HAW HAMBURG

BACHELORTHESIS
Patrick Müller

Evaluation der Anbindung von LoRaWAN-Raumklimasensoren an eine Gebäudeautomatisierungs-steuerung

FAKULTÄT TECHNIK UND INFORMATIK
Department Informations- und Elektrotechnik

FACULTY OF ENGINEERING AND COMPUTER SCIENCE
Department of Information and Electrical Engineering

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Patrick Müller

Evaluation der Anbindung von LoRaWAN-Raumklimasensoren an eine Gebäudeautomatisierungssteuerung

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Informations- und Elektrotechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Jochen Maaß
Zweitgutachter: Prof. Dr. Florian Wenck

Abgegeben am 10. März 2022

Patrick Müller

Thema der Bachelorthesis
Evaluation der Anbindung von LoRaWAN-Raumklimasensoren an eine Gebäudeautomatisierungssteuerung

Stichworte
LoRa, LoRaWAN, LPWAN, SPS, Sensoren, Gateways, IoT, CODESYS, Automatisierung, Gebäudeautomatisierung

Kurzzusammenfassung
Diese Arbeit befasst sich mit der Möglichkeit der Umsetzung eines LoRaWAN-Netzwerkservers auf einer Speicherprogrammierbaren Steuerung, welche keine Internetverbindung besitzt. Dabei wird herstellerunabhängig ein CODESYS-Programm entwickelt, welches im Rahmen eines Testaufbaus die Sensordaten verarbeitet.

Patrick Müller

Title of the paper
Evaluation of the connection of LoRaWAN indoor climate sensors to a building automation control system

Keywords
LoRa, LoRaWAN, LPWAN, PLC, sensors, gateways, IoT, CODESYS, automation, building automation

Abstract
This report adresses the possibility of implementing a LoRaWAN network server on a programmable logic controller that does not have an internet connection. A manufacturer-independent CODESYS program is being developed, which processes the sensor data as part of a test setup.

Inhaltsverzeichnis

Abkürzungsverzeichnis

ABP	Activation by Personalization
AES	Advanced Encryption Standard
AppEUI	Application EUI
AppKey	Application Key
AppSKey	Application Session Key
CBC	Cipher Block Chaining
DevAddr	Device Adress
DevEUI	Device EUI
EUI	Extended Unique Identifier
FB	Funktionsbaustein
FSK	Frequency Shift Keying
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
JSON	JavaScript Object Notation
LNS	LoRaWAN Network Server
LoRa	Long Range
LPWAN	Low Power WAN
LSB	Least Significant Bit
MAC	Media Access Control
MIC	Message Integrity Code
MSB	Most Significant Bit
NB-IoT	Narrowband IoT
NwkSKey	Network Session Key
OTAA	Over-the-Air-Activation
PSK	Phase Shifting Keying
RSSI	Received Signal Strength Indicator
SFTP	SSH File Transfer Protocol
SNR	Signal to Noise Ratio
SPS	Speichprogrammierbare Steuerung
SSH	Secure Socket Shell
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
WAN	Wide Area Network
WLAN	Wireless Local Area Network

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Begriffe wie „Smart Cities“, „Smart Buildings“ oder „Smart Grids“ und die damit verbundenen IoT-Technologien sind ein wesentlicher Bestandteil in aktuellen Debatten und Technologien zum Erreichen einer besseren Energieeffizienz im Gesamtkontext eines klimabewussteren Energiemanagements. Das Internet der Dinge ist ein immer größer werdender Teil der Informationswelt. Im Jahr 2019 waren es mit 10 Milliarden aktiven Geräten erstmals genauso viele wie nicht IoT-Geräte, wobei bis zum Jahr 2025 ein Anstieg auf 30,9 Milliarden erwartet wird (siehe Abbildung 1). Besonders im Segment der Gebäudeautomatisierung, sowohl im privatem als auch im industriellen Bereich, finden sich immer wieder neue Anwendungsfälle für intelligente Geräte, die sich miteinander vernetzen können, um sukzessive immer mehr Funktionen zu erfüllen.

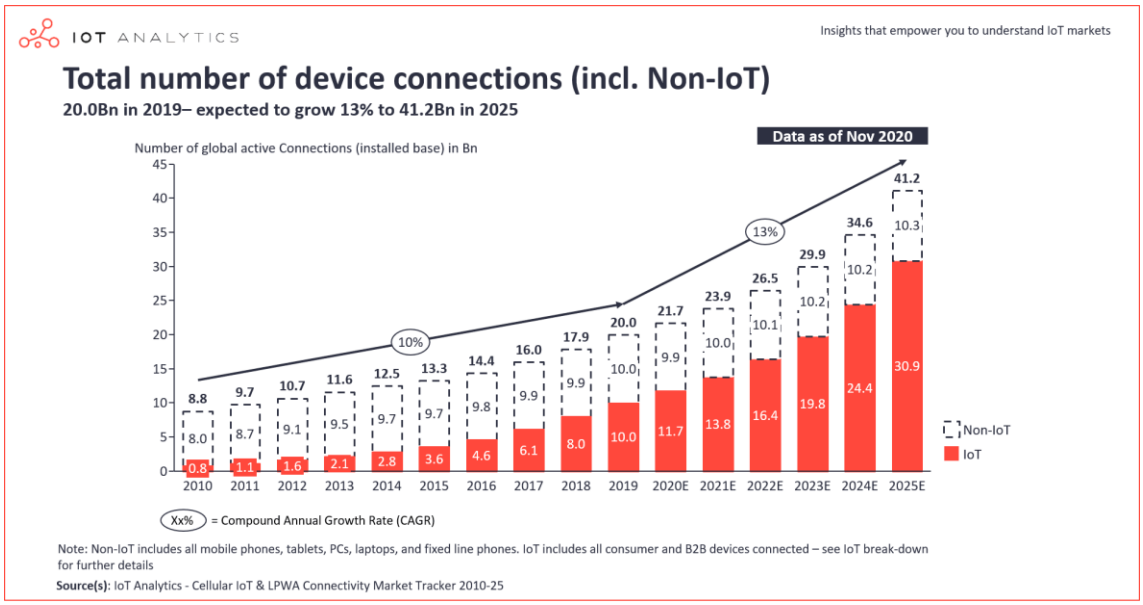


Abbildung 1: Anzahl und Entwicklung von aktiven IoT-Geräten weltweit [1]

