Updates aktiv bleiben. Besonders bei regelmäßigen Updates kann sich dies signifikant auf die Lebensdauer der Batterie auswirken.

5.10.4 Vorschläge zur Verbesserung der Übertragungsdauer

Der größte Problempunkt des Updatemechanismus liegt in der hohen Dauer des Transfers. Diese lässt sich jedoch nicht ohne Weiteres umgehen, da durch die Verwendung von LoRa der Zeitrahmen vorgegeben ist. Im Folgenden sollen jedoch einige Vorschläge diskutiert werden, mit denen diese Zeit sich verringern lässt.

Reduktion des Einflusses des Duty-Cycles

Die Begrenzungen durch den Duty-Cycle sind einer der größten Faktoren, die zu der hohen Dauer von Updates beitragen. Eine Möglichkeit, die Updates zu beschleunigen, wäre dabei, mehrere Gateways in Kombination zu verwenden. Dabei würde ein Gateway mit maximaler Geschwindigkeit senden, bis der Duty-Cycle erreicht ist, und dann den Transfer vom nächsten Gateway fortsetzen lassen [26]. Bei einem Duty-Cycle von 10% würden somit theoretisch zehn Gateways benötigt werden, um permanent im Multicast senden zu können. Während der Einsatz von zehn Gateways unrealistisch erscheint, könnte selbst mit nur zwei Geräten der Durchsatz zumindest verdoppelt werden.

Kompression

Durch vorherige Kompression der Daten können in der Regel große Mengen an Daten eingespart werden. Insofern ist es naheliegend, auch die Firmware vor dem Transfer zu komprimieren. Während der dadurch gewonnene Vorteil offensichtlich ist, gibt es dabei jedoch auch einige Nachteile, die vor dem Einsatz von Kompression beachtet werden sollten.

1. Wahl des Algorithmus: Es sollte ein Algorithmus gewählt werden, welcher effizient auf Mikrocontrollern zu implementieren ist. Die genauen Anforderungen hängen vom konkret eingesetzten System ab. Besonders der begrenzte Arbeitsspeicher kann problematisch werden. Auch sollte der Algorithmus nicht zu viel Programmspeicher benötigen, damit sich die Kompression noch lohnt.
2. Speicher-Mangel bei Dekompression: In der Theorie benötigt das Zielsystem genug Speicher um sowohl das komprimierte als auch das unkomprimierte Firmware-Image im Flash speichern zu können. Falls dies nicht der Fall ist, kann es zur Korruption von Daten kommen. Dekompression beim Empfangen ist nicht möglich, da der Fragmentierungs-Algorithmus die komprimierten Daten für ggf. notwendige Korrekturen benötigt.

Delta-Updates

Neben der Verwendung von klassischer Kompression gibt es auch noch die Möglichkeit, Delta-Updates anzuwenden. Bei Delta-Updates werden nur die Teile der Firmware übertragen, die sich tatsächlich geändert haben. Besonders bei kleineren Updates, die kaum Code verändern, lohnt sich der Einsatz dieser, um weniger Daten übertragen zu müssen. Auch hier gibt es dabei einige Punkte, die beim Einsatz von Delta-Updates beachtet werden müssen.

1. Atomare Updates: Es muss ein Algorithmus gewählt werden, der nicht ein existierendes Image modifiziert, sondern aus diesem und dem Delta-Update ein neues Image an einer anderen Stelle im Speicher generiert. Andernfalls könnten Fehler beim Update das Gerät unbrauchbar machen.
2. Speicher-Mangel bei Dekompression: Aufgrund der Eigenschaften des verwendeten Fragmentierungs-Algorithmus ist es notwendig, zuerst das komplette Delta-Update zu speichern, bevor es angewendet werden kann. Es muss also Speicher sowohl für das Delta-Update, als auch für das daraus generierte, neue Firmware-Image vorhanden sein.
3. Auswirkungen auf den Multicast-Transfer: Ein Delta-Update setzt voraus, dass auf dem Endgerät eine bestimmte Firmware-Version vorhanden ist. Im Falle von Multicast-Übertragungen müssen also alle Empfänger über die gleiche Version verfügen. Falls dies nicht der Fall ist, was in der Praxis oftmals passieren kann, müssen mehrere, verschiedene Delta-Übertragungen durchgeführt werden. Dies kann dazu führen, dass am Ende mehr Daten versendet werden, als bei einer normalen Übertragung vonnöten währen.

Als ein Beispiel kann hier die Implementierung von mbedOS aufgeführt werden, welche Delta-Updates implementiert [26].

Alternativer Fragmentierungs-Algorithmus

Während der aktuell verwendete Algorithmus sehr gut geeignet ist, um für die Codierung von Datenpaketen, die via LoRaWAN versendet werden, eingesetzt zu werden, ist es durchaus möglich, dass noch andere solche Algorithmen existieren, die sogar noch bessere Ergebnisse erzielen können. Besonders in Kombination mit den vorherigen Verbesserungsvorschlägen gibt es u. U. Möglichkeiten, zu optimieren. In diese Richtung wurden in der Arbeit jedoch keinerlei Nachforschungen betrieben, insofern kann an dieser Stelle keine Aussage getroffen werden, ob solche Algorithmen existieren.

6 Methode zur Erkennung des Einwurfs eines Briefes in einen Briefkasten

Eine wichtige Fähigkeit eines Smarten Briefkastens ist es, zu erkennen, ob sich Post in diesem befindet oder nicht. Die Anforderungen an diesen Mechanismus, mögliche Lösungen zur Realisierung dieser Anforderungen und der praktische Test dieser werden in diesem Kapitel diskutiert mit dem Ziel, eine Methode auszuwählen, die in der finalen Anwendung verwendet werden soll.

6.1 Methodik

Um eine passende Technologie für die Lösung des Problems zu finden, wird nach folgendem Schema vorgegangen. Zuerst werden in einer theoretischen Analyse die Anforderungen an diese im Rahmen der Arbeit ermittelt. Danach werden aus diesen einige mögliche Technologien ausgewählt, die diese Kriterien prinzipiell erfüllen und deren Eignung für das Projekt evaluiert.

Die auf diese Weise ausgewählte Hardware wird dann einem praktischen Test unterzogen, um die Eignung zu validieren und eventuelle Einschränkungen bezüglich des praktischen Einsatzes zu ermitteln. Aus diesen Daten wird dann eine Variante ausgewählt, die die Anforderungen am besten erfüllt, um für den Smarten Briefkasten verwendet zu werden.

6.1.1 Anforderungen an den Mechanismus

Die Anforderungen an einen solchen Mechanismus können sich je nach Anwendungsfall stark unterscheiden. Für den Smarten Briefkasten sind folgende Punkte von Relevanz:

1. **Binäre Detektion:** Es ist nur notwendig zu erkennen, ob sich Post im Briefkasten befindet oder nicht. Die Anzahl, Größe oder Art von Post ist nicht von Interesse und muss daher auch nicht erkannt werden.
2. **Keine Echtzeit-Anforderungen:** Es muss nicht sofort, sobald ein Brief eingeworfen wurde, erkannt werden, dass dies geschehen ist. Es ist ausreichend, wenn in einem regelmäßigen Intervall geprüft wird, ob Post im Briefkasten liegt oder nicht.
3. **Energieeffizienz:** Idealerweise sollte sich die Sensorik zwischen zwei solchen Prüfungen ausschalten lassen, um Energie zu sparen. Lange Kalibrierungen beim Initialisieren eines Sensors sind also eher nachteilig.
4. **Kostengünstig:** Die benötigte Hardware sollte kostengünstig und robust sein.
5. **Verlässlichkeit:** Post kommt in verschiedensten Formen. Auch die Position innerhalb des Briefkastens ist dem Zufall überlassen. Die gewählte Technologie sollte in der Lage sein, Post unter allen relevanten Umständen erkennen zu können.

6.1.2 Kriterien der praktischen Evaluation

Während mithilfe der theoretischen Analysen die generelle Eignung weitestgehend festgestellt werden kann, dienen die praktischen Untersuchungen der Verifizierung dieser, sowie der Ermittlung der folgenden Kriterien.

1. **Funktionalität:** Es soll sichergestellt werden, dass der Sensor überhaupt in der Lage ist, Objekte in seiner Nähe zu detektieren.

2. **Reichweite:** Es sollen die minimale und maximale Reichweite des Sensors unter optimalen Bedingungen ermittelt werden. Auf diese Weise lässt sich eingrenzen, wie groß der Briefkasten sein darf.
3. **Erkennung von Papier:** Es soll ermittelt werden, ob der Sensor Papier erfolgreich erkennen kann und unter welchen Bedingungen dies der Fall ist.
4. **Abdeckung:** Während in Kriterium 2 die generelle Reichweite ermittelt wurde, soll mit diesem Punkt die räumliche Abdeckung, die mit einem Sensor erreicht werden kann, festgestellt werden.

6.2 Mögliche Sensorik

In diesem Abschnitt werden einige mögliche Technologien diskutiert, mit denen theoretisch erkannt werden könnte, ob ein Brief eingeworfen wurde oder nicht. Dabei wird jeweils direkt ausgewertet, ob die Technik alle Kriterien aus Unterunterabschnitt 6.1.1 erfüllt und ob somit eine weiterführende praktische Analyse überhaupt notwendig ist.

6.2.1 Infrarot Lichtschranke

Bei dieser Variante werden eine Infrarot-Diode und eine Infrarot-Fotodiode gegenüber innerhalb des Briefkastens angebracht. Kann die Fotodiode ein Infrarot-Signal empfangen, so ist der Briefkasten leer. Andernfalls befindet sich etwas darin.

Diese Variante erfüllt Kriterium 1–4 ohne Probleme. Einfache Dioden und Fotodioden sind kostengünstig zu erwerben. Energieeffizienz wird aufgrund der Möglichkeit, die Dioden zwischen zwei Messungen auszuschalten, und der Möglichkeit, diese mit nur so viel Leistung zu betreiben, dass die Reichweite grade so ausreichend ist, als erfüllt angesehen.

Der einzige Problempunkt ist Kriterium 5. Faktoren wie die Dicke der eingeworfenen Post, einfallendes Sonnenlicht, die maximale Reichweite oder die Möglichkeit, dass ein Brief so fällt, dass er nicht von der Lichtschranke erfasst wird, sind alles Punkte, die sich negativ auf die Verlässlichkeit auswirken können. Der letzte Punkt kann dabei durch den Einsatz mehrerer solcher Lichtschranken in verschiedenen Winkeln ausgeglichen werden, die anderen Punkte müssen jedoch praktisch untersucht werden.

6.2.2 Infrarot Distanzmessung

Eine Alternative zur Lichtschranke ist der Infrarot-Distanz-Messer. Dieser sendet regelmäßige Infrarot-Pulse, die von Objekten im Briefkasten reflektiert und dann wieder von dem Sensor empfangen werden. Aus diesen Daten kann die Distanz des Objektes bestimmt werden.

Während eine Distanzmessung an sich nicht benötigt wird, erfüllt diese trotzdem den Zweck der Detektion von Post im Briefkasten. Auch hier kann der Sensor ausgeschaltet bleiben, solange nichts gemessen werden muss, womit die Kriterien 1–4 auch hier ohne Weiteres erfüllt werden.

Was die Verlässlichkeit, also Kriterium 5 betrifft, so kann ohne praktische Messungen keine endgültige Aussage getroffen werden. Entscheidend ist jedoch der Winkel, unter dem der Sensor Objekte erkennen kann. Anders als bei der Infrarot-Lichtschranke können hier nicht mehrere Sensoren kombiniert werden, da diese gegenseitige Interferenzen auslösen würden. Ein weiterer Problempunkt ist die unterschiedliche Reflexivität von verschiedenen Gegenständen, was dazu führen könnte, dass bestimmte Arten von Post nicht erfolgreich erkannt werden können.

6.2.3 Ultraschall Distanzmessung

Eine weitere Variante der Distanzmessung ist die mit Ultraschall. Dabei sendet ein Lautsprecher ein Ultraschall-Signal, welches von Objekten im Briefkasten reflektiert und dann wieder von dem Sensor detektiert wird. Aus diesem Echo kann dann die Distanz des Objekts berechnet werden.

Ultraschall-Sensoren sind teurer als Infrarot-Sensoren, aber immer noch in der unteren Preisklasse. Auch diese Sensoren können in den meisten Fällen ohne Kalibrierung betrieben werden und erfüllen somit die Kriterien 1–4.

Was die Verlässlichkeit (Kriterium 5) des Sensors betrifft, so gibt es zwei Bedenken. Das erste Problem sind mögliche Interferenzen innerhalb des Briefkastens durch Reflexion von Schallwellen an dessen Wänden. Des Weiteren haben auch diese Sensoren nur begrenzte Detektionswinkel. Ob mehrere dieser Sensoren in Kombination eingesetzt werden können, um alle Richtungen abzudecken ist fraglich, da auch hier Interferenzen möglich sind. Im Ausgleich dafür können mit Ultraschall alle Objekte gleichermaßen detektiert werden, die Textur dieser ist dabei nicht von Relevanz.

Aufgrund von Zeitgründen wurde dieser Typ von Sensor keiner praktischen Analyse unterzogen. Ob zuvor genannte Punkte also problematisch sind oder nicht, und ob dieser Typ von Sensor letztendlich geeignet ist, um Post in einem Briefkasten zu detektieren, kann somit nicht beantwortet werden.

6.2.4 Kraftmessung

Bei einer Kraftmessung wird ein Kraftmessgerät am Boden des Briefkastens angebracht, welches den Gewichtsunterschied detektiert, der durch den Einwurf von Post ausgelöst wird. Bei dieser Variante fallen gleich mehrere Punkte auf, die deren Anwendung problematisch machen. Dabei wurde der Honeywell FSG005WNPB als Referenz verwendet [22].

Empfindlichkeit und Robustheit Damit ein solcher Sensor angewendet werden kann, muss dieser empfindlich genug sein, um ein einzelnes Blatt Papier zu erkennen, aber trotzdem robust genug, um das Gewicht eines Buches auszuhalten. Diese Anforderungen stehen im Konflikt miteinander. Ein Blatt Papier beispielsweise erzeugt eine Kraft von 0,05 N, was gerade mal der fünffachen Auflösung des Sensors entspricht; mehrere gestapelte Bücher können jedoch ohne weiteres die maximalen 15 N, die der Sensor aushält, überschreiten. Des Weiteren müsste der Sensor breit genug sein, um den kompletten Boden des Briefkastens abzudecken, oder aber es müsste eine Bodenplatte auf den Sensor aufgesetzt werden. Hinzu kommt noch, dass solche Sensoren schockempfindlich sind. Der Einwurf eines Gegenstands mit mehr als 150 g könnte beispielsweise den referenzierten Sensor zerstören. Kriterium 5 kann somit nicht erfüllt werden.

Kosten Besonders Kraftmesser mit einer hohen Empfindlichkeit sind verhältnismäßig teuer, was deren Einsatz in großer Anzahl erschwert. Hinzu kommt, dass durch die Notwendigkeit einer Bodenplatte Briefkästen modifiziert werden müssten, um mit diesem Sensortyp ausgestattet werden zu können. Kriterium 4 kann somit auch nicht erfüllt werden.

Die obigen Argumente zeigen eindeutig, dass ein solcher Sensor für den Einsatz, der in dieser Arbeit vorgesehen ist, nicht geeignet ist. Aus diesem Grund wurden auch keine weiteren praktischen Untersuchungen durchgeführt.

6.2.5 Klappen-Detektor

Die letzte Art von Sensoren, die untersucht wurde, sind Detektoren, die erkennen, wenn der Briefkasten geöffnet wird. Dazu gehören einfache elektrische Schalter verschiedenster Art. Ein solcher Sensor schließt einen Messkreis, sobald die Klappe des Briefkastens geöffnet wird und erkennt somit, dass etwas eingeworfen wurde. Dieser Mechanismus setzt also voraus, dass der Briefkasten über eine Klappe verfügt. Ist dies nicht der Fall, kann er auch nicht eingesetzt werden.

Kriterium 1, 4 und 5 können dabei sofort als erfüllt betrachtet werden. Diese Art von Schalter ist sehr günstig und da ein Brief nicht eingeworfen werden kann, ohne dass die Klappe geöffnet wird, wird jeder Einwurf garantiert erkannt. Auch Kriterium 3 kann als erfüllt angesehen werden, solange der Messkreis offen bleibt, wenn der Briefkasten geschlossen ist. Falls der Mikrocontroller in der Lage ist, in den Tiefschlaf versetzt zu werden, kann auch ein permanent geschlossener Schaltkreis verwendet werden, der den Mikrocontroller aufgeweckt, wenn dieser ausgelöst wird.

Letztendlich wurde entschieden, diese Variante nicht zu verwenden, da sie das Vorhandensein einer Klappe erfordert. Dies ist eine Anforderung, die diese Arbeit nicht stellt, und somit muss der Sensor ausgeschlossen werden. Andernfalls ist dies die zu bevorzugende Methode, da sie am verlässlichsten arbeitet, ohne dabei beim Preis oder dem Energieverbrauch negativ aufzufallen. Praktische Analysen wurden entsprechend auch hier nicht durchgeführt.

6.3 Verwendete Hardware

In Unterabschnitt 6.2 wurden zwei verschiedene Arten von Sensoren ermittelt, für die im Folgenden weitere praktische Tests durchgeführt wurden. Für diese wird hier die verwendete Hardware aufgelistet, die getestet wurde.

6.3.1 Infrarot Lichtschranke

Für diesen Sensortyp wurde der Adafruit IR Breakbeam Sensor ausgewählt [1]. Dieser besteht aus einer Emitter-LED, welche den Infrarot-Lichtstrahl erzeugt, und einer Fotodiode, welche diesen empfängt.

Anschluss

Der Sensor besteht aus zwei getrennten Komponenten. Erstere ist die LED, welche bereits mit einem Innenwiderstand versehen ist. Bei den von dem Mikrocontroller bereitgestellten 3,3 V verbraucht diese ca. 10 mA. Die zweite Komponente ist die Fotodiode, welche aktiv ist, wenn der Infrarot-Lichtstrahl empfangen wird.

In Abbildung 33 wird dargestellt, wie die beiden Elemente an den Mikrocontroller angeschlossen werden. Dabei werden sowohl die LED als auch die Fotodiode mit einer 3,3 V Spannung versorgt. In diesem Versuch wurde dabei die konstante Spannungsversorgung des Mikrocontrollers verwendet. Genauso kann aber auch einer der GPIO-Pins verwendet werden, um die beiden Halbleiter anzusteuern.

Ansteuerung

Um diese Kombination anzusteuern, müssen zuerst beide Elemente mit Spannung versorgt werden. Die LED sendet dann einen Infrarotstrahl aus, der auf den Sensor gerichtet werden muss. Ist der Strahl verbunden, so kann eine Spannung von 3,3 V an P12 gemessen werden. Ist er unterbrochen, so sind es stattdessen 0 V.

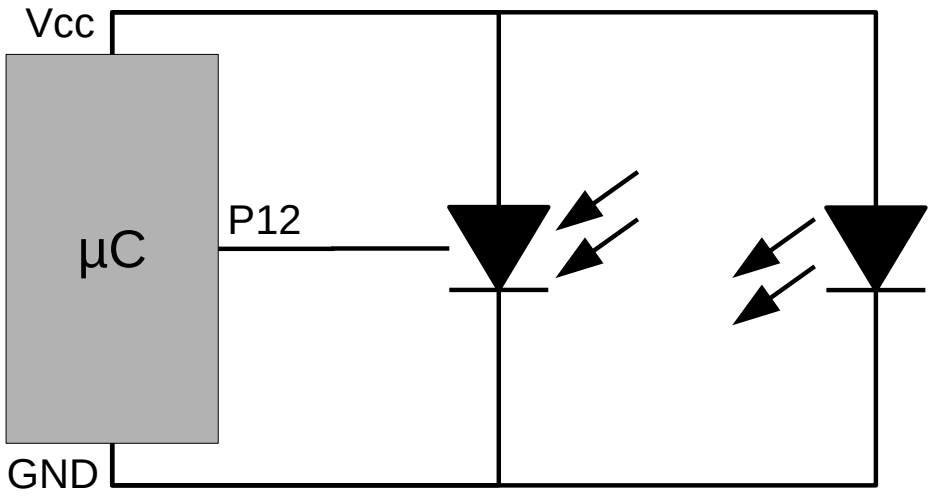

Abbildung 33: Versuchsaufbau Messung IR-Breakbeam Sensor

6.3.2 Infrarot Distanzmessung

Als Hardware für diesen Test wurden der TSSP4038 Proximity Sensor und die VSLB3940 Infrarot-LED von Vishay verwendet [63][62]. Zweitere dient dazu, LED-Impulse zu erzeugen, die an einem Objekt reflektiert und anschließend von dem Sensor gemessen werden.