Eine der wichtigsten Fähigkeiten von IoT-Geräten ist die Kommunikation. Dabei existieren verschiedene Protokolle mit jeweils unterschiedlichen Eigenschaften und Anwendungsbereichen. Während im Heimbereich zumeist WLAN, Bluetooth, Zigbee oder andere Funkstandards für kurze Distanzen zum Einsatz kommen, müssen Geräte außerhalb des privaten Umfelds auf alternative Technologien zurückgreifen. Eine solche Gruppe von Technologien sind die LPWAN-Übertragungen. Diese haben den Vorteil einer deutlich höheren

Reichweite, allerdings auf Kosten der Bandbreite. In Abbildung 2 sind die LPWAN-Übertragungen im Vergleich zu anderen Funkstandards in Bezug auf diese Eigenschaften dargestellt.

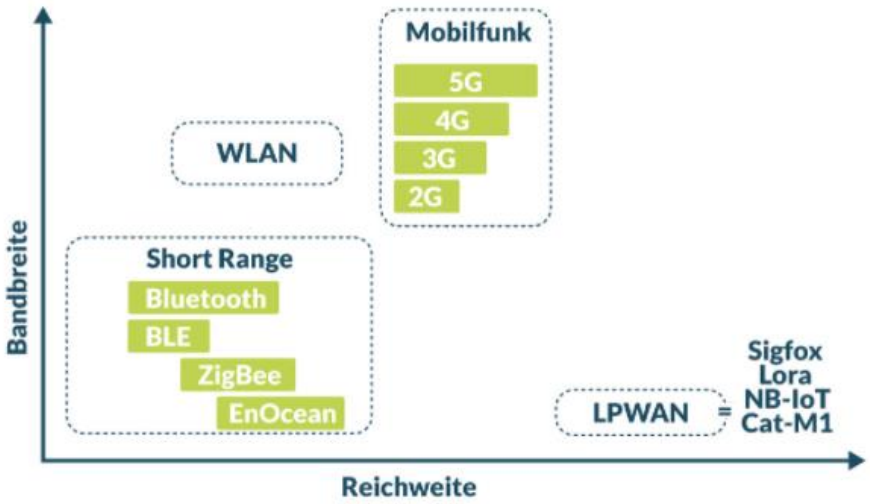


Abbildung 2: Vergleich der Funknetze in Bezug auf Reichweite und Bandbreite [2]

Den LPWAN-Markt beherrschen derzeit im Wesentlichen die Anbieter bzw. Technologien LoRa und NB-IoT, wobei darüber hinaus noch der Vollständigkeit halber Sigfox und LTE-M genannt werden müssen. Die expliziten Anteile und absoluten Zahlen der genannten Technologien am gesamten LPWAN sowie deren erwartetes Wachstum sind in Abbildung 3 dargestellt.

Unterschiede liegen hierbei beispielsweise in der Modulation, den verwendeten Frequenzbändern, den Datenraten, den Reichweiten, der Störungsrobustheit, dem Energieverbrauch und den Kosten. Jede dieser Technologien besitzt aufgrund der technischen Spezifikationen Vor-, aber auch Nachteile, sodass je nach gewünschtem Anwendungsfall entschieden werden sollte. [4][5]

Für den in dieser Arbeit untersuchten Anwendungsfall wird die LoRa-Technologie verwendet.

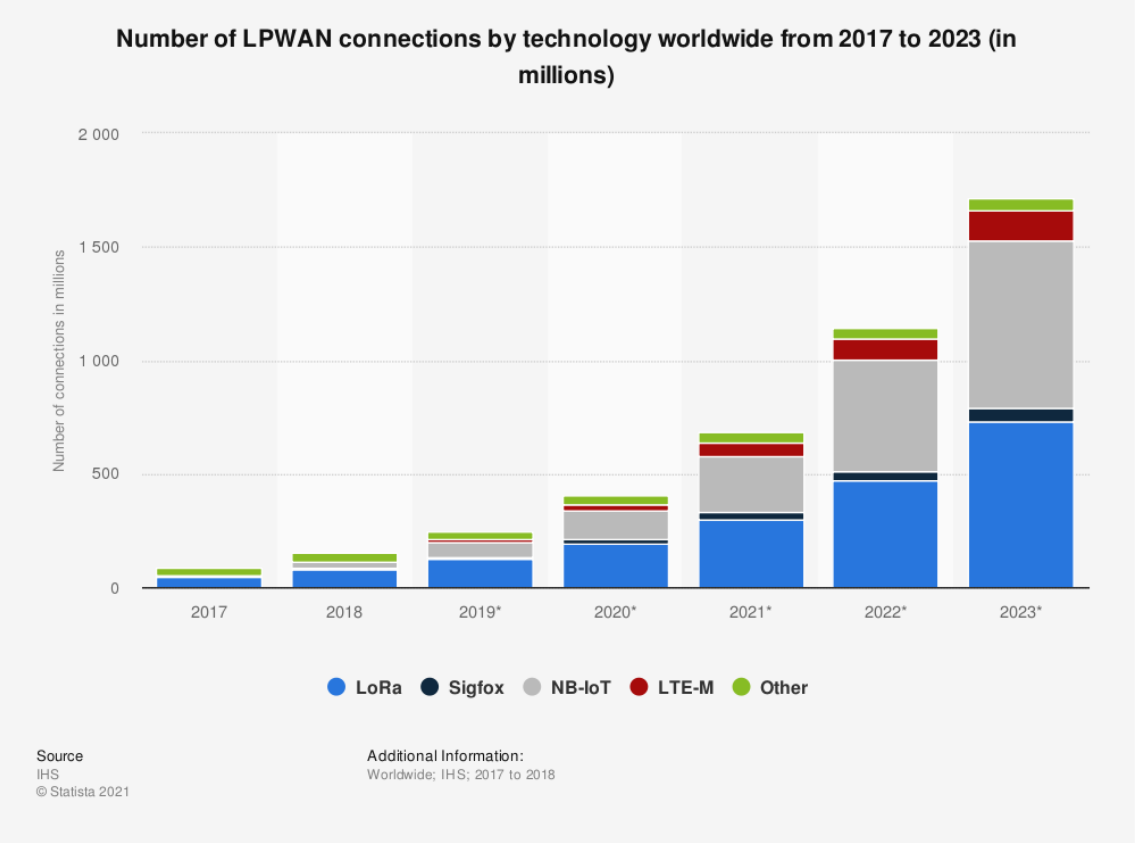


Abbildung 3: Anteil der verschiedenen Technologien am LPWAN [3]

1.1 Gegenstand der Thesis

In dieser Arbeit wird ein konkreter Anwendungsfall in Bezug auf die LoRa-Technologie evaluiert.

1.1.1 Anwendungsfall

Es wird in der vorliegenden Thesis untersucht, inwieweit und mit welchen etwaigen Einschränkungen man LoRa-Raumklimasensoren an eine Speicherprogrammierbare Steuerung der Gebäudeautomatisierung anbinden kann. Die Raumklimasensoren werden in Referenzräumen zur Datenerfassung eingesetzt, um Messpunkte für eine intelligente Heizkreisregelung zu generieren. Detailliertere Angaben zur Aufgabenstellung der genauen Definition des Arbeitsauftrags sind in Kapitel 3 nachzulesen.

1.1.2 Untersuchte Themen

Der Schwerpunkt dieser Thesis liegt in der Evaluierung des konkreten Anwendungsfalls, also in der praktischen Umsetzung innerhalb eines SPS-Programms. Dabei werden zunächst technische Grundlagen erörtert, bevor mit der Untersuchung und Behandlung der Realisierungsmöglichkeiten im Rahmen des Anwendungsfalls fortgefahren wird. Als Folge dieser Überlegungen und der dabei gewonnenen Erkenntnisse kann eine klare Aufgabenstellung definiert und im Folgenden systematisch umgesetzt werden. Nach erfolgter Programmierung wird ein Reichweitentest zur Einordnung der Leistungsfähigkeit der verwendeten Technologie in Bezug auf den Praxiseinsatz durchgeführt. Abschließend wird die erarbeitete Lösung in Bezug auf den Anwendungsfall und enthaltener Stärken und Schwächen bewertet.

2. Technische Grundlagen

In den folgenden Unterkapiteln werden wesentliche Grundlagen erläutert, die als Basis für diese Arbeit dienen. Weitere Details zu den vorgestellten Themen können in den jeweils angegebenen Quellen nachgeschlagen werden.

2.1 LoRa

In diesem Abschnitt wird zunächst die Funktechnologie LoRa mit den zugehörigen Eigenschaften beschrieben. Die Angaben stammen sinngemäß aus [7],[8],[9],[12]. Die Begriffe LoRa und LoRaWAN sind voneinander zu unterscheiden. Während LoRa lediglich bei der Funkübertragung vom Sensor zum Gateway verwendet wird (siehe auch 2.1.1), so beschreibt der Begriff LoRaWAN die gesamte Netzwerkarchitektur (siehe Abschnitt 2.2).

Abbildung 4: Logo der LoRa-Funktechnologie [6]

2.1.1 Was ist LoRa?

Die Funktechnologie LoRa ist eine von der Firma Semtech entwickelte LPWAN-Technologie, die eine drahtlose Kommunikation zwischen zwei Partnern ermöglicht. Es handelt sich hierbei um ein Frequenz-Modulationsverfahren, welches die physikalische Datenübertragung übernimmt und somit im OSI-Layer 1 angesiedelt werden kann. Das ISO/OSI-Referenzschichtenmodell hat das Ziel, Netzwerkarchitekturen zu vereinheitlichen und für jede Schicht klar definierte Aufgaben vorzugeben. Ein LoRA-Modul wird demnach benutzt, um die physikalische Grundlage eines LoRaWAN-Netzwerkes zu liefern und in einem solchen die Bitübertragungsschicht bzw. den Physical Layer zu übernehmen (siehe Abbildung 5).

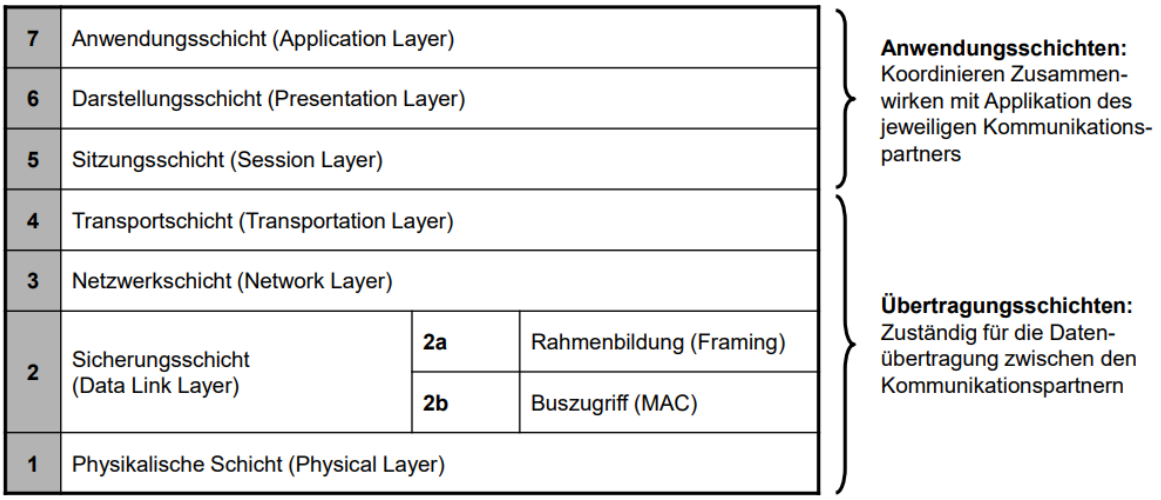


Abbildung 5: Die 7 Schichten des ISO/OSI-Referenzmodells zur Kommunikation [10]