Anschluss

Auch hier sind wieder zwei getrennte Komponenten vorhanden. Die LED muss dabei, wie in Abbildung 34 zu sehen, mit einem Widerstand von $R_1 = 330\ \Omega$ versehen werden, um den Strom auf 10 mA zu beschränken. Der Sensor selbst kann direkt an den Mikrocontroller angeschlossen werden. Beide werden mit einer Spannung von 3,3 V versorgt.

Ansteuerung

Zuerst muss der Sensor mit einer permanenten Spannung versorgt werden. Dieser legt nun standardmäßig ein Signal von 3,3 V an P12 an, welches auf 0 V reduziert wird, wenn dieser einen Infrarotpuls detektiert. Um diese zu erzeugen wird die LED über den GPIO-Pin P11 angesteuert und periodisch ein- und wieder ausgeschaltet.

Der Sensor kann dabei bei Pulsen von jeweils unterschiedlichen Frequenzen verschiedene Distanzen messen. Um ein Objekt in der Reichweite des Sensors zu erkennen, muss die Puls-Frequenz der LED zwischen 56 kHz und 32 kHz variiert werden [19]. Für jede Frequenz wird geprüft, ob der Sensor diese erkennt und somit ermittelt, wo sich das Objekt befindet. Da für diese Arbeit jedoch nur erkannt werden muss, ob sich ein Objekt in Reichweite befindet, und nicht, welche Distanz es hat, kann einfach eine feste Frequenz (z. B. 56 kHz für die maximale Reichweite) gewählt werden.

6.4 Evaluation der Hardware

In diesem Abschnitt wird die Durchführung der Tests für die Kriterien aus Unterunterabschnitt 6.1.2 beschrieben. Dazu wurde jeweils ein Testprogramm geschrieben, welches den Sensor steuert und ausgibt, ob ein Objekt detektiert wurde oder nicht. Um Reichwei-

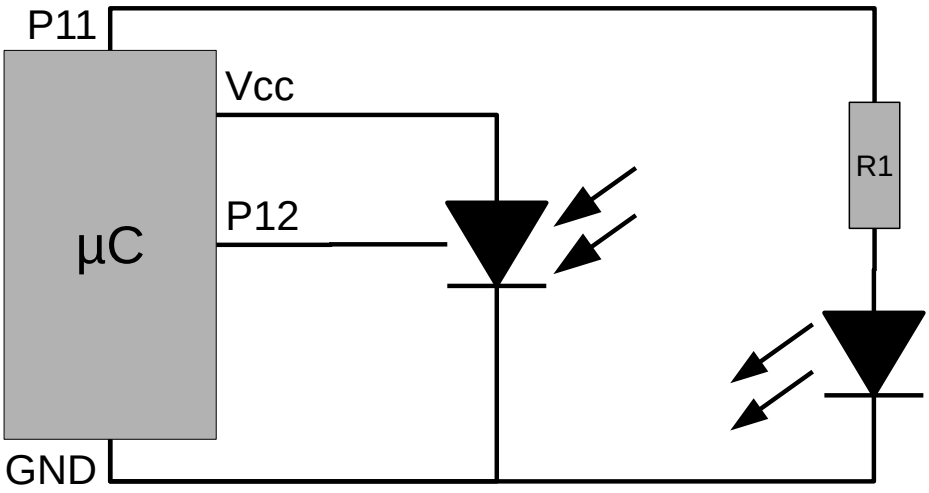

Abbildung 34: Versuchsaufbau Messung IR-Proximity Sensor

ten zu messen wurde ein Lineal verwendet und der Abstand zwischen den Testobjekten damit experimentell ermittelt. Dabei wurde eine Genauigkeit von 1 cm als Anforderung festgelegt, da für den geplanten Anwendungsfall keine exakten Werte benötigt werden.

6.4.1 Infrarot Lichtschranke

Für diesen Test konnte ein minimales Programm geschrieben werden, welches lediglich den Pin P12 beobachtet und dessen Änderungen ausgibt. Die Messergebnisse werden in Tabelle 16 aufgelistet.

Kriterium	**Ergebnis**
Funktionalität	Ist gegeben. Der Sensor kann Papier erkennen.
Reichweite	Unter optimalen Bedingungen konnte eine Reichweite von maximal 45 cm erreicht werden.
Erkennung	Papier kann problemlos erkannt werden, benötigt jedoch einen Mindestabstand von 10 cm, da andernfalls das Licht durch das Papier durchscheint. Außerdem wird der Sensor durch Sonnenlicht aktiviert.
Abdeckung	Der Strahl deckt nur einen Bereich in einer geraden Linie ab. Dafür kann jedes Objekt erkannt werden, dass diese Linie unterbricht.

Tabelle 16: Auswertung der Infrarot-Lichtschranke

Des Weiteren wurde ein zweites Experiment durchgeführt, bei dem gemessen wurde, bis zu welcher Distanz, in Abhängigkeit vom Winkel zu der LED, der Sensor noch ein Signal empfängt (siehe Abbildung 35). Während dies keine Aussage bezüglich der Detektion von Objekten trifft, da LED und Diode fest gegenüber montiert werden, kann mit dieser

Untersuchung ausgesagt werden, inwiefern mehrere verschiedene solcher Sensoren interferieren können. Das Ergebnis dieser Messung wird in Abbildung 36 dargestellt.

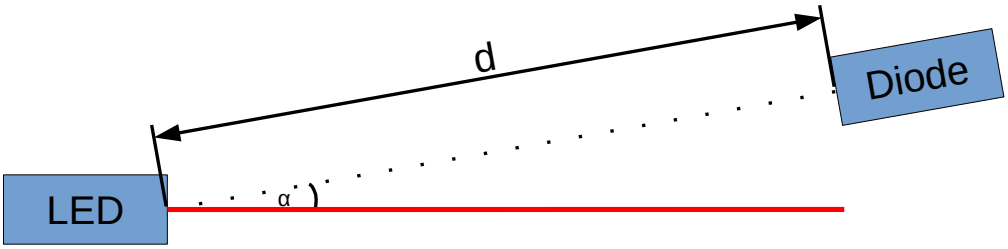

Abbildung 35: Versuchsaufbau Messung IR-Breakbeam Winkel

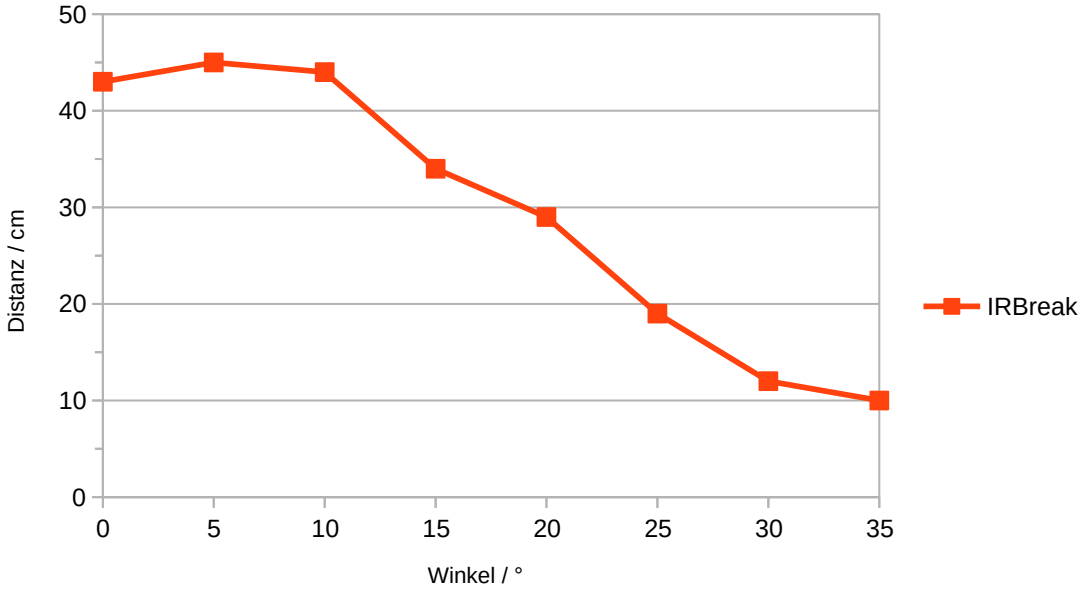

Abbildung 36: Messung IR-Breakbeam Winkel

6.4.2 Infrarot Distanzmessung

Auch für diesen Sensor-Test wurde ein Programm geschrieben. Dieses muss dabei mithilfe von Hardware-Timern die LED und den Sensor ansteuern, um nacheinander verschiedene Frequenzen zu testen und somit die Distanz (und damit Existenz) eines Objekts zu ermitteln. Die Ergebnisse der Untersuchungen sind in Tabelle 17 dargestellt. Dabei wurde für verschiedene Frequenzen und Winkel die Reichweite des Sensors ermittelt, um einen Überblick über die Abdeckung zu erhalten.

Dabei ist auffällig, dass obwohl laut der Dokumentation des Sensors dieser zwischen 32 kHz und 56 kHz operiert, 65 kHz als optimale Frequenz gemessen wurde. Der Grund dafür konnte nicht ermittelt werden, es wird allerdings vermutet, dass aufgrund von Ungenauigkeiten und der Trägheit des Mikrocontrollers bei eingestellten 65 kHz tatsächlich nur 56 kHz realisiert werden. Hier müsste genauer analysiert werden, woher diese Ungenauigkeit kommt. Im Rahmen dieser Arbeit kann dies jedoch ignoriert werden, da nur die maximale Frequenz verwendet wird, egal wie hoch diese nun tatsächlich ist.

Kriterium	Ergebnis
Funktionalität	Ist gegeben. Der Sensor kann Papier erkennen.
Reichweite	Unter optimalen Bedingungen konnte eine Reichweite von maximal 44 cm erreicht werden. Diese hängt von der Blink-Frequenz der LED ab, wie in Abbildung 37 zu sehen.
Erkennung	Papier kann problemlos erkannt werden und benötigt keinen Mindestabstand. Der Sensor wird nicht durch Sonnenlicht gestört.
Abdeckung	Der Sensor deckt einen Bereich bis maximal 35° ab, mit annähernd linear abnehmender Distanz, was in Abbildung 37 dargestellt wird.

Tabelle 17: Auswertung der Infrarot-Distanz-Messung

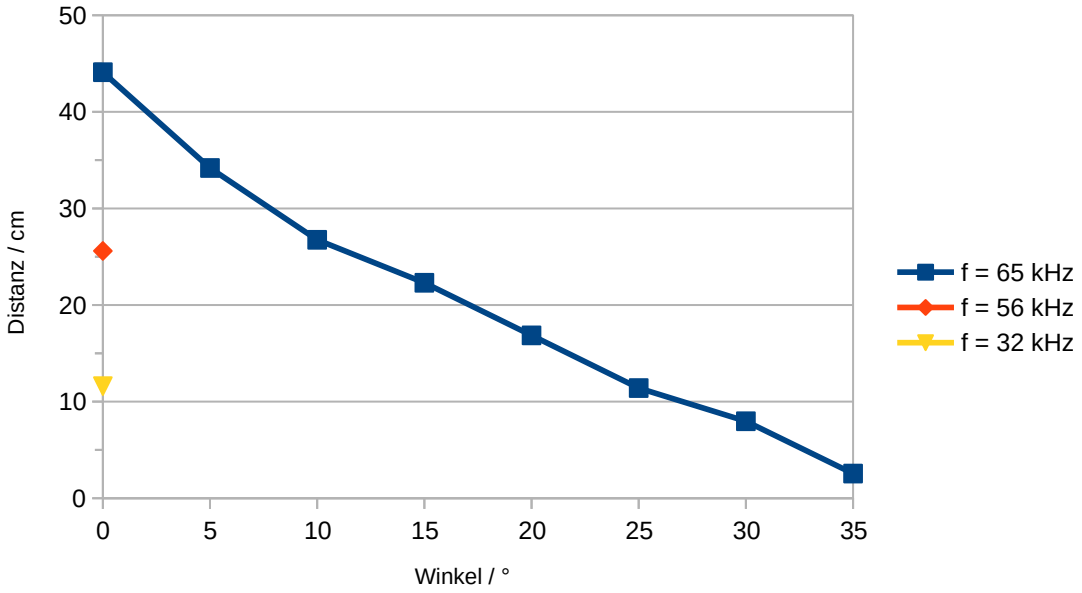

Abbildung 37: Reichweite des TSSP4038 in Abhängigkeit von Winkel und Frequenz

6.5 Ergebnisse

In der theoretischen Analyse wurden die möglichen Sensor-Typen auf drei Varianten beschränkt, von denen zwei praktisch getestet wurden. Beide zeigen akzeptable Ergebnisse und sind somit prinzipiell geeignet für den Einsatz als Brief-Detektor im Smarten Briefkasten. Im Folgenden werden die Vor- und Nachteile der beiden diskutiert und diese anschließend verglichen.

6.5.1 Verwendung der Infrarot Lichtschranke

Die Montage der Lichtschranke innerhalb des Briefkastens hängt von dem Formfaktor des Kastens ab. Dabei müssen die LED und die Diode gegenüber platziert werden, mit einem Abstand zwischen 10 cm und 45 cm. Um die konkrete Positionierung zu ermitteln, sollten mehrfach Briefe in den Kasten eingeworfen werden, um herauszufinden, wie Post innerhalb des Kastens liegen kann. Somit kann ein Punkt gefunden werden, an dem der

Strahl alle möglichen Positionen von Briefen abdeckt. Falls die gegenüberliegende Montage problematisch ist, ist es ebenfalls möglich, die LED und den Sensor nebeneinander anzubringen, mit einem Reflektor an der gegenüberliegenden Wand. Damit halbiert sich jedoch die Reichweite des Sensors.

Wird festgestellt, dass die vollständige Abdeckung des Briefkastens mit nur einem Sensor nicht möglich ist, müssen mehrere verschiedene Paare installiert werden. Dabei muss Abbildung 36 beachtet werden, sodass die Sensoren sich nicht gegenseitig stören. Des Weiteren muss darauf geachtet werden, dass zu keiner Tageszeit einer der Sensoren von Sonnenlicht getroffen wird. Auch indirekte Einstrahlung kann schon genügen, um den Sensor auszulösen.

6.5.2 Verwendung des Infrarot Distanzsensors

Komplizierter ist hingegen die Montage des Distanzsensors. Auch hier hängt die endgültige Position von dem Formfaktor des Briefkastens ab, allerdings gelten dabei zusätzliche, einschränkende Faktoren.

Das primäre Problem, das dabei entsteht, ist, dass typischerweise auch die Innenwände des Briefkastens das Infrarot-Signal reflektieren. Dies macht es unmöglich, einen Gegenstand von der Dicke eines Blattes zu erkennen. Um dieses Problem zu umgehen, gibt es zwei Möglichkeiten. Es können entweder die Innenwände des Briefkastens mit einem Material ausgekleidet werden, das Infrarotlicht absorbiert, oder der Sensor auf dem Boden positioniert werden.

Für die Zweite Variante muss dabei beachtet werden, dass Gegenstände, die direkt auf dem Sensor liegen, nicht erkannt werden können. Außerdem muss die Frequenz so gewählt werden, dass die Wände des Kastens außerhalb der Reichweite des Sensors liegen.

Neben diesem grundsätzlichen Problem muss auch hier experimentell ermittelt werden, wie Briefe in dem Kasten liegen können, um zu ermitteln, welcher Bereich abgedeckt werden muss. Dabei gibt Abbildung 37 an, welchen Bereich der Sensor abdecken kann. Ist dieser nicht ausreichend, so müssen mehrere Sensoren platziert werden, ohne dass diese sich gegenseitig Stören.

6.5.3 Vergleich der Lösungen

Vergleicht man die beiden Varianten, so kann festgestellt werden, dass mit beiden eine fast identische Distanz abgedeckt werden kann. Die Infrarot-Lichtschranke ist einfacher anzusteuern, dafür aber anfällig für Sonnenlicht und weniger flexibel.

Ein Vorteil, den der Distanzsensor gegenüber der Lichtschranke hat, ist der, dass dieser einen größeren Bereich abdecken kann. Bis zu einem Winkel von 15° kann noch ca. die Hälfte der maximalen Distanz erkannt werden. Im Ausgleich dafür ist es jedoch deutlich aufwendiger, diesen zu verbauen.

Letztendlich sind beide Lösungen geeignet, um eingeworfene Post zu erkennen. Darüber, welche Variante allgemein besser ist, kann keine Aussage getroffen werden, da dies stark von dem konkret verwendeten Briefkasten abhängt.

Da keine eindeutig bessere Variante ermittelt werden konnte, wurde entschieden, die Variante zu wählen, die einfacher zu realisieren ist. Da sowohl in der Ansteuerung als auch beim Einbau die Lichtschranke einfacher zu handhaben ist, wurde diese als Sensor für den Smarten Briefkasten ausgewählt.

7 Entwicklung des Smarten Briefkastens

Der letzte Schritt in dieser Arbeit ist die Entwicklung des Smarten Briefkastens. Die zugehörigen Schritte werden im Folgenden erläutert und das Design sowie die Realisierung des Briefkastens beschrieben. Dabei werden die aus den vorherigen Kapiteln ermittelten Ergebnisse vorausgesetzt und dienen als Basis für den Briefkasten.

7.1 Methodik

Der erste Schritt bei der Entwicklung des Briefkastens war der Entwurf der Anwendung. Dazu wurden zuerst die Anforderungen an den Briefkasten ermittelt und aus diesen die Anwendung entworfen. Auf dieser Basis wurde dann die zu verwendende Hardware ausgewählt und die Anwendung implementiert. Anschließend wurde ein praktischer Funktionstest durchgeführt, um die Funktionalität des Briefkastens zu gewährleisten. Abschließend wurde ermittelt, ob alle Anforderungen aus Unterunterabschnitt 7.2.1 erfüllt wurden.

7.2 Anwendungs-Entwurf

Im Folgenden wird das theoretische Design der Anwendung und deren Architektur beschrieben. Dazu werden zuerst einige Anforderungen ermittelt, aus denen anschließend die verschiedenen Komponenten der Anwendung und deren Design abgeleitet werden.

7.2.1 Anforderungen

Prinzipiell besteht die Anwendung aus zwei primären Komponenten, dem Smarten Briefkasten und der Client-Anwendung. Beim Smarten Briefkasten handelt es sich um die Hardware, welche in den Briefkasten eingebaut wird, um eingeworfene Post zu detektieren. Die Client-Anwendung hingegen ist eine reine Software-Anwendung, welche vom Nutzer auf dessen Smartphone installiert werden kann. Dabei sendet der Smarte Briefkasten Benachrichtigungen über neue Post an die App, sodass diese Information dem Nutzer durch die App angezeigt werden kann. Dabei wurden folgende Anforderungen für die beiden Komponenten ermittelt.

Smarter Briefkasten

1. Der Briefkasten muss dazu in der Lage sein, eingeworfene Post zu erkennen. Diese Detektion muss verlässlich sein; Echtzeit hingegen ist keine Anforderung.
2. Er muss diese Information, sowie den Batterie-Status des Geräts, über LoRaWAN an den Client versenden können. Die Daten sollen dabei E2E-verschlüsselt und authentifiziert sein. Es sollten dabei auch bei unverändertem Zustand Statusinformationen versendet werden können.
3. Die Software sollte vom Client konfiguriert werden können, sodass das Verhalten des Briefkastens an die Bedürfnisse des Nutzers angepasst werden kann. Dabei wurden folgende Einstellungsmöglichkeiten ermittelt:
    - Übertragung des Batterie-Status
    - Senden von Dummy-Nachrichten bei unverändertem Zustand
    - Intervall, in dem der Briefkasten auf Post prüft und ggf. Nachrichten versendet
    - Länge der Authentifizierungs-Daten
4. Der Briefkasten muss dazu in der Lage sein, OTA-Updates zu empfangen und durchzuführen. Des Weiteren muss er diese Updates kryptografisch verifizieren.