2.1.2 Signalübertragung

Zur Signalübertragung verwendet LoRA die Chirp-Spread-Spectrum-Technologie (kurz: CSS). Diese Modulationstechnik ermöglicht das Übermitteln von Datenmengen im Zuge eines speziellen Übertragungsverfahrens mit sehr hoher Funkreichweite.

Der Begriff CSS setzt sich aus zwei unterschiedlichen Aspekten der Informationstechnik zusammen. Zum einen aus den sogenannten *Chirps*, wodurch sich ein Signal mit zeitlichen Frequenzänderungen beschreiben lässt. Es wird dabei ein Signal gesendet, dessen Frequenz entweder zunimmt (ein positiver Chirp) oder abnimmt (ein negativer Chirp). Beim zweiten Teil, dem *Spread-Spectrum* oder auch Frequenzspreizung, wird ein Vorgang bezeichnet, bei dem ein Signal mit einer bestimmten Bandbreite gezielt umgewandelt wird, um eine größere Bandbreite und somit auch ein breiteres Frequenzspektrum zu nutzen – daher der Begriff Frequenzspreizung.

Wie eine mit diesem Verfahren modulierte Nachricht aussieht, ist in folgender Abbildung 6 dargestellt. Hierbei wird eine LoRa-Nachricht mit vier Präambelsymbolen, zwei Synchronisationssymbolen und vier Nachrichtensymbolen (10, 80, 185, 55) bei einem Spreizfaktor von 8 übertragen.

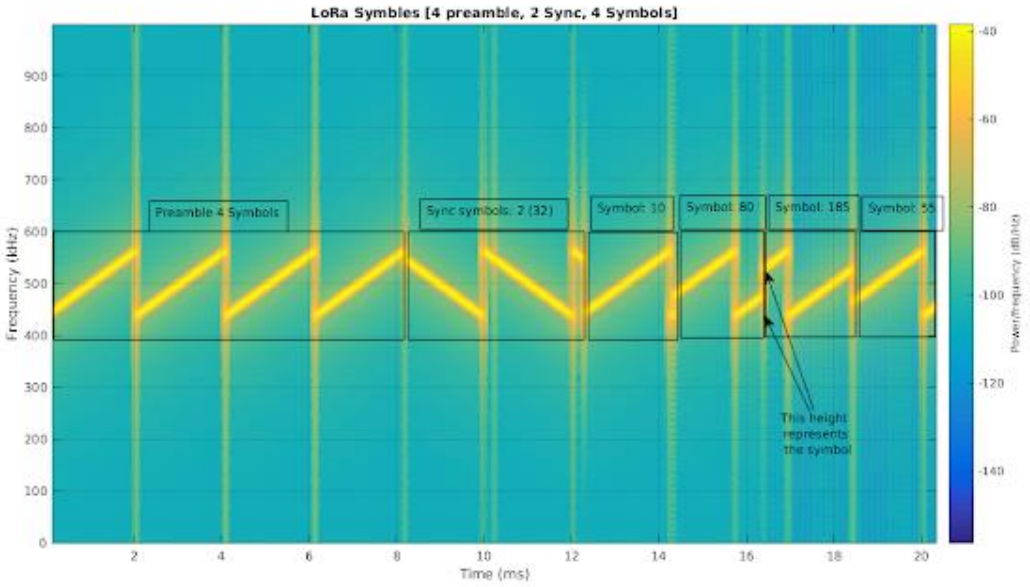


Abbildung 6: Chirp-Modulation einer LoRa-Nachricht [19]

Die größten Vorteile dieser Modulation im Vergleich zu FSK oder PSK, sind die großen Reichweiten und die Robustheit gegenüber Rauschen. Beide Faktoren werden von dem verwendeten Spreizfaktor und der Bandbreite bestimmt. Der Spreizfaktor legt dabei fest, wie lange ein einzelner Chirp dauert, also wie breit dieser „gespreizt" wird. Ein höherer Faktor hat breitere Symbole zur Folge, was sich positiv auf die Übertragungsreichweiten auswirkt. Allerdings führt dies auch zu einer langsameren Datenübertragung, da durch die breiteren Symbole die Übertragungszeit steigt. In LoRa sind Spreizfaktoren von 7 bis 12 definiert.

Die Funkreichweite liegt laut Angaben der Entwickler bei ca. zehn Kilometern im freien Feld und ca. zwei Kilometern in urbaner Umgebung. Die zu erreichenden Datenraten liegen zwischen 0,3 kbit/s und 50 kbit/s [11]. Die Bandbreite ist auf 125 kHz oder 250 kHz festgelegt. Die konkrete Wahl der Parameter ist durch den LoRaWAN-Standard festgelegt.

Die von LoRa verwendeten Frequenzen variieren je nach Region. In Europa wird standardweise auf einem Frequenzband von 863 bis 870 MHz gesendet, wobei die voreingestellten Kanäle gemäß der in folgender Abbildung 7 dargestellten Spezifikation festgelegt werden.

Modulation	Bandwidth [kHz]	Channel Frequency [MHz]	LoRa DR / Bitrate	Nb Channels	Duty cycle
LoRa	125	868.10 868.30 868.50	DR0 to DR5 / 0.3-5 kbps	3	< 1%

Abbildung 7: Default-Einstellungen im EU863-870 Frequenzband [13]

Erwähnenswert hierbei ist, dass es sich bei diesem Spektrum um ein nicht lizenziertes Frequenzband handelt. Somit werden keine Lizenzgebühren bei der Verwendung fällig, allerdings existiert eine zeitliche Sendebeschränkung von unter 1 %.

Die Datenraten (DR) codieren den von LoRa verwendeten Spreizfaktor und Bandbreite, sowie die von den Sensoren verwendeten Bitraten. Diese Zusammenhänge sind in Abbildung 8 dargestellt. Jeder LoRa zertifizierte Sensor in der EU muss mindestens die Datenraten 0 – 5 implementiert haben [13].

Data Rate	Configuration	Indicative physical bit rate [bit/s]
0	LoRa: SF12 / 125 kHz	250
1	LoRa: SF11 / 125 kHz	440
2	LoRa: SF10 / 125 kHz	980
3	LoRa: SF9 / 125 kHz	1760
4	LoRa: SF8 / 125 kHz	3125
5	LoRa: SF7 / 125 kHz	5470
6	LoRa: SF7 / 250 kHz	11000
7	FSK: 50 kbps	50000
8	LR-FHSS[12] CR1/3: 137 kHz BW	162
9	LR-FHSS CR2/3: 137 kHz BW	325
10	LR-FHSS CR1/3: 336 kHz BW	162
11	LR-FHSS CR2/3: 336 kHz BW	325
12..14	RFU	
15	Defined in [TS001][13]	

Abbildung 8: EU863-870: Zusammenhang Datenrate – Spreizfaktor – Bandbreite – Bitrate [13]

Die LoRa-Alliance ist eine offene, gemeinnützige Vereinigung, die sich seit ihrer Gründung im Jahr 2015 zu einer der größten und am schnellsten wachsenden Allianzen im Technologiesektor entwickelt hat. Ziel dieser Allianz ist es, LoRa als LPWAN-Standard zu entwickeln und somit großvolumige IoT-Abdeckungen zu erreichen. In der LoRa-Alliance befinden sich aktuell mehrere hundert

Mitglieder, darunter namhafte Unternehmen wie Semtech, IBM, Cisco, KPN, Swisscom, NEC und Microsoft [14].

2.2 LoRaWAN

In diesem Abschnitt wird der Begriff LoRaWAN erklärt und die wichtigsten Eigenschaften und Standards beschrieben. Die Themenauswahl korreliert mit der Relevanz zum praktischen Anwendungsfall und ist keinesfalls allumfassend. Die Angaben stammen sinngemäß aus [8],[12],[16],[18].

2.2.1 Was ist LoRaWAN?

LoRaWAN beschreibt die grundlegende Systemarchitektur des Netzwerks und das verwendete Kommunikationsprotokoll. Diese beiden Parameter haben den größten Einfluss auf die Netzwerkkapazität, Batterielebensdauer und die Sicherheit des Netzwerkes, allesamt wichtige Anforderungen im Internet der Dinge.

LoRaWAN ist ein MAC-Protokoll, welches auf LoRa aufbaut, wobei auch eine Verwendung von FSK möglich ist und welches bezüglich des ISO/OSI-Referenzmodells der Schicht 2 zugeordnet werden kann. Außerdem sind im LoRaWAN-Standard einige Elemente eines Netzwerk-Protokolls enthalten, also Elemente der Schicht 3 (vgl. Abbildung 5).

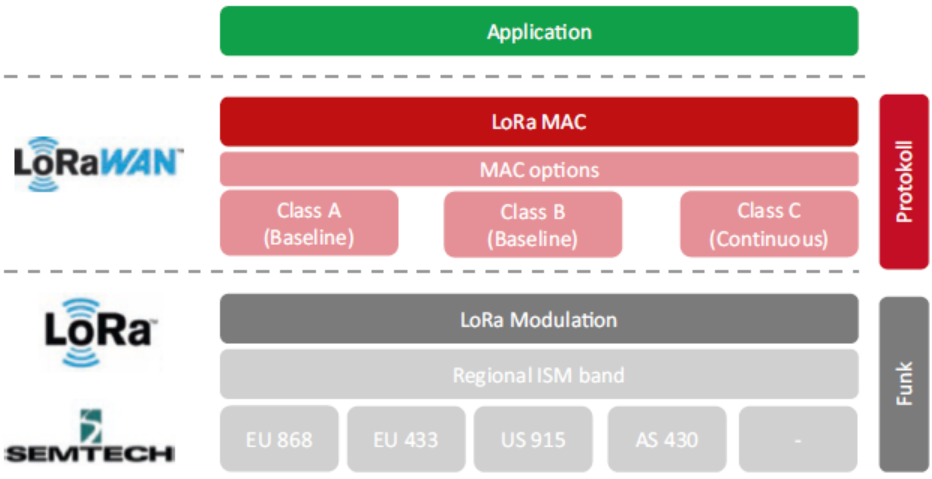


Abbildung 9: Unterscheidung LoRa und LoRaWAN [8][1]

[1] Kapitel 3.1: *Netzwerkarchitektur (technisches Funktionsprinzip);* Seite 26

2.2.2 Netzwerkarchitektur

Ein LoRaWAN-Netzwerk besteht typischerweise aus vier Hauptbestandteilen: Sensoren, mindestens einem Gateway, einem Netzwerk- und Anwendungsserver, wobei letztere nicht physisch voneinander getrennt sein müssen. Die Netzwerkarchitektur ist in einer Sterntopologie angeordnet. Im Unterschied zu Mesh-Netzwerken, wo Endknoten auch die Informationen anderer Endknoten weiterleiten, senden hierbei die Sensoren direkt an ein Gateway. Diese fungieren als Schnittstelle zwischen den Sensoren und dem Netzwerkserver und dienen demnach der Informationsübertragung. Dabei sind einzelne Sensoren nicht einem bestimmten Gateway zugeordnet, vielmehr empfangen mehrere Gateways die Datenpakete und leiten sie an den Netzwerkserver weiter. Das Herausfiltern redundanter Datenpakete übernimmt dabei der Netzwerkserver. Dieser ist häufig cloudbasiert, aber es gibt noch andere Möglichkeiten der Umsetzung (siehe Abschnitt 2.2.7). Die Kommunikation zwischen dem Gateway und dem Netzwerkserver erfolgt i. d. R. über Mobilfunk, Wi-Fi, Satellit oder Ethernet. Dieser Übertragungsweg wird als *Backhaul* bezeichnet.