5. Die Anwendung muss über einen Offband-Kanal initialisierbar sein, sodass Zugangsdaten für LoRaWAN und der E2E-Schlüssel dynamisch und sicher ausgehandelt werden können.
6. Zwischen dem Versand von zwei Statusnachrichten soll der Mikrocontroller in einen Schlaf-Modus versetzt werden, um Energie zu sparen.

Client-Anwendung

1. Die Anwendung soll für Mobilgeräte entwickelt werden. Im Rahmen dieser Arbeit ist Android als Plattform ausreichend. Prinzipiell soll jedoch so entwickelt werden, dass eine Portierung auf andere Plattformen einfach möglich ist.
2. Sie muss dem Nutzer eine grafische Oberfläche bieten, in der er mehrere Smarte Briefkästen verwalten kann. Es muss möglich sein, diese dynamisch hinzuzufügen, konfigurieren und entfernen zu können.
3. Der Client muss das gleiche Protokoll, welches auch der Briefkasten für den Offband-Beitritt verwendet, unterstützten, sodass der Client den Briefkasten über dieses initialisieren kann.
4. Die Anwendung muss dem Nutzer für jeden verbundenen Briefkasten den letzten verfügbaren Status anzeigen. Die Anwendung sollte diesen automatisch aktualisieren, der Nutzer soll aber auch in der Lage sein, diese manuell durchzuführen.
5. Auch im Hintergrund sollen weiterhin Informationen von den Briefkästen empfangen werden, sodass der Nutzer eine Benachrichtigung über neue Post erhält, selbst wenn die Anwendung nicht dafür geöffnet wurde.

7.2.2 System-Architektur

Aus den Anforderungen ergibt sich bereits, dass es mindestens zwei Komponenten gibt; den Briefkasten und den Client. Allerdings wird noch eine dritte Komponente benötigt: Eine Server-Anwendung, die Nachrichten vom Briefkasten zwischenspeichert, bevor diese an den Client gesendet werden, da dieser nicht permanent erreichbar ist. Es ergibt sich somit die Anwendungs-Struktur aus Abbildung 38. Dabei wurden der LoRaWAN-Stack und der CLI-Updater für die LoRaWAN-Updates mit eingezeichnet.

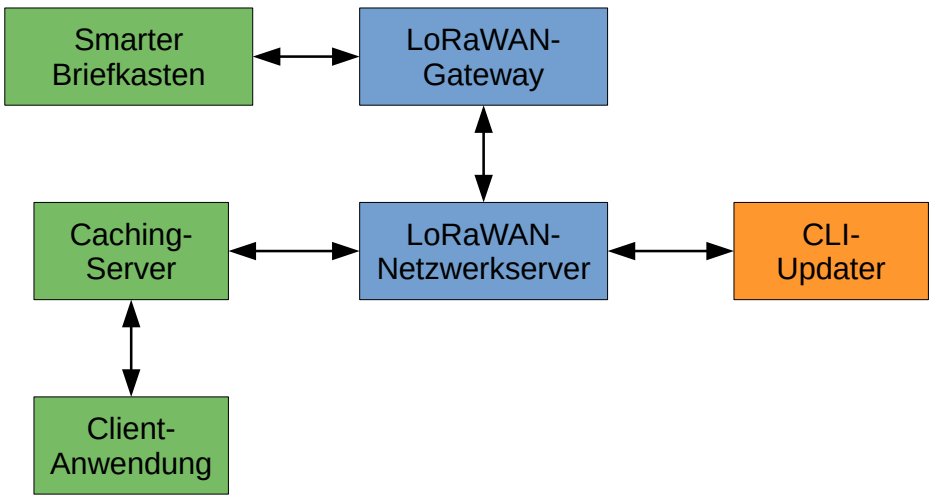

Abbildung 38: Architektur des Smarten Briefkastens

7.2.3 Definition des Postbox-Protokolls

Als nächster Schritt wurde ein Kommunikations-Protokoll zwischen dem Briefkasten, dem Caching-Server und der Client-Anwendung entwickelt. Dabei wurde der Fokus auf die Verbindung zwischen Briefkasten und Server gelegt. Die Kommunikation zwischen diesen beiden erfolgt über LoRaWAN, während die Datenübertragung zwischen Server und Client-Anwendung via HTTP stattfindet. Abbildung 39 zeigt eine Übersicht mit dem Ablauf der Kommunikation. Dabei ist der Offband-Beitritt in Rot eingezeichnet und steht repräsentativ für das Protokoll aus Abschnitt 4. Nachrichten, die in Grün eingezeichnet sind, sind zwischen dem Briefkasten und dem Client E2E-verschlüsselt und können somit vom Server nicht gelesen werden. Konfigurationsnachrichten werden über Port 51, verschlüsselte Datennachrichten über Port 52 versendet.

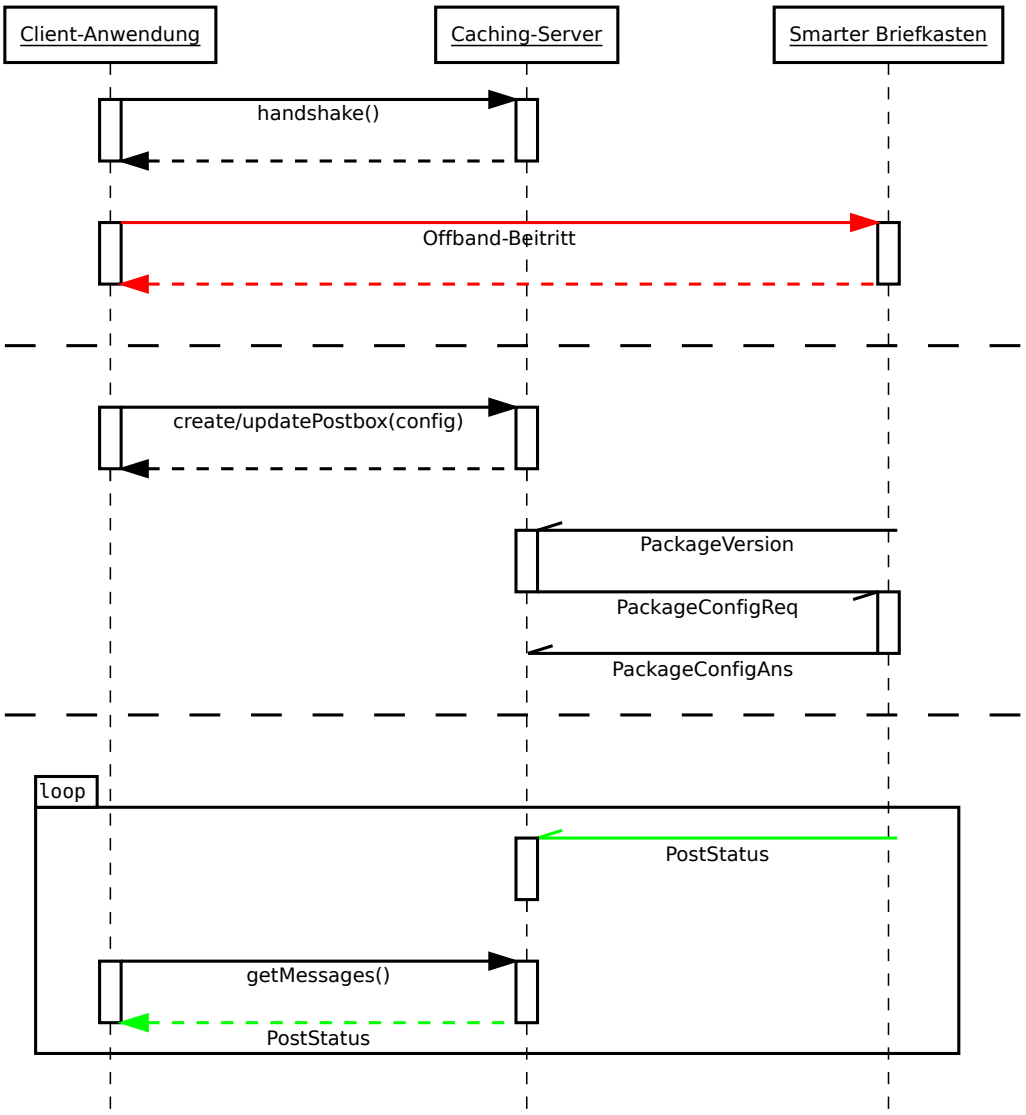

Abbildung 39: Sequenz-Diagramm Postbox-Protokoll

Genauer beschrieben werden an dieser Stelle nur die Nachrichten zwischen dem Briefkasten und dem Server. Das Protokoll für die Kommunikation zwischen Client-Anwendung und Server hingegen wird in Unterunterabschnitt 7.2.5 definiert. Das Protokoll startet mit dem Austausch von Verbindungsdaten zwischen dem Briefkasten und dem Client über einen zweiten Kanal. Sobald der Briefkasten die Zugangsdaten erhalten hat, tritt er dem LoRaWAN-Netzwerk bei und der Datentransfer via LoRaWAN beginnt.

Der Client muss dazu den Briefkasten beim Server registrieren. Der Smarte Briefkasten sendet währenddessen wiederholt so lange die PackageVersion-Nachricht (Code `0x00`, Tabelle 18, Port 51), bis das Gerät beim Caching-Server angemeldet wurde und dieser entsprechend antworten kann. Dabei ist PackageIdentifier auf `101` festgelegt und PackageVersion muss `1` sein.

Feld	PackageIdentifier	PackageVersion
Größe (Bytes)	1	1

Tabelle 18: PackageVersion Payload

Ist das Gerät beim Server angemeldet worden, so antwortet dieser mit der Konfiguration für den Briefkasten, indem eine PackageConfigReq-Nachricht (Code `0x01`, Tabelle 19, Port 51) an diesen versendet wird. Die Konfigurations-Felder entsprechen denen, die in Unterunterabschnitt 7.2.1 definiert wurden. Der Briefkasten bestätigt den Empfang dieser Nachricht mit einer PackageConfigAns-Nachricht (Code `0x01`, keine Payload, Port 51).

Feld	Config
Größe (Bytes)	2

Tabelle 19: PackageConfigReq Payload

Config Felder	RFU	TagLen	Period	SendUnchanged	ReportCharge
Größe (Bits)	6	4	4	1	1

Tabelle 20: PackageConfigReq Config Feld

Das Config-Feld der PackageConfigReq-Nachricht (Tabelle 20) ist dabei wie folgt zu interpretieren. ReportCharge und SendUnchanged sind boolesche Werte, welche angeben, ob die Ladung der Batterie mitgesendet werden soll, und ob auch bei unverändertem Zustand Daten versandt werden. Period codiert die Periode, mit der der Status des Briefkastens transferiert werden soll. Das Zeitintervall berechnet sich zu $\Delta t = 2^{Period}$ (in Sekunden) und ist somit auf das Intervall $[1; 32768]$ s begrenzt. Das TagLen-Feld gibt codiert die Länge l_{MAC} des MACs, welcher an mit GCM-AES verschlüsselte Nachrichten angehangen wird. Diese berechnen sich nach Gleichung 17.

$$l_{MAC} = \begin{cases} 0 & \text{wenn } TagLen = 0 \\ TagLen + 1 & \text{andernfalls} \end{cases} \tag{17}$$

Ab diesem Zeitpunkt ist der Briefkasten konfiguriert und bereit, Daten zu versenden. Wann immer der Status des Briefkastens verfügbar ist, sendet dieser eine PostStatus-Nachricht (Code `0x02`, Tabelle 21, Port 52). Diese ist E2E-verschlüsselt und kann somit nicht vom Server gelesen werden. Stattdessen wird die Nachricht zwischengespeichert, bis die Client-Anwendung diese abholt.

Dabei besteht das Status-Feld der PostStatus-Nachricht (Tabelle 22) aus zwei Werten. Mit dem Mail-Feld wird als boolescher Wert angegeben, ob sich Post im Briefkasten befindet oder nicht. Das Charge-Feld codiert den aktuellen Akku-Zustand des Gerätes. Die

Feld	Status
Größe (Bytes)	1

Tabelle 21: PostStatus Payload

Status Felder	Charge	Mail
Größe (Bits)	7	1

Tabelle 22: PostStatus Status Feld

prozentuale Ladung c_p als Zahl des Intervalls $[0; 1]$, kann mit Gleichung 18 aus diesem Wert berechnet werden.

$$c_p = \begin{cases} 0 & \text{wenn } Charge \leq 13 \\ 1 & \text{wenn } Charge \geq 113 \\ (Charge - 13)/100 & \text{andernfalls} \end{cases} \tag{18}$$

7.2.4 Design des Smarten Briefkastens

Bei dem Design des Smarten Briefkastens musste darauf geachtet werden, dass alle Anforderungen aus Unterunterabschnitt 7.2.1 erfüllt werden. Dazu wurden die verschiedenen Aufgaben, die die Anwendung erfüllen muss, in mehrere Gruppen unterteilt, welche diese übernehmen. Abbildung 40 stellt den Zusammenhang dieser Komponenten untereinander dar. Im Folgenden werden außerdem die Aufgaben dieser beschrieben.

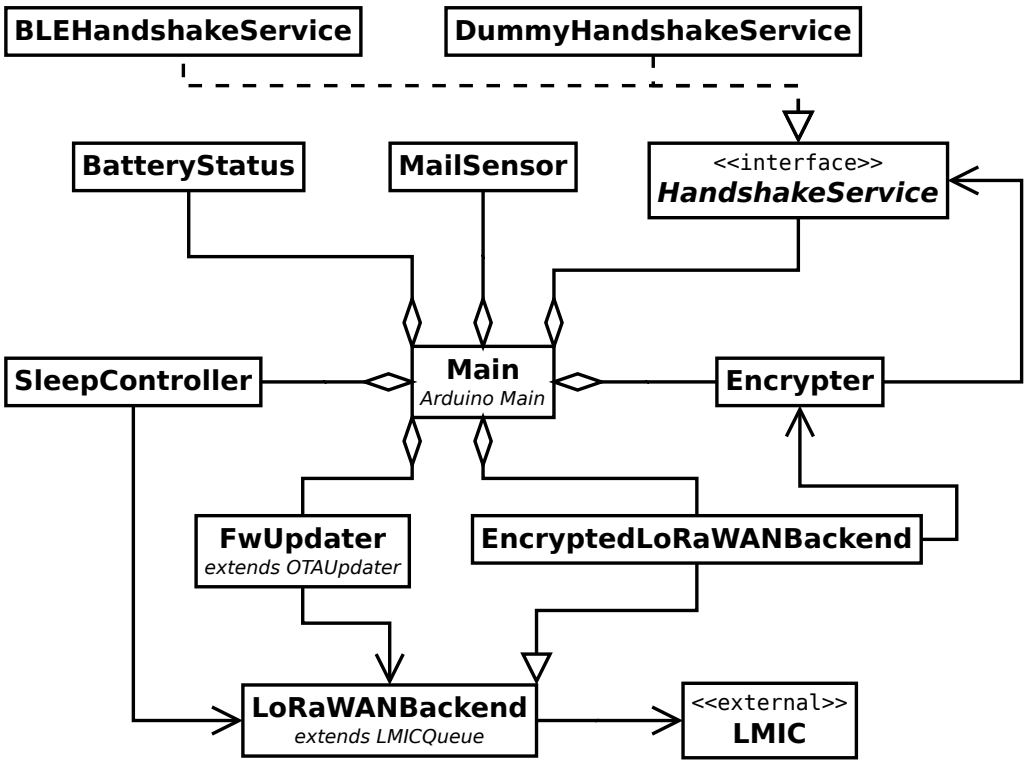

Abbildung 40: Abhängigkeiten der Komponenten des Smarten Briefkastens

MailSensor Die Aufgabe dieser Komponente ist es, den Sensor anzusteuern, der erkennt, ob sich Post im Briefkasten befindet. Dabei wird eine Methode bereitgestellt, mit der dieser Status bei Bedarf abgefragt wird.

BatteryStatus Mit dieser Klasse kann der Batterie-Status des Mikrocontrollers abgefragt werden. Dabei wird der Status als Prozentwert ermittelt und zurückgegeben.

LoRaWANBackend Die LoRaWANBackend-Komponente kapselt die Verwendung der externen LMIC-Bibliothek, um die Verwendung einfacher und sicherer zu machen. Dabei wird die Initialisierung und Verwaltung des LoRaWAN-Status ebenfalls übernommen. Nach außen hin bietet die Komponente nur Funktionen zum Starten der Schnittstelle sowie zum Senden und Empfangen von Daten.

Encrypter Die Aufgabe des Encrypters ist es, Daten für den LoRaWAN-Transfer E2E zu ver- und entschlüsseln. Dazu greift er auf den HandshakeService zu, um von diesem den geheimen Schlüssel zu erhalten.

EncryptedLoRaWANBackend Eine Erweiterung des normalen Backends, welche den Encrypter verwendet, um Daten automatisch zu ver- und entschlüsseln, in Abhängigkeit von dem Port, auf dem sie gesendet und empfangen werden.

FwUpdater Eine einfache Klasse, welche den OTAUpdater aus Abschnitt 5 verwendet, um OTA-Updates empfangen und installieren zu können. Dabei wird die von dem LoRaWANBackend erweiterte LMICQueue verwendet.

HandshakeService Der HandshakeService ist eine Schnittstelle, welche die Funktionen, die zum Durchführen eines Offband-Beitritts benötigt werden, definiert. Außerdem definiert sie Funktionen, um die so ausgehandelten Daten abgreifen zu können.

BLEHandshakeService Diese Komponente implementiert die zuvor definierte HandshakeService Schnittstelle, um das in Abschnitt 4 definierte Protokoll bereitzustellen. Dabei wird BLE als Übertragungskanal verwendet.

DummyHandshakeService Eine minimale Dummy-Implementierung des HandshakeService, die fest in das Gerät einprogrammierte Zugangsdaten zurückgibt, ohne einen echten Handshake durchzuführen. Kann für Debug-Zwecke oder bei Hardware, die kein BLE unterstützt, verwendet werden.

SleepController Mit dem SleepController werden die Schlaf-Funktionen gesteuert. Er kann den Mikrocontroller in einen Schlafmodus versetzen, wenn keine Daten gesendet werden müssen, und ist für den Erhalt des LoRaWAN-Status beim Schlafen verantwortlich.

Main Die Hauptkomponente repräsentiert das primäre Programm, welches alle anderen Komponenten verwendet. Diese werden hier initialisiert, miteinander verbunden und verwendet, um die Funktionen des Briefkastens bereitzustellen. Der Programmablauf des Programms wird in vereinfachter Form in Abbildung 41 dargestellt.

7.2.5 Design des Caching-Servers

Die primäre Aufgabe der Server-Anwendung ist das Zwischenspeichern und Weiterleiten von Nachrichten zwischen dem Briefkasten und der Client-Anwendung. Dieser soll intern eine Datenbank zum Verwalten der Teilnehmer und deren Daten verwenden. Des Weiteren gibt es eine Schnittstelle zum LoRaWAN-Netzwerkserver und zu den Clients. Die beiden Schnittstellen nutzen wiederum die Datenbank, um Daten zwischenzuspeichern und miteinander auszutauschen. Für die Kommunikation via LoRaWAN wird das Postbox-Protokoll aus Unterunterabschnitt 7.2.3 implementiert.