In folgender Abbildung ist eine typische LoRaWAN-Netzwerkarchitektur dargestellt.

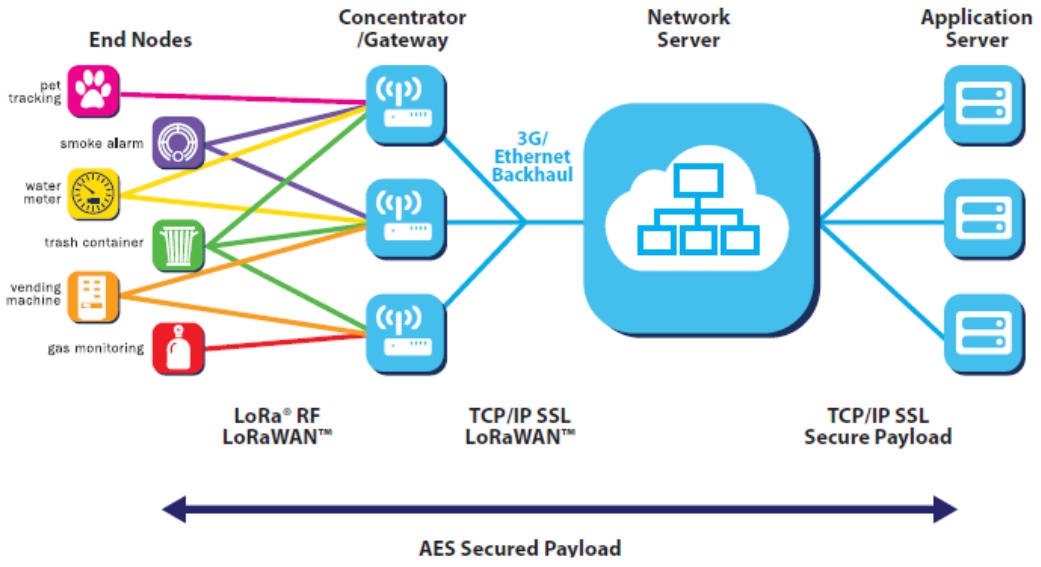


Abbildung 10: Netzwerkarchitektur eines LoRaWAN-Netzwerkes [16][2]

[2] Kapitel 4: *What is LoRaWAN?*; Seite 8

Der Sensor und das Gateway besitzen jeweils eine Physikalische Schicht (PHY, ISO/OSI-Schicht 1), welche zur Übertragung der einzelnen Bits dient. Dabei wird LoRa bzw. FSK verwendet. Darüber hinaus verfügen beide Systeme über einen *Hardware Abstraction Layer* (HAL), dessen Aufgabe die Abrufung von Hardwareinformationen und die Hardwarekommunikation ist. Das *Serial Peripheral Interface* (SPI) ist ein Bus-System, welches nach dem Master-Slave-Prinzip arbeitet und zum Datenaustausch dient (siehe Abbildung 11).

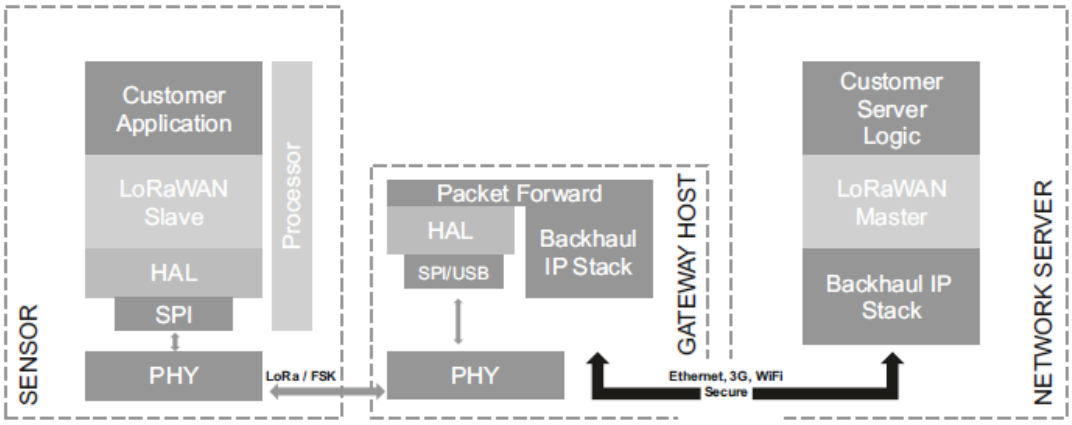


Abbildung 11: detaillierterer Aufbau eines LoRaWAN-Netzwerkes [8][3]

Ein sogenannter *Packet Forwarder* ist ein Programm, welches auf einem Gateway läuft und einerseits mit dem LoRa Chip interagiert, um LoRa-Pakete zu empfangen bzw. zu senden. Andererseits wird über dieses Programm eine Kommunikation mit dem Netzwerkserver realisiert, um die Pakete für die jeweiligen Anwendungen zu übertragen [17].