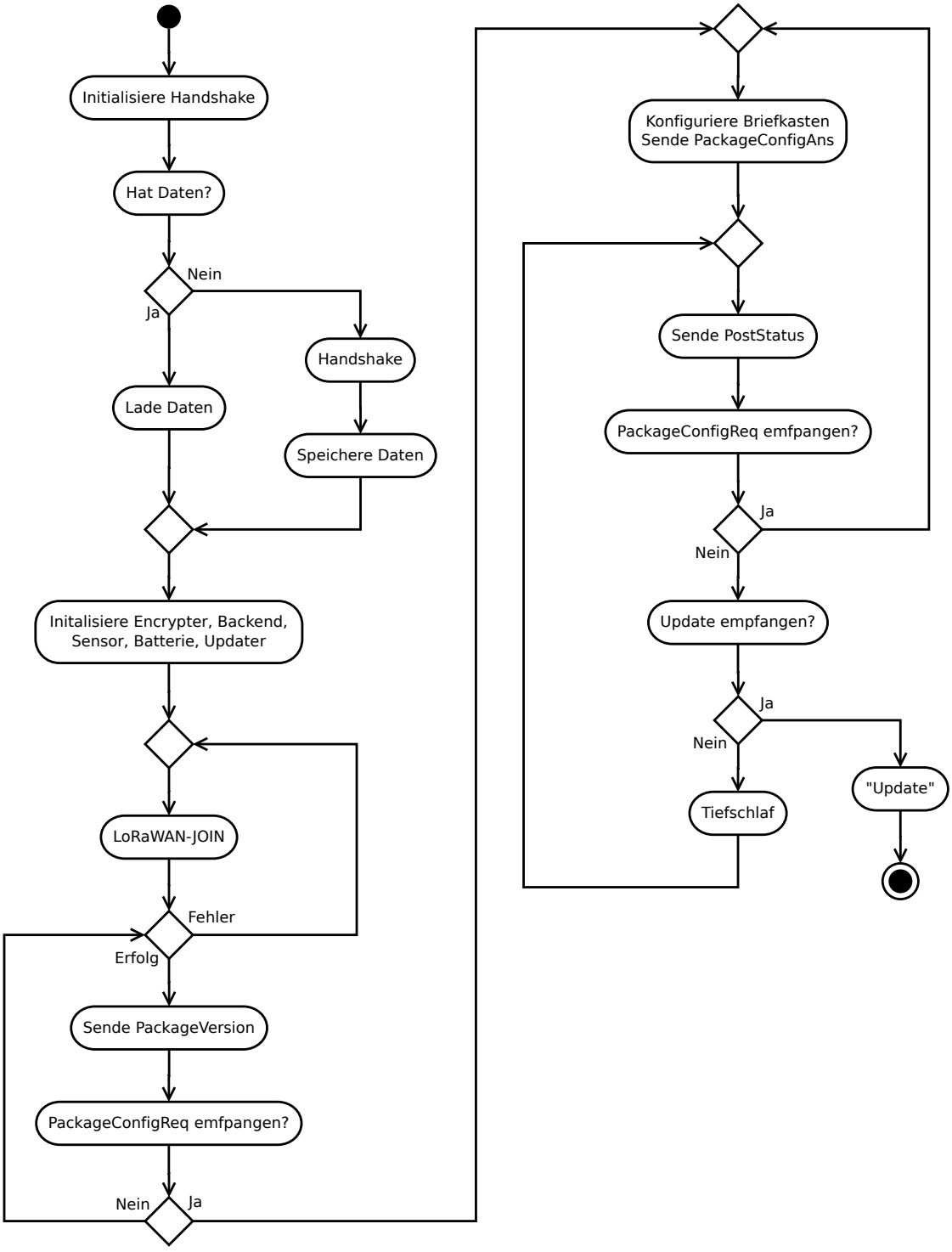

Abbildung 41: Programmablauf des Smarten Briefkastens

Datenbank

Als Erstes wurde dazu die Datenbank entworfen. Dabei konnten zwei Entitäten ausgemacht werden (siehe Abbildung 42). In der Postboxes-Tabelle werden Informationen über die Briefkästen, die dem Server bekannt sind, gespeichert. Zu diesen Informationen gehören die Identität des Briefkastens, dessen Konfiguration und einige LoRaWAN-Parameter. In der Messages-Tabelle werden dann die verschlüsselten Status-Nachrichten der Briefkästen zwischengespeichert, zusammen mit einigen Metadaten, die zur Entschlüsselung benötigt werden, bis der zugehörige Client diese abholt.

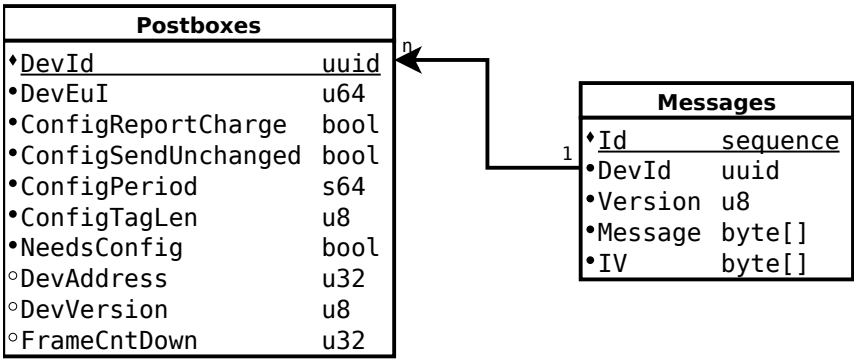

Abbildung 42: Relationaler Datenbankentwurf des Caching-Servers

Client-API

Während die Kommunikation zwischen dem Server und dem Briefkasten durch das Postbox-Protokoll festgelegt ist, sind für den Datenaustausch zwischen Server und Client nur einige Methoden von dem Protokoll definiert.

Betrachtet man diese Kommunikation genauer, so kann festgestellt werden, dass diese immer nach dem Request-Reply-Prinzip erfolgt. Es wird also jedes Mal eine Anfrage vom Client gesendet, auf die der Server antwortet. Für eine solche Struktur bietet es sich an, eine HTTP-REST-Schnittstelle zu entwerfen, da diese genau dem gleichen Prinzip folgt. Ein solcher Entwurf wurde für die Client-API erstellt und in Tabelle 23 ausgeführt. Dabei wurden noch Methoden zum Löschen von Verbindungen hinzugefügt.

Methode	Subpfad	Parameter	Inhalt	Ergebnis
`/v1/postbox[/<devId: string{uuid}>]`				
POST	-/-	-/-	`Setup`	`{"devId": string{uuid}}`
PATCH	-/-	-/-	`Config`	Leer
DELETE	-/-	-/-	-/-	Leer
GET	`/messages`	limit	-/-	`Message[]`
`/v1/handshake[/<devEuI: string{hex}>]`				
POST	-/-	-/-	-/-	`LoRaParams`
DELETE	-/-	-/-	-/-	Leer

Tabelle 23: Definition der Client-HTTP-REST-API

Es werden dabei zwei Klassen angeboten: `postbox` und `handshake`. Mit der `postbox`-Klasse können Verbindungen zu Smarten Briefkästen verwaltet werden. Um eine Verbindung zu erstellen wird die POST-Methode verwendet. Der Inhalt wird im JSON-Format übertragen

und ist in Listing 3 definiert. Als Antwort sendet der Server die `devId` zurück, welche als Pfad-Parameter für alle anderen Methoden dieser Klasse benötigt wird.

```
Setup {
    "devEuI": string,
    "config": {
        "reportCharge": bool,
        "sendUnchanged": bool,
        "period": int,
        "tagLen": int
    }
}
```

Listing 3: JSON-Definition des Setup-Objekts

Soll eine Konfiguration verändert werden, so kann diese mit der PATCH-Methode aktualisiert werden. Dazu wird vom Client ein `Config`-Objekt an den Server gesendet. Dieses wird in Listing 4 definiert. Als Antwort erhält der Client nur den HTTP-Status, um über den Erfolg oder den Fehlschlag der Anfrage informiert zu werden. Um eine Konfiguration wieder zu löschen, kann die DELETE-Methode verwendet werden.

```
Config {
    "reportCharge": bool,
    "sendUnchanged": bool,
    "period": int,
    "tagLen": int
}
```

Listing 4: JSON-Definition des Config-Objekts

Um die Nachrichten, die der Server vom Briefkasten erhalten und zwischengespeichert hat abzurufen, wird die GET-Methode auf dem `/messages`-Subpfad verwendet. Dabei kann mit dem `limit`-Parameter angegeben werden, wie viele Nachrichten maximal zurückgegeben werden sollen. Als Ergebnis sendet der Server eine Liste von `Message`-Objekten, welche in Listing 5 definiert sind.

```
Message {
    "version": int,
    "timestamp": string{datetime}
    "iv": string{base64},
    "payload": string{base64}
}
```

Listing 5: JSON-Definition des Message-Objekts

Mit der zweiten Klasse, der `handshake`-Klasse, kann die Client-Anwendung über den Server Zugangsdaten für das LoRaWAN-Netzwerk anfordern, um diese anschließend an den Briefkasten zu übertragen. Dazu wird die Post-Methode verwendet. Konnte der Server erfolgreich ein Gerät der LoRaWAN-Anwendung hinzufügen und entsprechende Zugangsdaten erlangen, so werden diese in Form eines `LoRaParams`-Objekts an den Client zurückgesendet. Das Objekt ist in Listing 6 definiert. Um ein auf diese Weise erstelltes Gerät wieder zu löschen, kann der Client die DELETE-Methode verwenden.

```
LoRaParams {
    "devName": string,
    "devEuI": string{hex},
    "appKey": string{base64}
}
```

Listing 6: JSON-Definition des LoRaParams-Objekts

7.2.6 Design der Client-Anwendung

Für das Design der Client-Anwendung wurden zuerst Mock-Ups entworfen, die die verschiedenen Benutzeroberflächen, die dem Nutzer angezeigt werden sollen, definieren. Anschließend wurden aus diesen die Struktur der Anwendungslogik abgeleitet.

Das linke Bild in Abbildung 43 zeigt die Übersicht eines Briefkastens. Dort wird der letzte bekannte Status angezeigt und es finden sich Menüs für weitere Aktionen. Wird das Verlaufs-Symbol angeklickt, so öffnet sich der Verlauf, in dem alle empfangenen Statusnachrichten, chronologisch geordnet, aufgelistet werden. Dieser ist in der Mitte der Abbildung zu finden. Über die …-Aktion können weitere Operationen, wie das Editieren oder Löschen eines Briefkastens, gefunden werden. Wird das Hamburger-Menü vom Nutzer geöffnet, so erscheint eine Übersicht mit allen eingespeicherten Briefkästen, wie rechts in der Abbildung dargestellt. Hier können außerdem neue Briefkästen hinzugefügt werden.

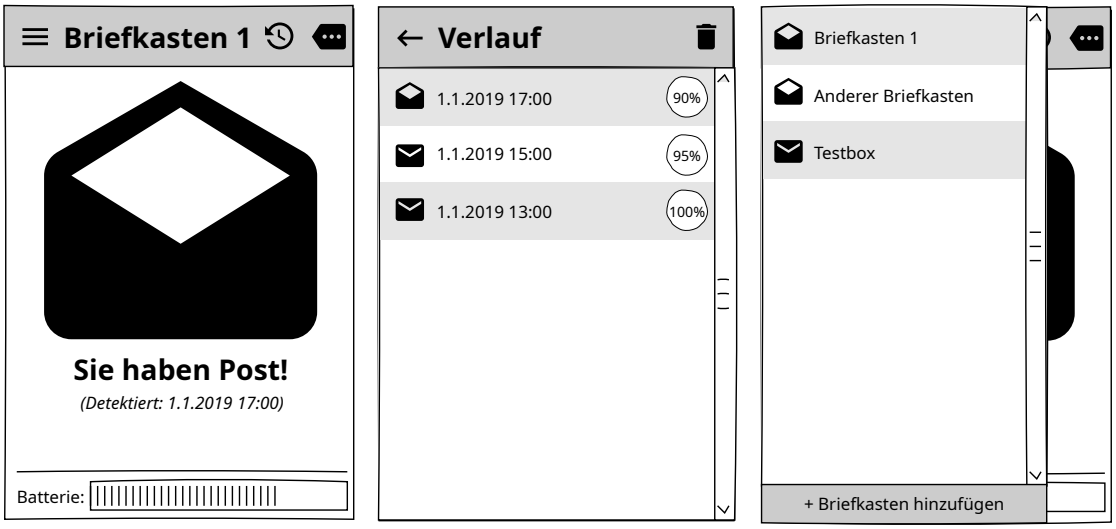

Abbildung 43: Mockup Client: PostboxView, HistoryView, PostboxView (ausgeklappt)

Existiert beim Start der App kein Briefkasten, oder fügt der Nutzer manuell einen hinzu, so öffnet sich die linke Ansicht aus Abbildung 44. Dort kann entweder der Offband-Beitritt durchgeführt werden, indem der Nutzer ein Bluetooth-Gerät aus der Liste auswählt, oder die Daten manuell eingibt, indem auf die mittlere Abbildung gewechselt wird. Wurde ein Gerät ausgewählt, öffnet sich der dort rechts zu sehende Konfigurationsdialog, mit dem das Gerät konfiguriert werden kann. Dieser wird ebenfalls verwendet, wenn der Nutzer die Konfiguration eines existierenden Briefkastens ändern will.

Aus diesen Anforderungen an die Benutzeroberfläche wurde dann die Anwendungslogik entworfen. Dazu wurden mehrere Komponenten ermittelt, die benötigt werden, um die Daten für die Oberfläche aufzubereiten. Diese sind in Abbildung 45 dargestellt, dabei sind die Oberflächen mit Kreisen ebenfalls eingezeichnet, um zu zeigen, wie diese mit den Komponenten zusammenarbeiten. Im Folgenden werden die Komponenten und die Auf-

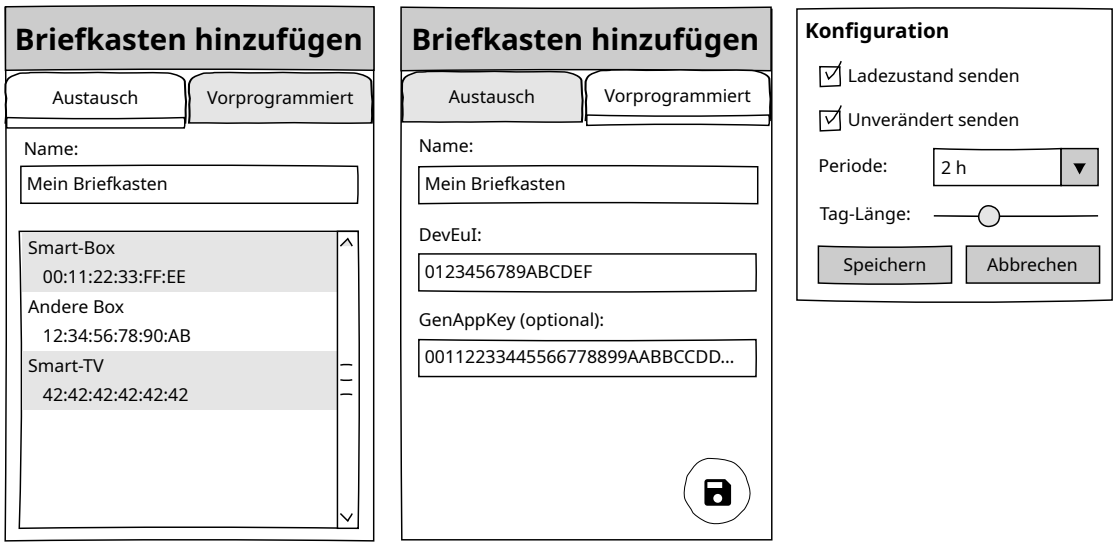

Abbildung 44: Mockup Client: AddHandshakeView, AddPresharedView, ConfigDialog

gaben, die sie übernehmen, beschrieben. Da die Benutzeroberfläche bereits beschrieben wurde, wird auf diese Elemente nicht weiter eingegangen.

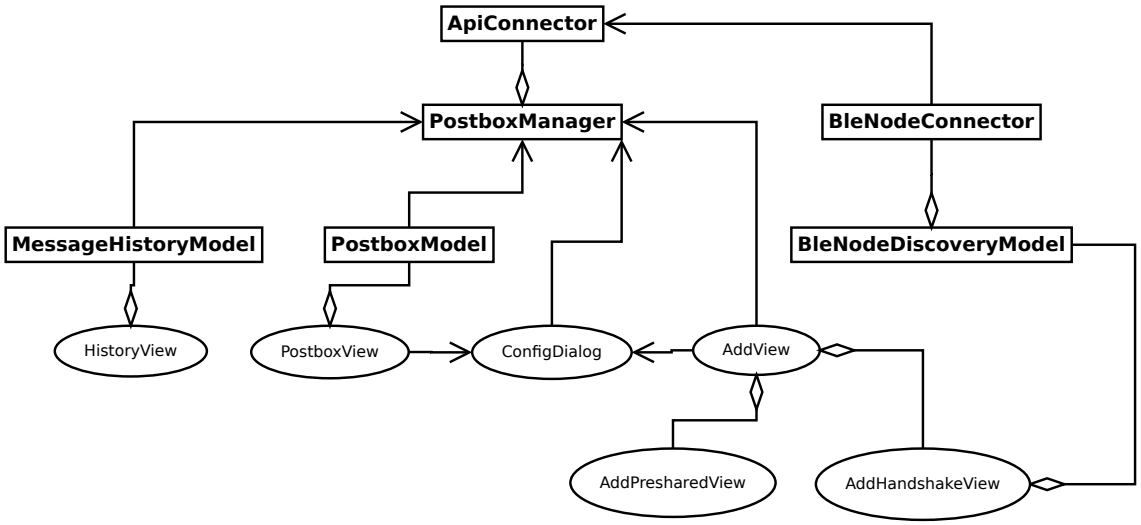

Abbildung 45: Abhängigkeiten der Komponenten der Client-Anwendung

ApiConnector Diese Komponente implementiert die Client-seite der vom Server bereitgestellten REST-Schnittstelle. Sie bietet nach außen hin die von diesem definierten Funktionen an und kapselt die unterliegende HTTP-Kommunikation ab.

PostboxManager Der PostboxManager dient als zentrales Element der Anwendung. Er speichert und verwaltet die Daten der verbundenen Briefkästen, also deren Konfiguration, den geheimen Schlüssel und die empfangenen Nachrichten. Auch die Entschlüsselung der Nachrichten wird von dieser Komponente übernommen. Sie nutzt den ApiConnector, um mit dem Server zu kommunizieren und entsprechend Änderungen an der Konfiguration oder das Empfangen von Nachrichten durchzuführen.

PostboxModel Diese Komponente dient als Brücke zwischen dem PostboxManager und der Benutzeroberfläche, welche die Übersicht des Briefkastens darstellt. Auch für die Übersicht der verschiedenen Briefkästen stellt diese Komponente die Logik bereit.

MessageHistoryModel Eine weitere Brücke, um den Verlauf von Nachrichten, welche der PostboxManager bereitstellt aufbereitet. Sie wird von der Benutzeroberfläche verwendet, um entsprechende Daten anzuzeigen.

BleNodeDiscoveryModel Diese Komponente stellt Funktionen bereit, um Bluetooth-Geräte zu finden und bereitet die Daten so auf, dass diese von der zugehörigen Benutzeroberfläche angezeigt werden können. Will der Nutzer sich mit einem Gerät verbinden, wird dafür die BleNodeConnector Komponente verwendet.

BleNodeConnector Der BleNodeConnector hat die Aufgabe, den Offband-Beitritt via BLE durchzuführen. Dazu verbindet er sich mit dem Endgerät und nutzt den ApiConnector, um das Gerät im LoRaWAN-Netzwerk anzumelden.

7.3 Auswahl der Hardware

Bis zu diesem Zeitpunkt wurde mit zwei verschiedenen Hardware-Plattformen gearbeitet: Dem Adafruit Huzzah32 und dem Adafruit Feather M0. Für die eigentliche Anwendung wurde jedoch entschieden, nur mit dem Adafruit Huzzah32 weiterzuarbeiten. Die Gründe dafür stammen aus den vorherigen Kapiteln: Die entscheidenden zwei Tatsachen sind, dass nur der Huzzah32 BLE unterstützt, und dass beim Feather M0 Multicast-Firmware-Updates nicht zuverlässig funktionieren. Es wurde jedoch weiterhin die Anforderung gestellt, dass der Code für beide Plattformen kompiliert werden kann. Getestet wurde hingegen nur auf dem Adafruit Huzzah32. Bis zu diesem Zeitpunkt wurde in der Arbeit mit zwei verschiedenen Hardware-Plattformen gearbeitet: em Adafruit Huzzah32 und dem Adafruit Feather M0. Für die eigentliche Anwendung wurde jedoch entschieden, nur mit dem Adafruit Huzzah32 weiterzuarbeiten. Die Gründe dafür stammen aus den vorherigen Kapiteln, die entscheidenden zwei Tatsachen sind, dass nur der Huzzah32 BLE unterstützt, und dass beim Feather M0 Multicast-Firmware-Updates nicht zuverlässig funktionieren. Es wurde jedoch weiterhin die Anforderung gestellt, dass der Code für beide Plattformen kompiliert werden kann. Getestet wurde hingegen nur auf dem Adafruit Huzzah32.

Neben dem Mikrocontroller wurde auch entschieden, welcher Sensor für die Detektion von Post verwendet werden soll. Die Auswahl dafür wurde bereits in Unterunterabschnitt 6.5.3 getroffen und entsprechend der IRBreakbeam-Sensor verwendet.

7.4 Implementierung

Die Implementierung der drei Anwendungen erfolgte nach den Anforderungen und Entwürfen aus Unterabschnitt 7.2. Im Folgenden wird dabei kurz auf die verwendeten Techniken und Programmiersprachen sowie Bibliotheken eingegangen. Dabei können Details zu den Bibliotheken in Unterunterabschnitt 2.7.6 gefunden werden.

7.4.1 Implementierung der Server-Anwendung

Es wurde eine Server-Anwendung mit dem C++-Qt-Framework implementiert. Die Anwendung verfügt über keine Benutzeroberfläche, kann aber über die Kommandozeile konfiguriert werden. Die Implementierung des Servers verlief ohne Probleme. Es konnten alle Anforderungen und Designgrundlagen realisiert werden. Als Datenbank wurde SQLite ausgewählt, da diese Datenbank ohne eine dedizierte Datenbank-Anwendung auskommt, sodass der Server alleinstehend lauffähig ist. Es wurde dabei die Abstraktions-

Schnittstelle QtSql verwendet; ein Austausch des Datenbank-Systems ist somit ohne großen Aufwand möglich.

Drittanbieter-Bibliotheken

- Qt-Framework (QtCore, QtNetwork, QtMqtt, QtSql)
- QHttp
- QCliParser

Kommandozeile

--**jwt** <**jwt**> Ein JSON-Web-Token, welches benötigt wird, um in LoRaServer Geräte einer LoRaWAN-Anwendung hinzuzufügen.

--**access-key** <**key**> Ein Passwort oder anderweitiger Schlüssel, der benötigt wird, um sich mit dem Netzwerkserver zu verbinden.

--**localhost** Startet die HTTP-Schnittstelle nur lokal, sodass nur Anwendungen auf dem gleichen System auf diese zugreifen können.

--**port** <**port**> der TCP-Port, über den der Server erreichbar sein soll. Wird diese Option weggelassen, so wird Port 8080 verwendet.

<**app-id**> Die ID der LoRaWAN-Anwendung, mit der der Server arbeiten soll.

7.4.2 Implementierung des Smarten Briefkastens

Die Anwendung für den Smarten Briefkasten wurde mit Arduino implementiert. Dabei wurde der Code aus allen vorherigen Kapiteln in dieser Anwendung zusammengeführt. Somit konnten die E2E-Verschlüsselung, der BLE-Handshake und die OTA-Updates ohne großen Aufwand eingebunden werden. Die übrigen Anforderungen und Designs wurden dann auf Basis der Bibliotheken implementiert.

Ebenfalls implementiert wurde eine Schlaf-Funktion, die den Briefkasten zwischen dem Versand der einzelnen Status-Nachrichten in einen Schlaf-Modus versetzt. Der ESP32 bietet dazu mehrere Modi an, die verschieden effektiv sind [29]. Es wurden sowohl der ”Light Sleep” als auch der ”Deep Sleep” Modus implementiert. Ersterer hält den Arbeitsspeicher des Mikrocontrollers aktiv und ist somit einfach zu implementieren. Der Tiefschlaf hingegen deaktiviert den Arbeitsspeicher, sodass einige Daten, wie z. B. der Status der LoRaWAN-Verbindung, im RTC-Speicher abgelegt werden müssen. Abbildung 46 zeigt, welche Komponenten des ESP32 in diesem Modus noch aktiv sind.

Die Implementierung verlief ohne nennenswerte Probleme. Was jedoch festgestellt werden konnte, ist, dass die fertige Anwendung 99% der primären Code-Partition des ESP32 verbraucht (ca. 33% des kompletten Speichers). Circa 70% davon sind die BLE-Implementierung von Arduino.

7.4.3 Implementierung der Client-Anwendung

Als letzte Anwendung wurde die Client-Anwendung implementiert. Dazu wurde das C++-Framework Qt verwendet. Die Benutzeroberfläche wurde mit QML erstellt. Das Programm wurde so entwickelt, dass die komplette Oberfläche auf allen von Qt unterstützten Plattformen verwendet werden kann. Auch hier konnten alle Anforderungen und Designvorlagen erfüllt werden, ohne dass nennenswerte Probleme auftraten. Einige Screenshots von der realisierten Benutzeroberfläche können im Anhang gefunden werden.

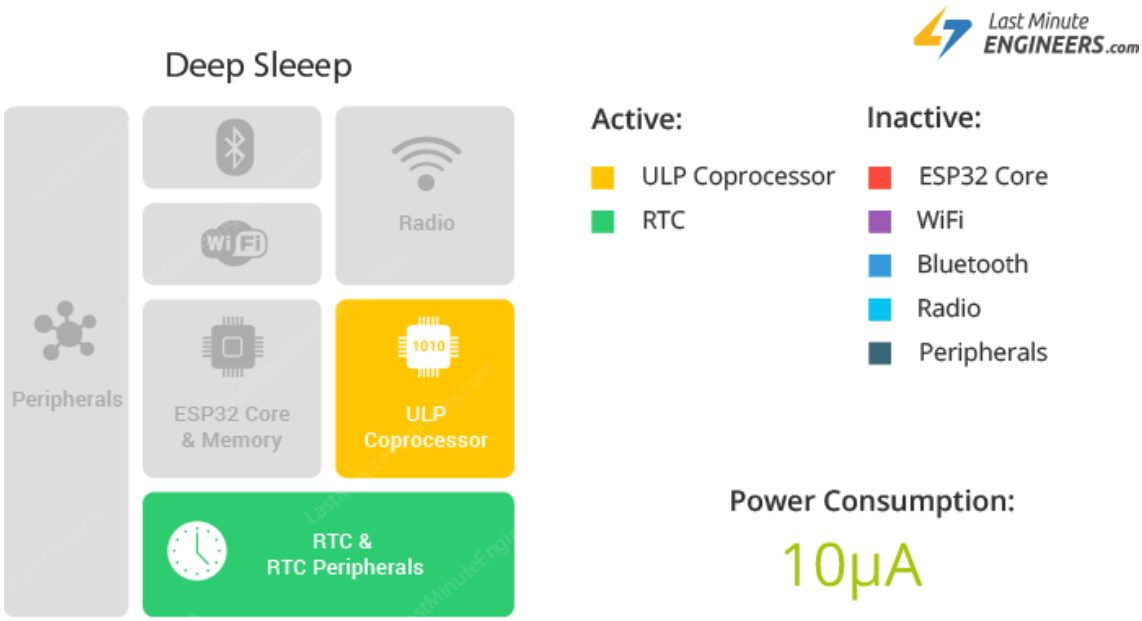

Abbildung 46: Zustand des ESP32 im ”Deep Sleep” Modus [29]

Während die Anwendung prinzipiell plattformunabhängig ist, stimmt dies nicht für die Hintergrund-Benachrichtigungen. Diese wurden nur für Android implementiert, da Qt für diese keine generalisierte Schnittstelle bereitstellt. Es wurden stattdessen die nativen Schnittstellen verwendet. Die Anwendung wird durch diese jede Stunde im Hintergrund gestartet und prüft dann für alle gespeicherten Briefkästen, ob Nachrichten vorliegen. Ist dies der Fall, wird dem Nutzer eine Benachrichtigung angezeigt.

Als Alternative zu dem wiederholten Abfragen von Nachrichten könnte Cloud-Messaging verwendet werden. Bei diesem werden die Dienste des Smartphones verwendet, um Daten an eine Anwendung zu senden, sodass die Anwendung nur dann gestartet wird, wenn neue Daten verfügbar sind. Da die Einrichtung eines solchen Dienstes jedoch nicht trivial ist, wurde dies nicht realisiert.

Drittanbieter-Bibliotheken

- Qt-Framework (QtCore, QtGui, QtQml, QtQuick, QtQuickControls2, QtSvg, QtNetwork, QtBluetooth)
- CryptoPP
- CryptoQQ

7.5 Evaluation

Um die Anwendungen zu evaluieren, wurden diese auf Testgeräten installiert und ausgeführt. Dabei wurde der Adafruit Huzzah32 mit dem IRBreakbeam Sensor in experimentellem Aufbau verwendet, um den Briefkasten zu simulieren (siehe Abbildung 47). Der zugehörige Schaltplan kann in Abbildung 48 gefunden werden. Die Server-Anwendung wurde auf dem Host-System des Autors ausgeführt, die Android-App auf einem Smartphone installiert.

Anschließend wurde die Funktionsfähigkeit der Anwendungen untersucht. Dazu wurden folgende Aufgaben durchgeführt und deren korrekte Funktionsweise überprüft. Es konnten alle diese Tests erfolgreich und ohne Probleme durchgeführt werden.

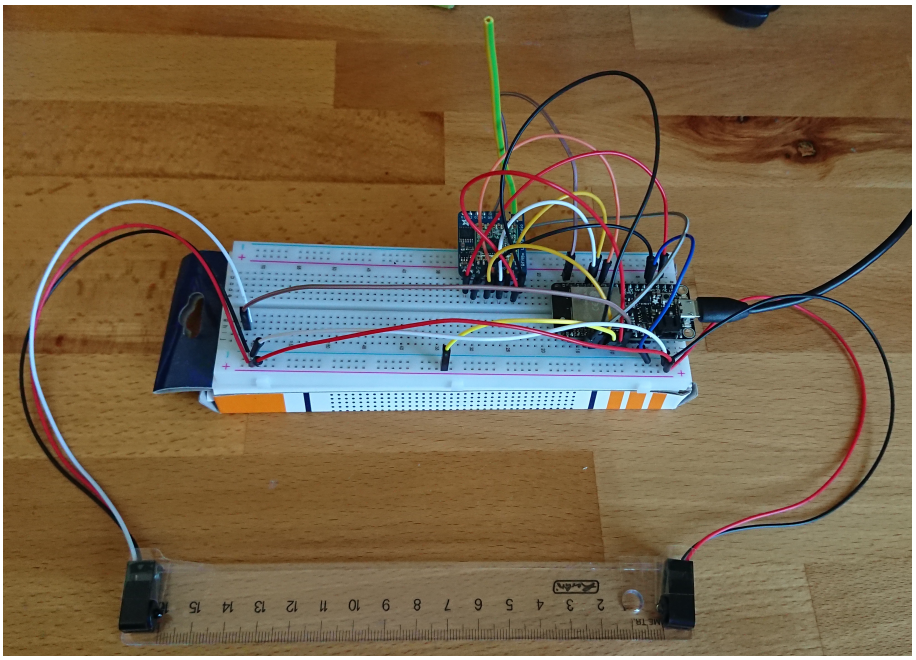

Abbildung 47: Aufbau der Test-Hardware für den Smarten Briefkasten

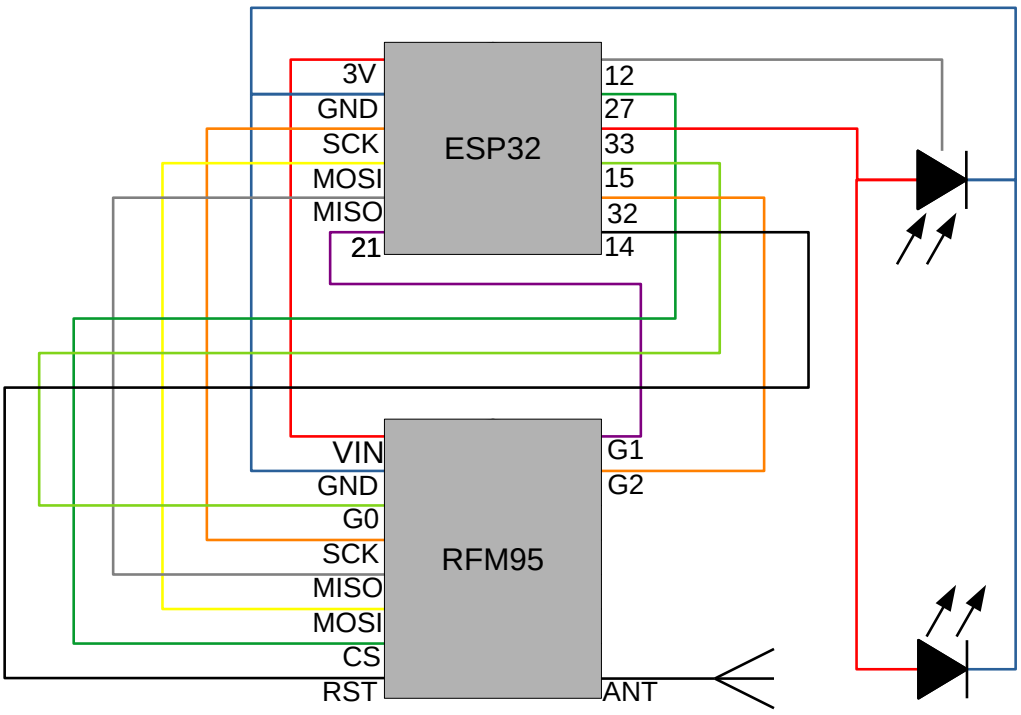

Abbildung 48: Schaltplan des Smarten Briefkastens

1. Handshake
    a) Setup des Briefkastens mit vorprogrammierten Zugangsdaten
    b) Durchführung des BLE-Handshakes, sodass der Briefkasten verschlüsselte LoRaWAN-Nachrichten verschicken kann
    c) Initiale Konfiguration des Briefkastens
    d) Rekonfiguration nach Neustart des Briefkastens
2. Abfrage des Briefkastens
    a) Automatische Übertragung des Status nach vorgegebenem Intervall
    b) Korrekte Verschlüsselung der Nachrichten
    c) Übertragung von korrekten Informationen
    d) Zwischenspeicherung und direkte Abfrage von Nachrichten durch den Client beim Server
    e) Automatische Abfrage von Nachrichten im Hintergrund durch den Client
3. Konfiguration des Briefkastens
    a) Beachtung aller Konfigurations-Parameter
    b) Korrekte Interpretation aller Konfigurations-Parameter
    c) Erfolgreiche Aktualisierung der Konfigurations-Parameter über Neustart des Briefkastens
    d) Erfolgreiche Live-Aktualisierung der Konfigurations-Parameter
4. Löschen des Briefkastens in der Client-Anwendung
5. Korrekte Durchführung von gültigen Firmware-Updates
6. Korrektes Schlaf-Verhalten des Mikrocontrollers
    a) Test ohne Schlaf-Funktionen
    b) Test nur mit dem ”Light Sleep”
    c) Test mit dem ”Deep Sleep”

7.6 Ergebnisse

Mit diesem Kapitel konnte erfolgreich eine Anwendung implementiert werden, mit der automatisiert erkannt werden kann, wenn Post in einen Briefkasten eingeworfen wird. Die erstellten Anwendungen erfüllen alle an sie gestellten Anforderungen. Das Ergebnis ist ein praxistauglicher, smarter Briefkasten, der Daten E2E-verschlüsselt überträgt und in der Lage ist, sicher OTA-Firmware-Updates durchzuführen.

Es konnte ebenfalls verifiziert werden, dass die Protokolle und Bibliotheken, die im Rahmen dieser Arbeit entwickelt wurden, geeignet sind, um mit diesen den Smarten Briefkasten zu realisieren. Es war möglich, die Bibliotheken ohne großen Mehraufwand in den Briefkasten einzubinden.

Was an dieser Stelle aus Zeitgründen nicht weiter untersucht werden konnte, ist der Energieverbrauch der Briefkasten-Anwendung. Zwar wurden die Schlaf-Funktionen des ESP32 verwendet, um den Mikrocontroller zwischen zwei Messungen in Schlaf zu versetzen; wie effektiv diese Maßnahme ist, konnte jedoch nicht untersucht werden.

8 Fazit

Ziel dieser Arbeit war es, einen Smarten Briefkasten zu entwerfen, mit dem E2E-verschlüsselte Datenübertragungen und sichere OTA-Firmware-Updates getestet werden können. Diese Zielsetzung konnte erfüllt werden; es wurde genau ein solcher Briefkasten entworfen und realisiert. Dabei wurden anhand dieser Anwendung mehrere Unterpunkte untersucht, deren Ergebnisse im Folgenden zusammengefasst aufgelistet werden. Neben diesen Erkenntnissen bleiben jedoch noch einige offene Fragestellungen, die als Grundlage für weiterführende Arbeiten dienen können. Auch diese werden im folgenden beschrieben.

Erkenntnisse

1. LoRaWAN verwendet keine vollständige E2E-Verschlüsselung. Es wurde eine eigene, zusätzliche Verschlüsselung entwickelt, um dieses Manko auszugleichen. Dabei konnte ermittelt werden, dass die Verschlüsselung selbst keine negativen Auswirkungen auf den Stromverbrauch des Systems hat. Lediglich bei der Verwendung von authentifizierter Verschlüsselung entsteht durch den notwendigen Versand von zusätzlichen Bits ein höherer Energieverbrauch.
2. Der dynamische Netzwerkbeitritt via JOIN in LoRaWAN ist, wenn korrekt verwendetet, eine sichere Methode. Allerdings werden fest einprogrammierte Schlüssel verwendet, was ein Sicherheitsrisiko darstellt. Um PFS erreichen zu können, wurde ein Protokoll entwickelt, mit dem ein Austausch von Schlüsseln über einen zweiten Kanal ermöglicht wird. Dafür müssen jedoch sowohl das Endgerät als auch das Client-System über einen solchen Kanal verfügen. Es wurde beispielhaft BLE verwendet, um den Kanal zu realisieren.
3. OTA-Firmware-Updates über LPWAN-Netze, wie LoRaWAN eines ist, stellen ganz eigene Herausforderungen an ein solches Protokoll. Es wurde dazu auf Basis von einigen Spezifikationen der LoRa Alliance ein Update-Protokoll entwickelt, welches genau diese Herausforderungen versucht zu meistern. Dabei wurde die Eignung der Protkolle der LoRa Alliance für diesen Zweck verifiziert und ein Protokoll erstellt, welches Updates verlässlich und kryptografisch abgesichert via Multicast verteilen kann. In praktischen Tests konnte gezeigt werden, dass Updates zwar mehrere Stunden an Zeit in Anspruch nehmen, dafür aber verlässlich und mit minimalen Zusatzdaten übertragen werden können.
4. Eine weitere Erkenntnis, die gewonnen wurde, ist, dass Software-Bibliotheken für LoRaWAN aktuell noch viele Fehler und Abweichungen vom Standard aufweisen, besonders, wenn Datenübertragung via Class-B/C durchgeführt werden soll. Es musste an vielen Stellen in der Arbeit eine ausführliche und zeitaufwendige Fehlersuche durchgeführt werden, um diese Probleme zu korrigieren. Besonders in die Realisierung von Multicast-Anwendungen musste viel Zeit investiert werden.
5. Obwohl dies keiner der primären Analysepunkte war, wurden außerdem Techniken untersucht, mit denen Post in einem Briefkasten erkannt werden kann. Dazu wurden mehrere Varianten verglichen und eine Lösung auf Basis einer Infrarot-Lichtschranke entwickelt. Diese zeigte sich als die energiesparendste, am einfachsten umzusetzende und verlässlichste Lösung für diese Aufgabe.