2.2.3 Sensor-Geräteklassen

Aufgrund der vielfältigen Anwendungsmöglichkeiten von Endgeräten sind unterschiedliche Anforderungsprofile zu erfüllen. Um trotz dieser Vielzahl von Endanwendungsprofilen die Verwendung zu optimieren, verwendet LoRaWAN verschiedene Übertragungsmodi und teilt die entsprechenden Endgeräte in die Klassen A bis C ein. Die Geräteklassen stellen die Netzwerk-Downlink-Kommunikationslatenz der Batterielebensdauer gegenüber und wägen diese

[3] Kapitel 3.1: *LoRaWAN - Netzwerkarchitektur*, Seite 26

gegeneinander ab (siehe Abbildung 12). Die Sensoren aller Geräteklassen sind bidirektional aufgebaut.

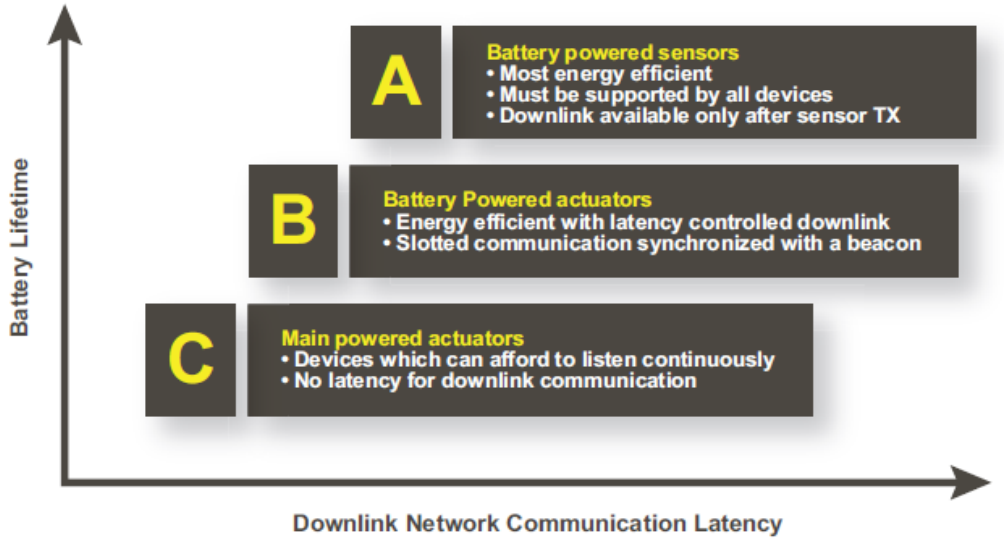


Abbildung 12: Übersicht der LoRaWAN-Geräteklassen [16][4]

Klasse A-Geräte

Der Klasse A-Modus ist der primäre Übertragungsmodus und muss von jedem Endgerät und Gateway unterstützt werden. Dabei folgen auf jede Uplink-Übertragung eines Sensors zwei kurze Downlink-Empfangsfenster. Die Dauer der Zeitfenster ist von den Bedürfnissen des jeweiligen Gerätes abhängig und folgt dem Zeitprinzip des ALOHA-Protokolltyps, wonach die Zeitfenster der Übertragung zufällig gewählt werden. Die Kommunikation erfolgt ausschließlich in diesem Zeitraum, die restliche Zeit befindet sich das Endgerät im Stand-by-Modus. Daher ist dieser Modus der energieeffizienteste.

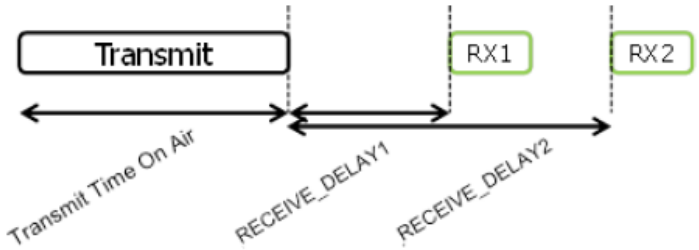


Abbildung 13: LoRaWAN Klasse A-Übertragung [12][5]

[4] Kapitel 4: *What is LoRaWAN?*; Seite 10
[5] Kapitel 3.3 *Receive Windows*; Seite 13

Klasse B-Geräte

Im Gegensatz zu Endgeräten der Klasse A verfügen Klasse B-Geräte über feste Zeitfenster zur Übermittlung der Daten, dessen Öffnung über ein vom Gateway gesendetes Synchronisationsbeacon, ein kleines Datenpaket mit Metadaten, realisiert wird. Aus diesem Beacon und der Ping-Slot-Periodizität kann ein Klasse B-Gerät errechnen, wann Daten empfangen werden können. Zum entsprechenden Zeitpunkt wird dann der LoRa-Transceiver eingeschaltet, um eventuell vorhandene Daten zu empfangen (siehe Abbildung 14). Somit können auch Daten vom Endgerät empfangen werden ohne eine vorherige Uplink-Übertragung. Allerdings werden auch Zeitfenster geöffnet, wenn keine Datenpakete zum Empfang bereitstehen. Die Implementierung dieses Übertragungsmodus ist optional für LoRa zertifizierte Endgeräte.

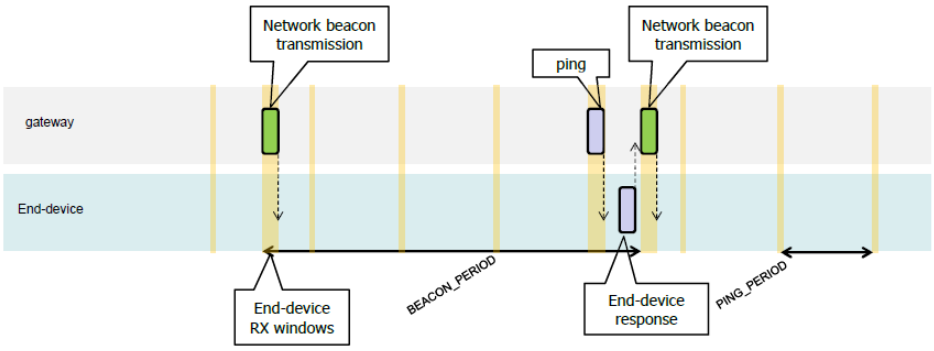


Abbildung 14: LoRaWAN Klasse B-Übertragung mit Beacon und Ping-Slots [12][6]

Klasse C-Geräte

Geräte der Klasse C besitzen nahezu ständig geöffnete Empfangsfenster, welche nur für das Senden von Paketen geschlossen werden. Auch dieser Übertragungsmodus ist optional.