Offene Fragestellungen / Themen für weiterführende Arbeiten

1. Während authentifiziere E2E-Verschlüsselung überaus wichtig für viele Anwendungsfälle ist, wurde festgestellt, dass dafür mehr Daten versendet werden müssen als für die unauthentifizierte Verschlüsselung. Die zugehörigen MACs werden dabei gekürzt, um eine Balance zwischen Sicherheit und Effizienz zu erreichen. Die Auswirkung dieser auf den Energieverbrauch wurde jedoch nicht untersucht. Eine fortführende Arbeit könnte den Einfluss der Länge in Kombination mit verschiedenen LoRaWAN-Parametern, wie z. B. dem Spreading-Faktor, auf den Energieverbrauch untersuchen und ermitteln, wie lang ein MAC sein muss, um ausreichend Sicherheit zu bieten.
2. Das Protokoll für die LoRaWAN-OTA-Updates wurde entwickelt, ohne spezielle Funktionen zum Energiesparen mit einzubauen. Besonders bei Übertragungen via Class-B sowie in den Zuständen des Transfers, in denen das Endgerät warten muss, könnte durch den Einsatz von verschiedenen Schlaf-Modi Energie eingespart werden. Da Firmware-Updates oftmals mehrere Stunden dauern, kann es ein interessantes Thema sein, zu untersuchen, wie viel Energie durch solche Optimierungen eingespart werden kann.
3. Aktuell werden Firmwareupdates unkomprimiert versendet. Eine weiterführende Arbeit könnte untersuchen, wie durch Kompression, Delta-Updates oder andere Methoden die Anzahl von zu übertragenden Datenpaketen minimiert werden kann, sodass Updates schneller (und somit auch energieeffizienter) übertragen werden können. Dabei stellen diese Maßnahmen jeweils neue Herausforderungen für den Einsatz bei leistungsschwachen Mikrocontrollern dar.
4. Bisher wurden nur zwei Sensor-Varianten zum Erkennen von Posteinwurf praktisch getestet. Dabei wurden nur grundlegende Untersuchungen durchgeführt. Eine zukünftige Arbeit könnte dort ansetzen und erweiterte Tests durchführen, um bessere Aussagen über die Leistungsfähigkeit der Sensoren treffen zu können. Außerdem sollten weitere Lösungen untersucht und verglichen werden. Auch der Test mit verschiedenen Typen von Briefkästen bietet sich an.

9 Literaturverzeichnis

[1] Adafruit: *HD-DS25CM-3MM*. Version 1, 2018. https://cdn-shop.adafruit.com/product-files/2167/C2916.pdf. – [Online; Stand 16. Juni 2019]

[2] Adafruit: *Adafruit Feather M0 Radio with LoRa Radio Module*. https://learn.adafruit.com/adafruit-feather-m0-radio-with-lora-radio-module. Version: 2019. – [Online; Stand 16. Juni 2019]

[3] Adafruit: *Adafruit HUZZAH32 - ESP32 Feather*. https://learn.adafruit.com/adafruit-huzzah32-esp32-feather. Version: 2019. – [Online; Stand 16. Juni 2019]

[4] Adafruit: *Adafruit RFM69HCW and RFM9X LoRa Packet Radio Breakouts*. https://learn.adafruit.com/adafruit-rfm69hcw-and-rfm96-rfm95-rfm98-lora-packet-padio-breakouts/overview. Version: 2019. – [Online; Stand 16. Juni 2019]

[5] Adafruit: *Introduction to Bluetooth Low Energy*. https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt. Version: 2019. – [Online; Stand 16. Juni 2019]

[6] Ahelleya, Ashutosh ; Samuel, Sayooj: *CTR Bit Flipping Attack*. https://github.com/ashutosh1206/Crypton/tree/master/Block-Cipher/Attack-CTR-Bit-Flipping. Version: November 2018. – [Online; Stand 16. Juni 2019]

[7] Aras, E. ; Ramachandran, G. S. ; Lawrence, P. ; Hughes, D.: Exploring the Security Vulnerabilities of LoRa. In: *2017 3rd IEEE International Conference on Cybernetics (CYBCONF)*, 2017, S. 1–6

[8] Atmel: *SMART ARM-Based Microcontroller Datasheet*. Version 1, September 2015. https://cdn.sparkfun.com/datasheets/Dev/Arduino/Boards/Atmel-42181-SAM-D21_Datasheet.pdf. – [Online; Stand 16. Juni 2019]

[9] Bernstein, Daniel J.: Curve25519: New Diffie-Hellman Speed Records. In: *Public Key Cryptography - PKC 2006*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2006, 207–228. – [Online; Stand 16. Juni 2019]

[10] Bernstein, Daniel J. ; Duif, Niels ; Lange, Tanja ; Schwabe, Peter ; Yang, Bo-Yin: High-speed high-security signatures. In: *Journal of Cryptographic Engineering* 2 (2012), September, Nr. 2, 77–89. https://ed25519.cr.yp.to/ed25519-20110926.pdf. – [Online; Stand 16. Juni 2019]

[11] Bodewes, Maarten: *AES-GCM recommended IV size: Why 12 bytes?* https://crypto.stackexchange.com/a/41610. Version: November 2016. – [Online; Stand 16. Juni 2019]

[12] Bundesnetzagentur: *Allgemeinzuteilung von Frequenzen zur Nutzung durch Funkanwendungen mit geringer Reichweite für nicht näher spezifizierte Anwendungen; Non-specific Short Range Devices (SRD)*. https://www.bundesnetzagentur.de/SharedDocs/Downloads/DE/Sachgebiete/Telekommunikation/Unternehmen_Institutionen/Frequenzen/Allgemeinzuteilungen/2018_05_SRD_pdf.pdf?__blob=publicationFile&v=2. Version: Vfg 5/2018, Mai 2018. – [Online; Stand 16. Juni 2019]

[13] Diffie, W. ; Hellman, M. E.: Privacy and authentication: An introduction to cryptography. In: *Proceedings of the IEEE* 67 (1979), March, Nr. 3, S. 397–427

[14] Dworkin, Morris J. ; Barker, Elaine B. ; Nechvatal, James R. ; Foti, James ; Bassham, Lawrence E. ; Roback, E. ; Jr., James F. D.: *Federal Inf. Process. Stds. (NIST FIPS) - 197*. NIST Pubs, 2001 https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf. – [Online; Stand 16. Juni 2019]

[15] Eastlake, D. ; Hansen, T. ; Huawei ; AT&T Labs ; Internet Engineering Task Force (IETF): *US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)*. 6234, Mai 2011. https://tools.ietf.org/html/rfc6234. – [Online; Stand 16. Juni 2019]

[16] Eckstein, Michael: *So funktioniert LTE NarrowBand-IoT*. https://www.elektronikpraxis.vogel.de/so-funktioniert-lte-narrowband-iot-a-665905/. Version: November 2017. – [Online; Stand 16. Juni 2019]

[17] Espressif: *ESP32-WROOM-32 Datasheet*. Version 2.7, Oktober 2018. https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf. – [Online; Stand 16. Juni 2019]

[18] Ethier, Justin ; Moses ; D.W. ; kingmakerking: *How is 4G LTE encrypted?* https://security.stackexchange.com/questions/21395/how-is-4g-lte-encrypted. Version: Oktober 2012. – [Online; Stand 16. Juni 2019]

[19] Fisher, John ; VISHAY: *Vishay's TSSP4056 Sensor for Fast Proximity Sensing*. Rev. 21-10-15, Oktober 2015. https://www.vishay.com/docs/82741/tssp4056sensor.pdf. – [Online; Stand 16. Juni 2019]

[20] Gallager, Robert G.: Low-density parity-check codes. In: *IRE Trans. Information Theory* 8 (1962), Nr. 1, 21–28. http://www.inference.org.uk/mackay/gallager/papers/ldpc.pdf. – [Online; Stand 16. Juni 2019]

[21] Ghoslya, Sakshama: *All About LoRa and LoRaWAN - LoRa Decoding*. https://www.sghoslya.com/p/lora_9.html. Version: Januar 2017. – [Online; Stand 16. Juni 2019]

[22] Honeywell: *FSG Series Force Sensor*. 008028-2-EN, Februar 2013. https://sensing.honeywell.com/honeywell-sensing-force-sensors-fsg-product-sheet-008028-2-en.pdf. – [Online; Stand 16. Juni 2019]

[23] HopeRF Electronic: *RFM95/96/97/98(W) - Low Power Long Range Transceiver Module Datasheet*. V1.0, 2006. https://cdn.sparkfun.com/assets/learn_tutorials/8/0/4/RFM95_96_97_98W.pdf. – [Online; Stand 16. Juni 2019]

[24] IMST GmbH: *WiMOD iC880A DATASHEET*. Version 1.1, Juni 2018. https://wireless-solutions.de/downloads/Radio-Modules/iC880A/iC880A_Datasheet_V1_1.pdf. – [Online; Stand 16. Juni 2019]

[25] Jongboom, Jan: *Firmware updates over Low-Powered Wide Area Networks*. https://os.mbed.com/blog/entry/firmware-updates-over-lpwan-lora/. Version: März 2018. – [Online; Stand 16. Juni 2019]

[26] Jongboom, Jan ; Stokking, Johan: Enabling firmware updates over LPWANs / arm MBED. Version: März 2018. http://janjongboom.com/downloads/ew2018-paper.pdf. 2018. – Forschungsbericht. – [Online; Stand 16. Juni 2019]

[27] K. Moriarty, Ed. ; Kaliski, B. ; Rusch, A.: *PKCS #5: Password-Based Cryptography Specification Version 2.1*. 8018, Januar 2017. https://tools.ietf.org/html/rfc8018. – [Online; Stand 16. Juni 2019]

[28] Krejčí, R. ; Hujňák, O. ; Švepeš, M.: Security survey of the IoT wireless protocols. In: *2017 25th Telecommunication Forum (TELFOR)*, 2017, S. 1–4

[29] Last Minute Engineers: *Insight Into ESP32 Sleep Modes Their Power Consumption*. https://lastminuteengineers.com/esp32-sleep-modes-power-consumption/. Version: 2019. – [Online; Stand 16. Juni 2019]

[30] Lee, JungWoon ; Hwang, DongYeop ; Park, JiHong ; Kim, Ki-Hyung: Risk analysis and countermeasure for bit-flipping attack in LoRaWAN. In: *2017 International Conference on Information Networking (ICOIN)*, 2017, S. 549–551

[31] Linka, Hendrik: Untersuchung der technischen und kryptografischen Grundlagen des Netzwerkprotokolls LoRaWAN / Hochschule Bonn-Rhein-Sieg. 2018. – Forschungsbericht

[32] LoRa-Alliance: *What is the LoRaWAN Specification?* https://lora-alliance.org/about-lorawan. Version: 2019. – [Online; Stand 16. Juni 2019]

[33] LoRa Alliance, Inc.: *LoRa Alliance Enhances LoRaWAN Protocol with New Specifications to Support Firmware Updates Over the Air*. https://lora-alliance.org/in-the-news/lora-alliancetm-enhances-lorawantm-protocol-new-specifications-support-firmware-updates. Version: Oktober 2018. – [Online; Stand 16. Juni 2019]

[34] LoRa Alliance, Inc.: *LoRaWAN™ Application Layer Clock Synchronization Specification*. v1.0.0, September 2018. https://lora-alliance.org/resource-hub/lorawan-application-layer-clock-synchronization-specification-v100. – [Online; Stand 16. Juni 2019]

[35] LoRa Alliance, Inc.: *LoRaWAN™ Fragmented Data Block Transport Specification*. v1.0.0, September 2018. https://lora-alliance.org/resource-hub/lorawan-fragmented-data-block-transport-specification-v100. – [Online; Stand 16. Juni 2019]

[36] LoRa Alliance, Inc.: *LoRaWAN™ Regional Parameters*. v1.0.3 revA, Juli 2018. https://lora-alliance.org/resource-hub/lorawan-regional-parameters-v103reva. – [Online; Stand 16. Juni 2019]

[37] LoRa Alliance, Inc.: *LoRaWAN™ Remote Multicast Setup Specification*. v1.0.0, September 2018. https://lora-alliance.org/resource-hub/lorawan-remote-multicast-setup-specification-v100. – [Online; Stand 16. Juni 2019]

[38] LoRa Alliance, Inc.: *LoRaWAN™ Specification*. v1.0.3, Juli 2018. https://lora-alliance.org/resource-hub/lorawantm-specification-v103. – [Online; Stand 16. Juni 2019]

[39] LoRaServer, Orne B.: *Multicast - LoRa Server Server documentation*. https://www.loraserver.io/loraserver/features/multicast/. – [Online; Stand 16. Juni 2019]

[40] LowPowerLab: *Current Ranger - Introduction*. https://lowpowerlab.com/guide/currentranger/. – [Online; Stand 16. Juni 2019]

[41] LowPowerLab: *Current Ranger - Low Pass Filter function (LPF)*. https://lowpowerlab.com/guide/currentranger/low-pass-filter-function-lpf/. – [Online; Stand 16. Juni 2019]

[42] LowPowerLab: *Current Ranger - Safety and proper usage*. https://lowpowerlab.com/guide/currentranger/safety-and-proper-usage/. – [Online; Stand 16. Juni 2019]

[43] LowPowerLab: *Current Ranger - Specs and Architecture*. https://lowpowerlab.com/guide/currentranger/specs-architecture/. – [Online; Stand 16. Juni 2019]

[44] Lueth, Knud L.: *State of the IoT 2018: Number of IoT devices now at 7B – Market accelerating*. https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/. Version: August 2018. – [Online; Stand 16. Juni 2019]

[45] Matthews, Kayla: *The Current State of IoT Cybersecurity*. https://www.iotforall.com/current-state-iot-cybersecurity/. Version: Oktober 2018. – [Online; Stand 16. Juni 2019]

[46] McClelland, Calum: *LPWAN - The Benefits of LPWAN Technology vs. Other IoT Connectivity Options*. https://www.leverege.com/blogpost/lpwan-benefits-vs-iot-connectivity-options. Version: April 2017. – [Online; Stand 16. Juni 2019]

[47] McGrew, David A. ; Viega, John: The Security and Performance of the Galois/Counter Mode (GCM) of Operation. In: *Progress in Cryptology - INDOCRYPT 2004*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2005, S. 343–355

[48] Miller, Robert: LoRa Security - Building a Secure LoRa Solution / MWR LABS. Version: März 2016. https://labs.mwrinfosecurity.com/publications/lo/. 2016. – Forschungsbericht. – [Online; Stand 16. Juni 2019]

[49] Nemec, Matus ; Sys, Marek ; Svenda, Petr ; Klinec, Dusan ; Matyas, Vashek: The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli. In: *24th ACM Conference on Computer and Communications Security (CCS'2017)*, ACM, 2017, S. 1631–1648

[50] Ray, Brian: *What Is LoRa? A Technical Breakdown*. https://www.link-labs.com/blog/what-is-lora. Version: Juni 2018. – [Online; Stand 16. Juni 2019]

[51] Ray, Brian: *What Is LoRaWAN? [Technical Breakdown]*. https://www.link-labs.com/blog/what-is-lorawan. Version: Oktober 2018. – [Online; Stand 16. Juni 2019]

[52] Reynders, B. ; Pollin, S.: Chirp spread spectrum as a modulation technique for long range communication. In: *2016 Symposium on Communications and Vehicular Technologies (SCVT)*, 2016, S. 1–5

[53] Semtech: *LoRa Modem Design Guide*. https://www.semtech.com/uploads/documents/LoraDesignGuide_STD.pdf. Version: Juli 2013. – [Online; Stand 16. Juni 2019]

[54] SeungJae Na ; DongYeop Hwang ; WoonSeob Shin ; Ki-Hyung Kim: Scenario and countermeasure for replay attack using join request messages in LoRaWAN. In: *2017 International Conference on Information Networking (ICOIN)*, 2017, S. 718–720

[55] SimpleSoft: *Using SimpleIoTSimulator to simulate LoRaWAN Networks*. https://www.simplesoft.com/SimpleIoTSimulatorForLoraWan.html. Version: 2018. – [Online; Stand 16. Juni 2019]

[56] Sinha, Rashmi S. ; Wei, Yiqiao ; Hwang, Seung-Hoon: A survey on LPWA technology: LoRa and NB-IoT. In: *ICT Express* 3 (2017), Nr. 1, 14–21. http://www.sciencedirect.com/science/article/pii/S2405959517300061. – [Online; Stand 16. Juni 2019]

[57] Strutz, Tilo: Low-Density-Parity-Check-Codes - Eine Einführung / Hochschule für Telekommunikation Leipzig. Version: Juni 2016. http://www1.hft-leipzig.de/strutz/Kanalcodierung/ldpc_tutorial.pdf. 2016. – Forschungsbericht. – [Online; Stand 16. Juni 2019]

[58] Time and Date AS: *Warum gibt es Schaltsekunden?* https://www.timeanddate.de/zeitzonen/schaltsekunden. Version: 2019. – [Online; Stand 16. Juni 2019]

[59] Tomasin, S. ; Zulian, S. ; Vangelista, L.: Security Analysis of LoRaWAN Join Procedure for Internet of Things Networks. In: *2017 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, 2017, S. 1–6

[60] u-blox: *NEO-M8 u-blox M8 concurrent GNSS modules Data Sheet*. R03, August 2016. https://www.u-blox.com/sites/default/files/NEO-M8-FW3_DataSheet_%28UBX-15031086%29.pdf. – [Online; Stand 16. Juni 2019]

[61] Vejlgaard, B. ; Lauridsen, M. ; Nguyen, H. ; Kovacs, I. Z. ; Mogensen, P. ; Sorensen, M.: Coverage and Capacity Analysis of Sigfox, LoRa, GPRS, and NB-IoT. In: *2017 IEEE 85th Vehicular Technology Conference (VTC Spring)*, 2017, S. 1–5

[62] VISHAY: *High Speed Infrared Emitting Diode, 940 nm, GaAlAs, MQW*. Rev. 1.6, Januar 2017. https://www.vishay.com/docs/81931/vslb3940.pdf. – [Online; Stand 16. Juni 2019]

[63] VISHAY: *IR Sensor Module for Reflective Sensor, Light Barrier, and Fast Proximity Applications*. Rev. 1.9, April 2019. https://www.vishay.com/docs/82458/tssp40.pdf. – [Online; Stand 16. Juni 2019]

[64] Wolf, Jack K.: AN INTRODUCTION TO ERROR CORRECTING CODES Part 3 / University of California, San Diego. Version: 2010. http://circuit.ucsd.edu/~yhk/ece154c-spr16/pdfs/ErrorCorrectionIII.pdf. 2010. – Forschungsbericht. – [Online; Stand 16. Juni 2019]

Anhang

Im Folgenden werden einige Anhänge angeführt, auf welche in diesem Dokument verwiesen wird. Dies sind:

- Alle Diagramme zu den Messergebnissen aus Abschnitt 3.
- Die Diagramme zu den Auswertungen des Multicast-Transfers aus Abschnitt 5.
- Screenshots die Client-Anwendung, welche in Abschnitt 7 entwickelt wurde.

Neben diesen können einige weitere Anhänge auf der beiliegenden CD gefunden werden. Dabei handelt es sich um:

- Die unverarbeiteten Messergebnisse aus verschiedenen Kapiteln.
- Die vorbereitende Auswertung dieser Ergebnisse, aus der einige der Grafiken, welche in dieser Arbeit Verwendung finden, erstellt wurden. Auch einige tabellarische Berechnungen können dort gefunden werden.
- Der Quellcode der verschiedenen Testanwendungen, welche im Rahmen dieser Arbeit entwickelt wurden. Dabei wurden verwendete Zugangsdaten und andere geheime Informationen entfernt.
- Der Quellcode des entwickelten Smarten Briefkastens. Auch hier wurden Zugangsdaten entfernt.

Der Quellcode des Smarten Briefkastens kann außerdem noch Online auf dem GitLab-Server des Fachbereichs Informatik der Hochschule Bonn-Rhein-Sieg unter folgender URL gefunden werden: `https://git.fslab.de/fbarz2s/smart-postbox`

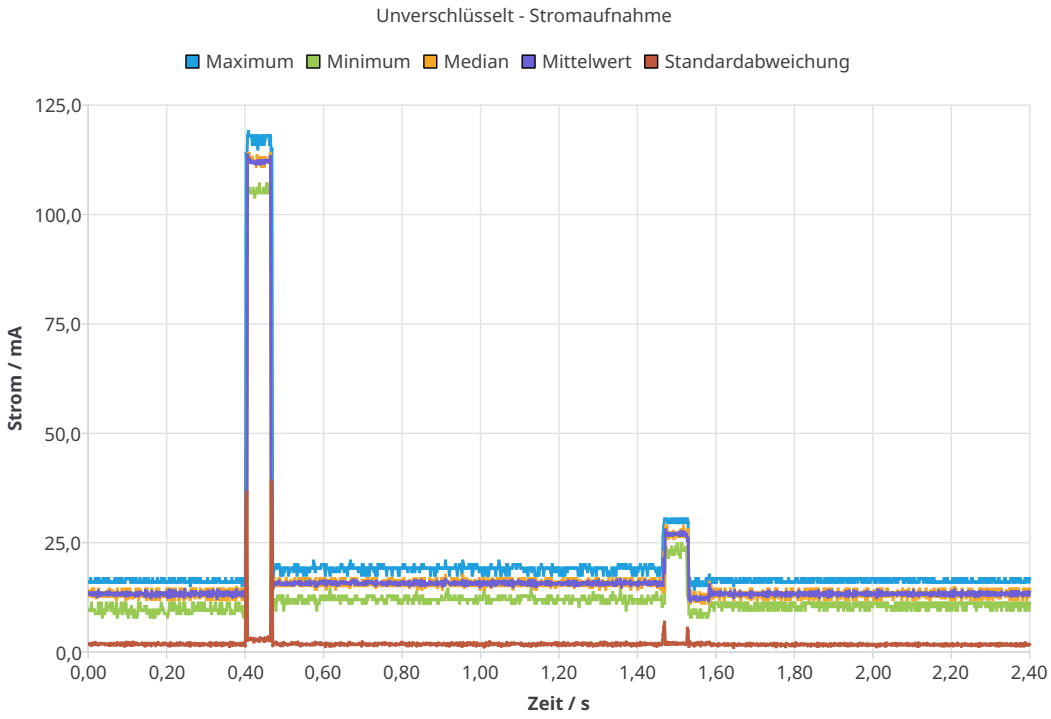

Abbildung 49: Messung Unverschlüsselt

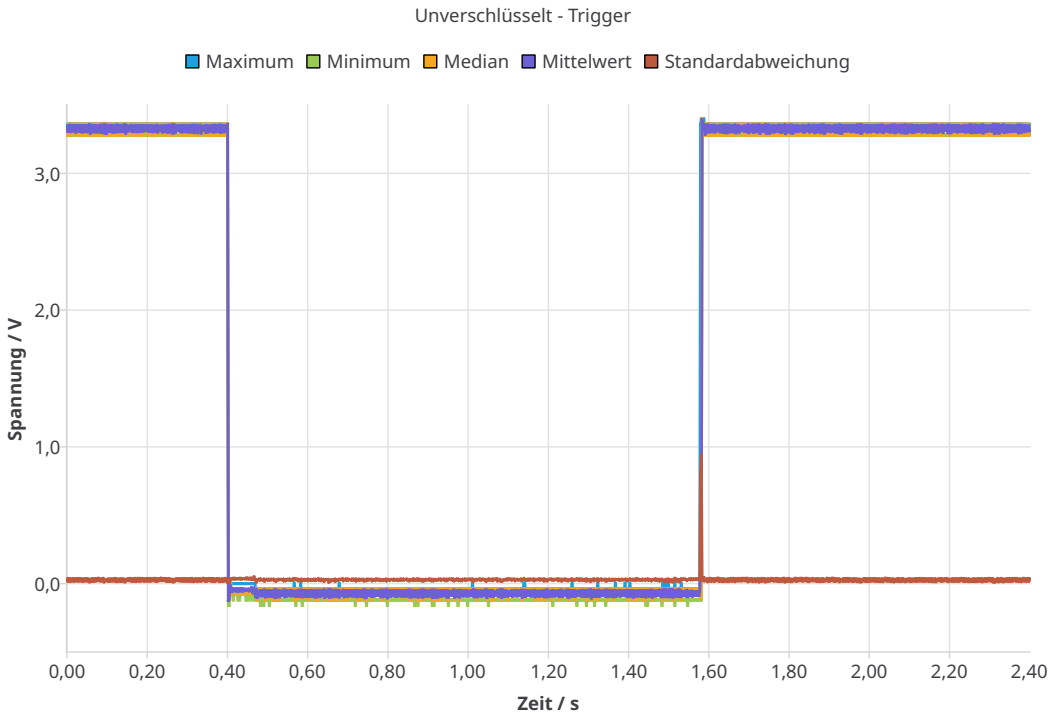

Abbildung 50: Trigger Unverschlüsselt
]fig:anhang:e2e:l0:trigger

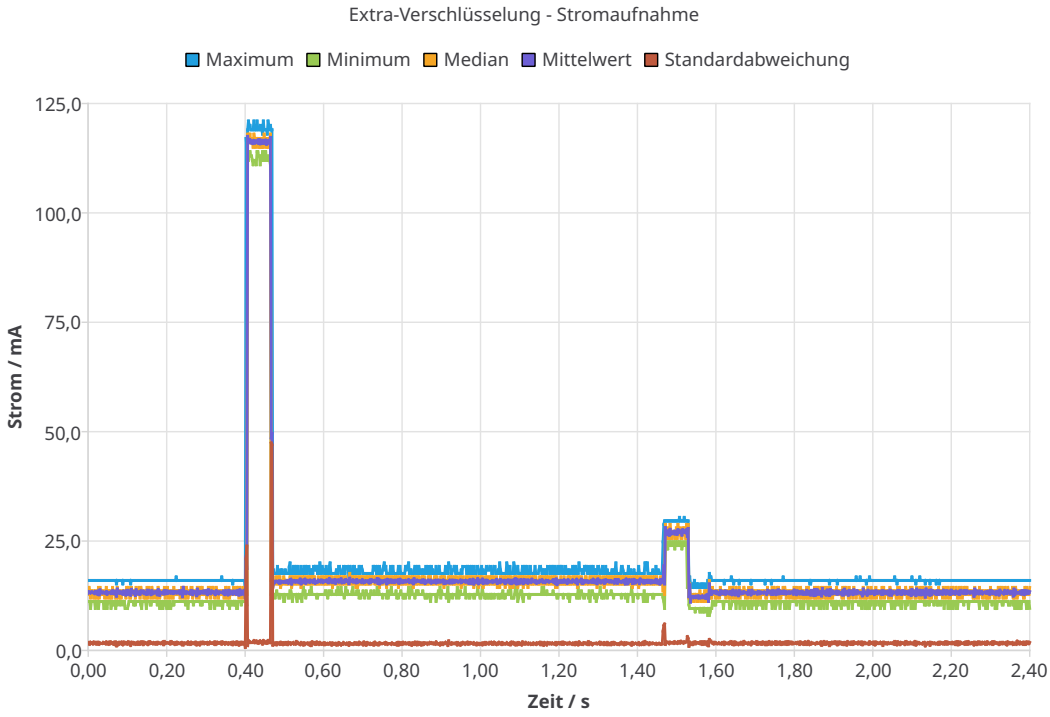

Abbildung 51: Messung Extra-Verschlüsselung

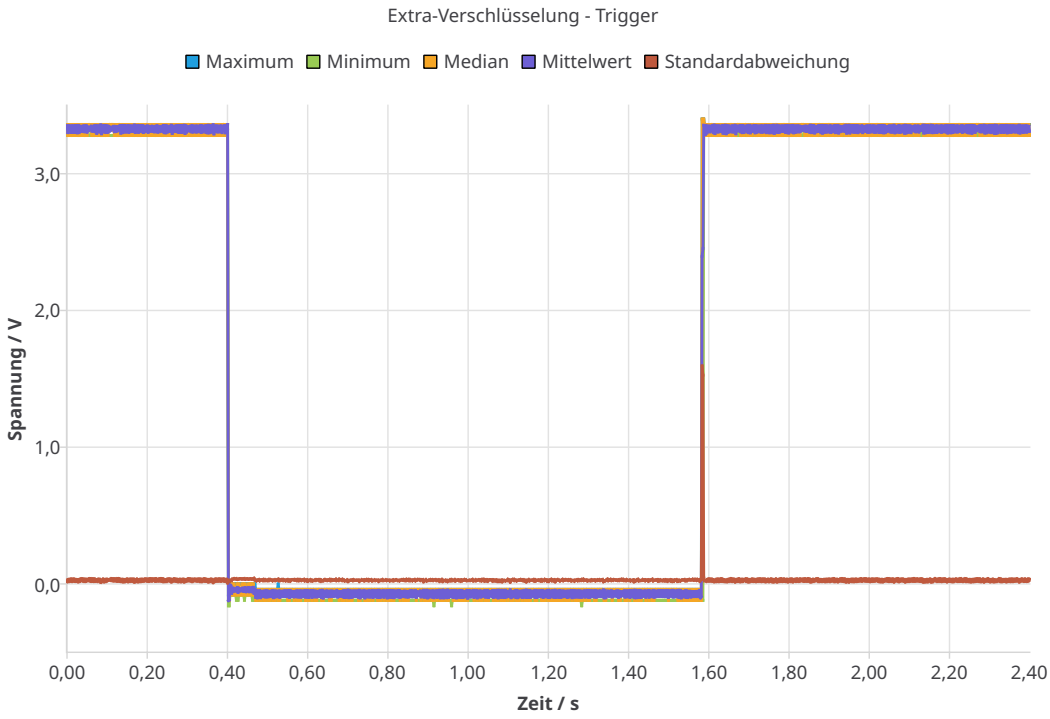

Abbildung 52: Trigger Extra-Verschlüsselung

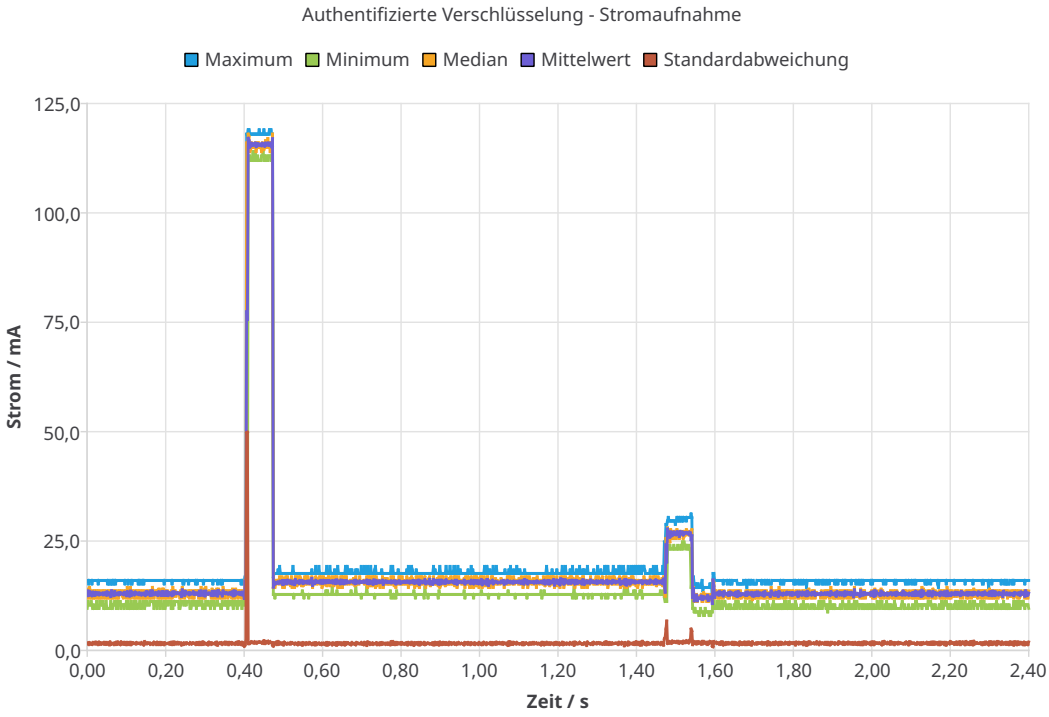

Abbildung 53: Messung authentifizierte Verschlüsselung

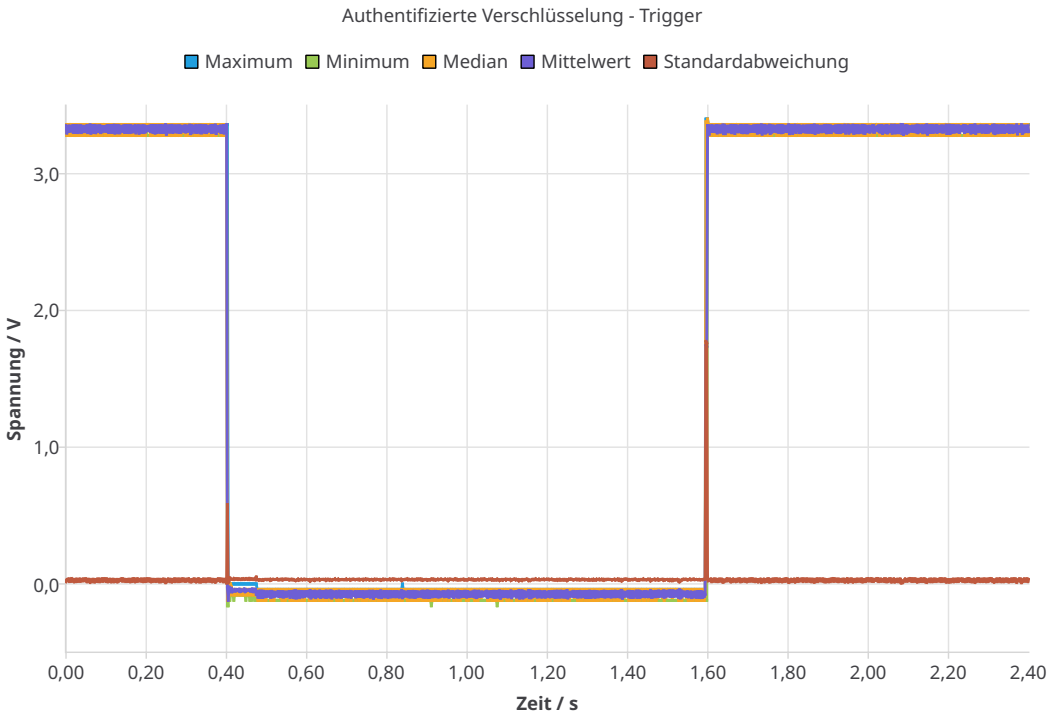

Abbildung 54: Trigger authentifizierte Verschlüsselung

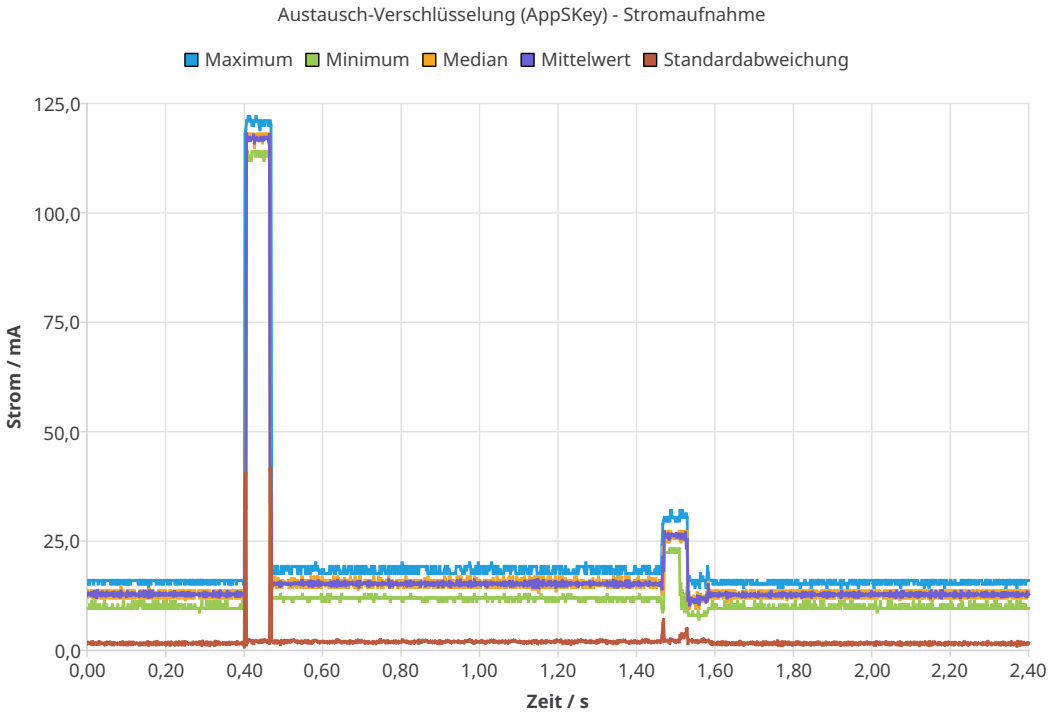

Abbildung 55: Messung Austausch-Verschlüsselung

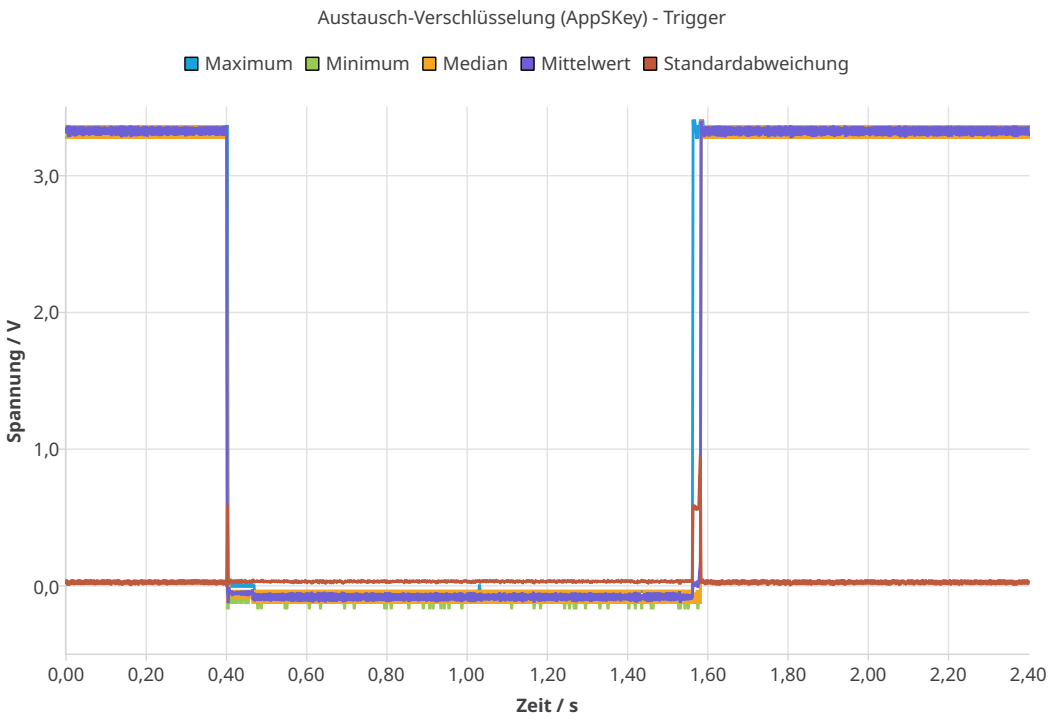

Abbildung 56: Trigger Austausch-Verschlüsselung

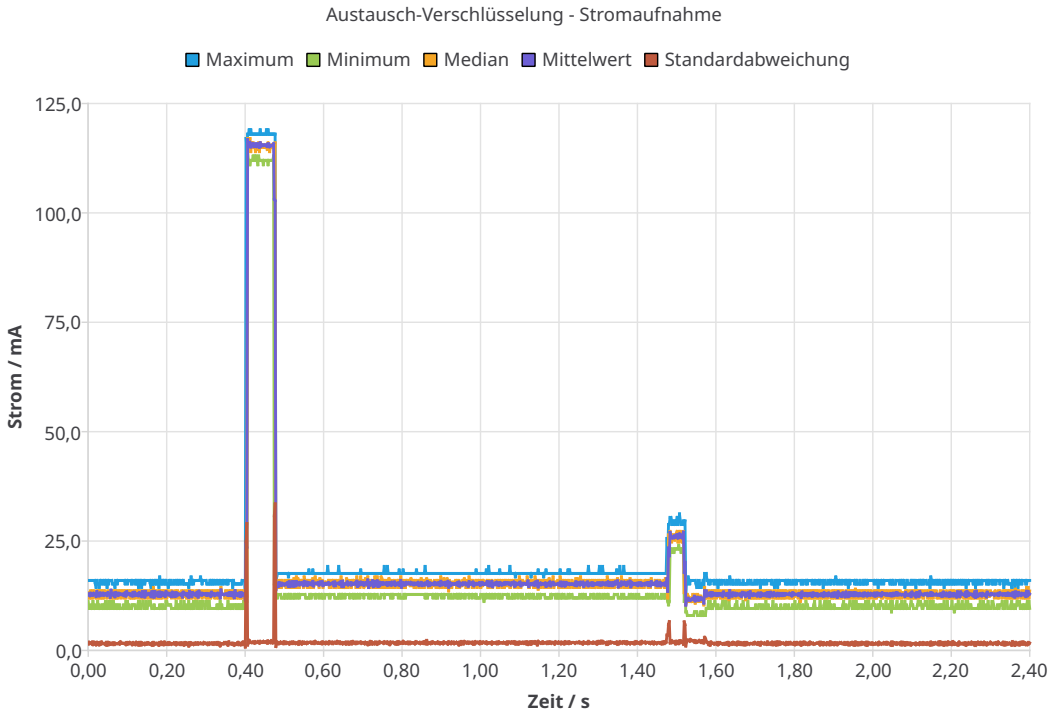

Abbildung 57: Messung `AppSKey`-Transfer für Austausch-Verschlüsselung

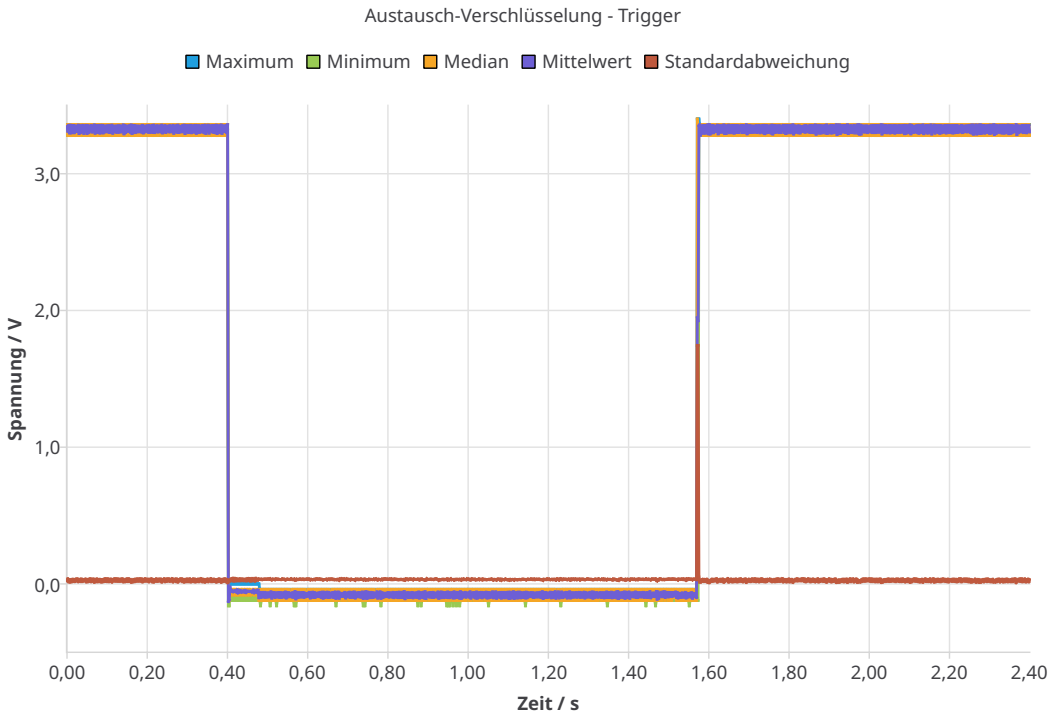

Abbildung 58: Trigger `AppSKey`-Transfer für Austausch-Verschlüsselung

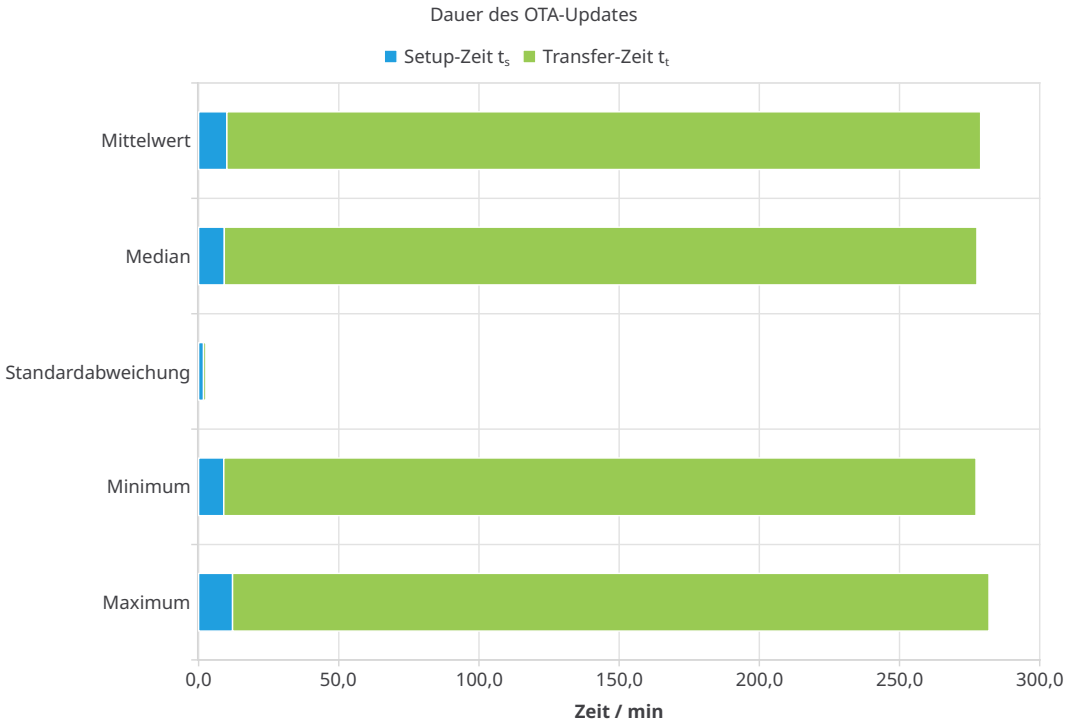

Abbildung 59: Update-Zeit des OTA-Multicast-Transfers

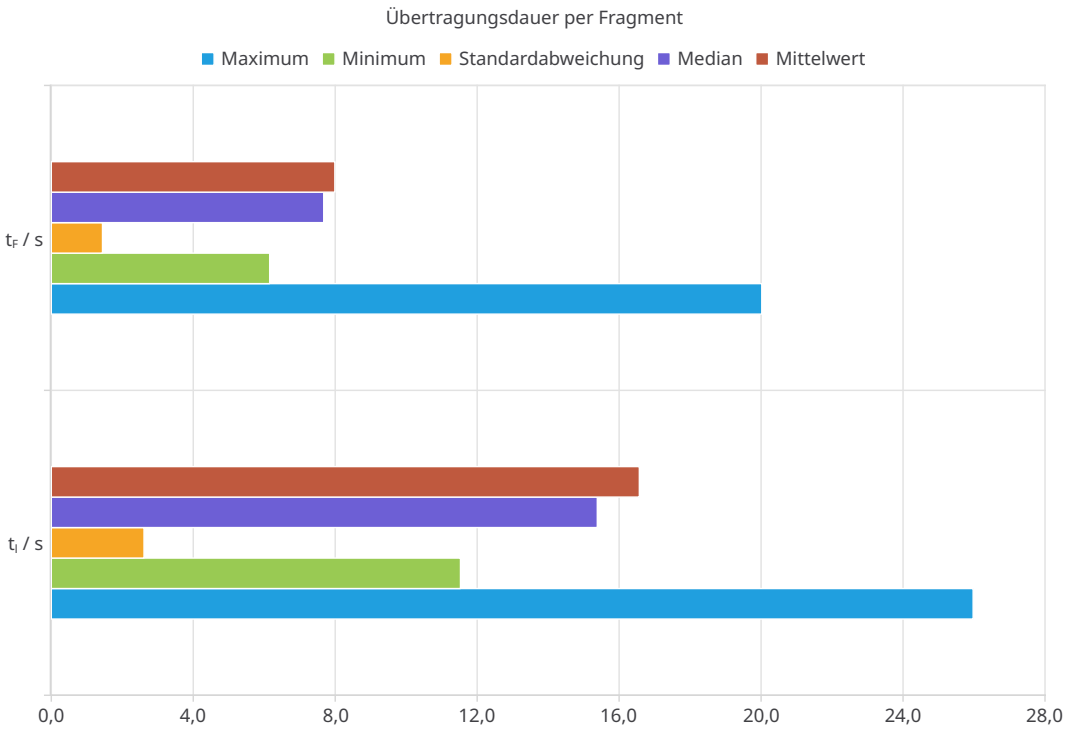

Abbildung 60: Fragment-Transfer-Zeiten des OTA-Multicast-Transfers

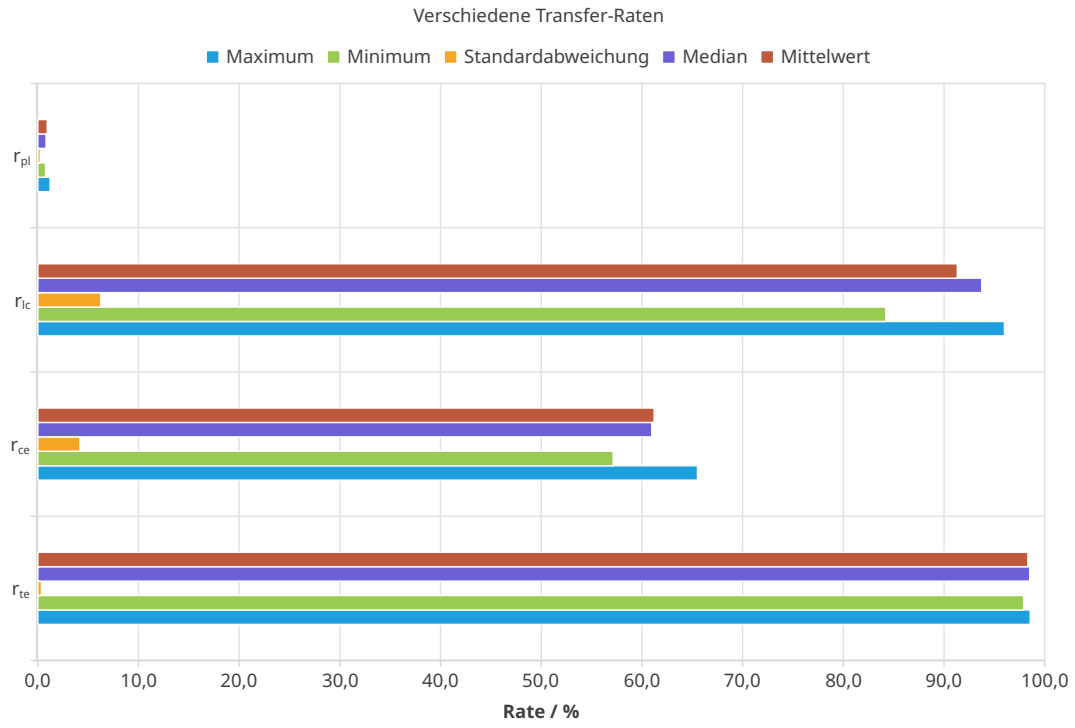

Abbildung 61: Raten des OTA-Multicast-Transfers

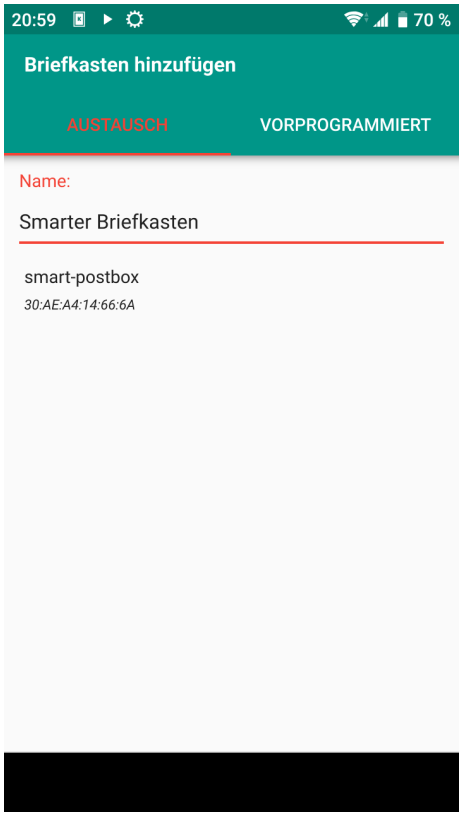

Abbildung 62: Client-App Screenshot 1

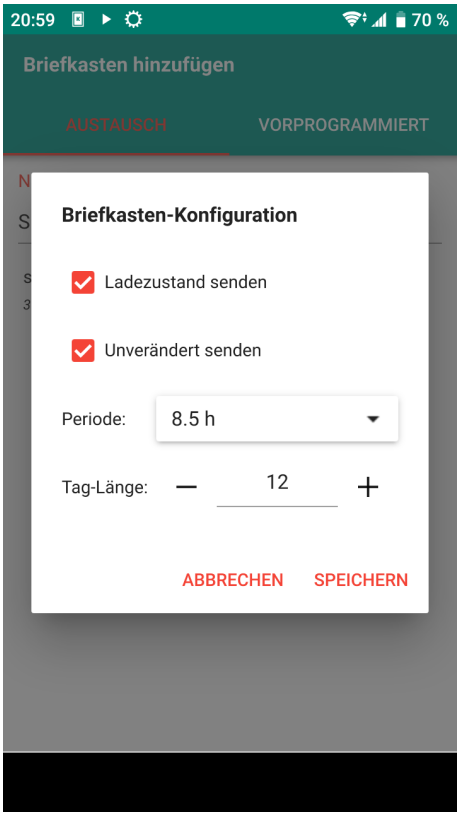

Abbildung 63: Client-App Screenshot 2

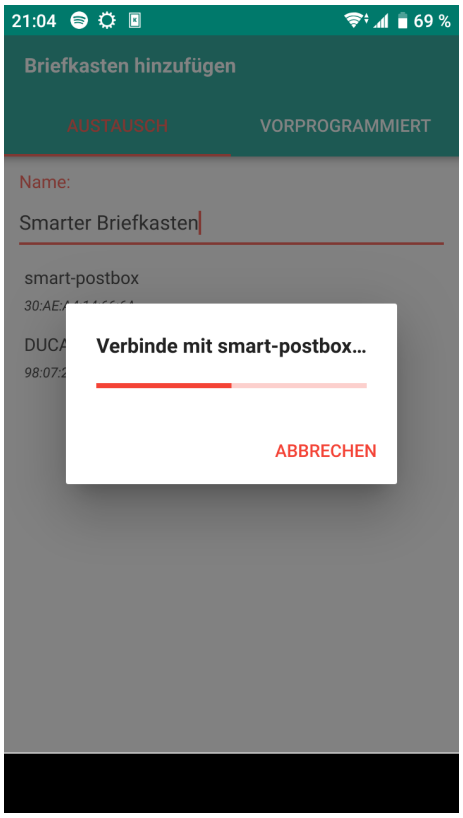

Abbildung 64: Client-App Screenshot 3

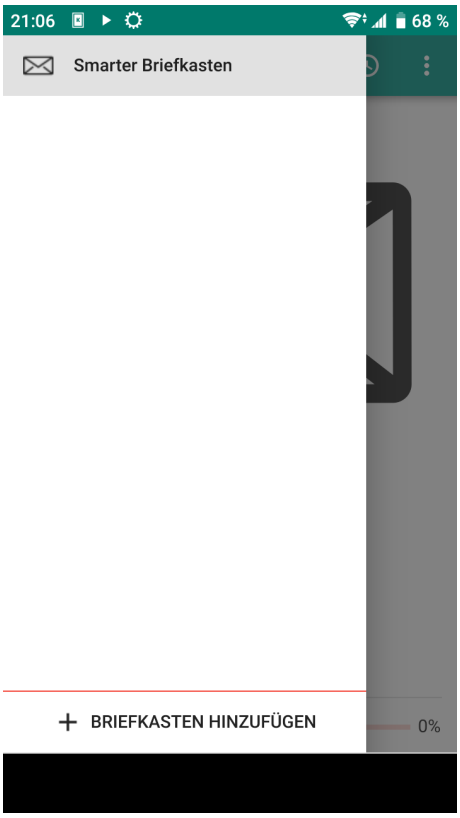

Abbildung 65: Client-App Screenshot 4

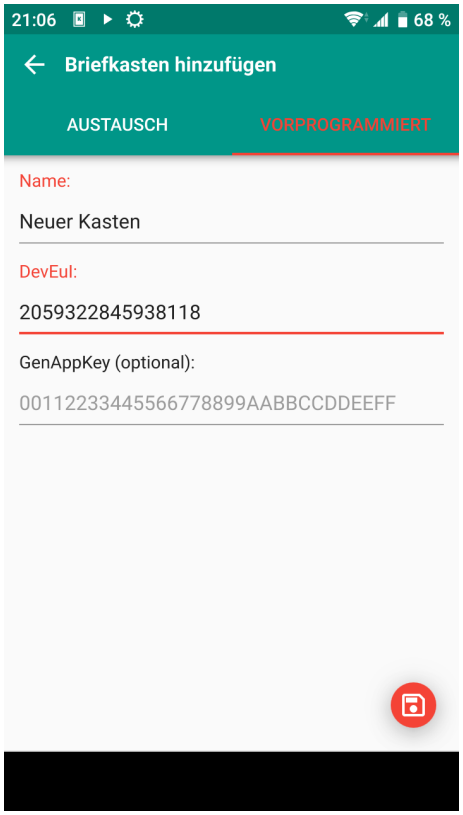

Abbildung 66: Client-App Screenshot 5

Abbildung 67: Client-App Screenshot 6

Abbildung 68: Client-App Screenshot 7

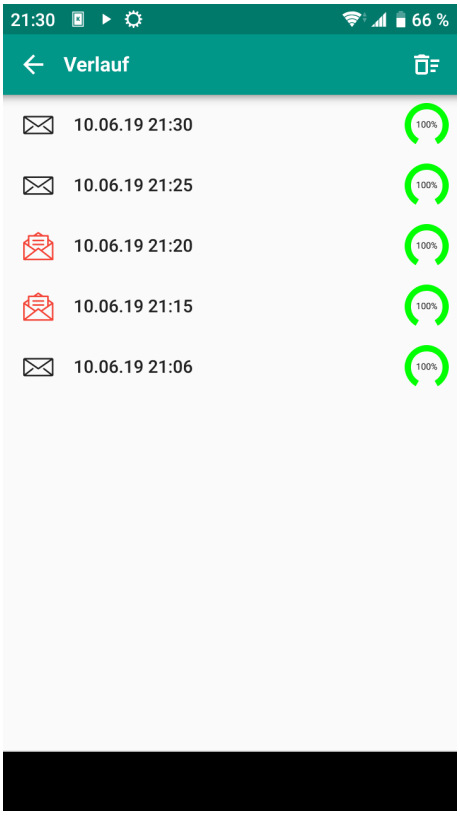

Abbildung 69: Client-App Screenshot 8