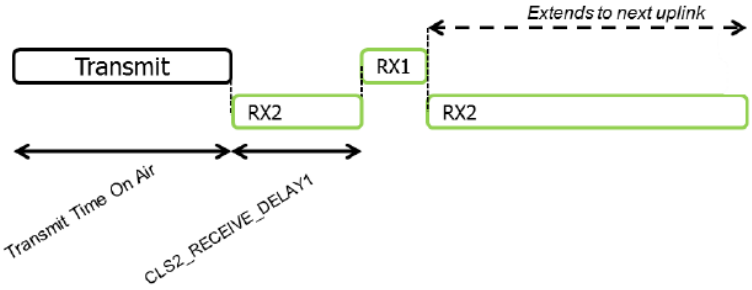


Abbildung 15: LoRaWAN Klasse C-Übertragung [12][7]

[6] Kapitel 9: *Principle of synchronous network initiated downlink (Class-B option);* Seite 70
[7] Kapitel 17: *Continuously listening end-device;* Seite 87

2.2.4 IT-Sicherheit

LoRaWAN verwendet zwei Sicherheitsebenen: eine Ebene für das Netzwerk und eine Ebene für die Anwendung. Jede dieser Ebenen hat einen eigenen Sitzungsschlüssel, welcher 128 Bit lang ist. Wie diese generiert werden, hängt von der Art des Netzwerkbeitritts ab. Detailliertere Angaben hierzu folgen im nächsten Abschnitt 2.2.5 *Arten des Netzwerkbeitritts*. Als Verschlüsselungsalgorithmus verwendet LoRaWAN einen AES-128 im leicht abgewandelten CBC-Modus.

Auf der Netzwerkebene soll die Authentizität eines Gerätes mithilfe des Netzwerkssitzungsschlüssels (NwkSKey) sichergestellt werden. Hierzu wird die Nachrichtenintegrität mit Hilfe des Message Integrity Codes (MIC) validiert. Zusätzlich beinhaltet jede LoRa-Nachricht einen Frame Counter, welcher dem Schutz vor Telegram-Replay-Angriffen dient.

Auf der Anwendungsebene übernimmt der Anwendungssitzungsschlüssel (AppSKey) die Aufgabe der Ende-zu-Ende verschlüsselten Kommunikation zwischen dem Netzwerkserver und dem Sensor. Mithilfe des AppSKey werden die Nutzdaten des Sensors verschlüsselt.

Auch wenn die Sitzungsschlüssel niemals übertragen werden, so müssen sie dennoch im Sensor und LNS bzw. Applikationsserver gespeichert sein. Somit ist aus Sicherheitsaspekten die Möglichkeit eines physikalischen Zugriffes auf den Schlüsselspeicherort ebenfalls zu bedenken und gerade bei der Verwaltung einer Vielzahl von Endknoten ist ein intelligentes Lifecycle-Key-Management notwendig.

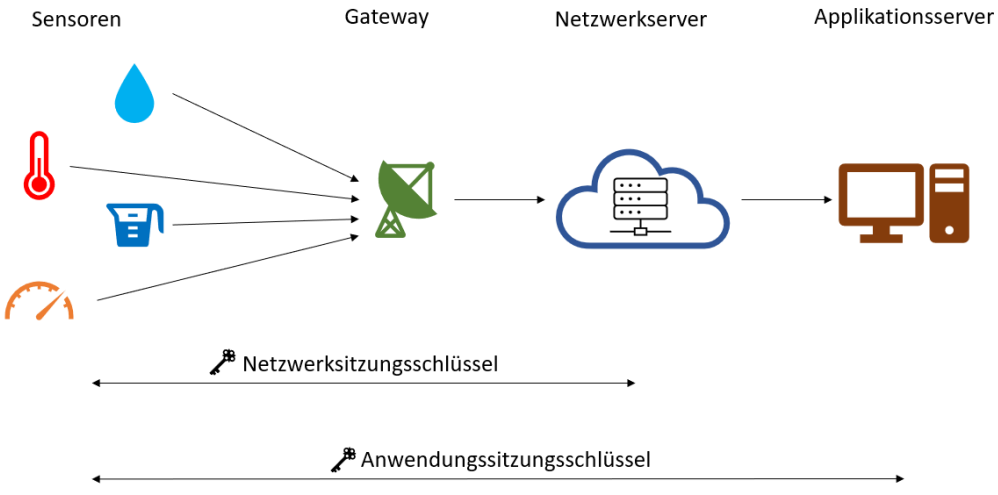


Abbildung 16: Verschlüsselung im LoRaWAN mit 2 Sicherheitsebenen

2.2.5 Netzwerkbeitritt allgemein

Jeder Sensor benötigt einen Netzwerkserver, um ein LoRaWAN-Netzwerk aufzubauen. Dabei kann grundsätzlich zwischen zwei verschieden Beitrittsarten zum LNS unterschieden werden. Zum einen besteht die Möglichkeit einer Over-The-Air-Aktivierung (OTAA) und zum anderen ist auch eine Activation by Personalization (ABP) realisierbar. Diese beiden Verfahren werden im Folgenden ausführlich erläutert.

OTAA

Bei der OTAA handelt es sich um ein dynamisches Aktivierungsverfahren, welches in überwiegender Mehrheit die Werkseinstellung der Sensoren ist. Bei der Zuordnung eines Sensors zum LNS kommt es hierbei zu einer konkreten, von der LoRa-Alliance im LoRaWAN-Standard dokumentierten, Join-Prozedur. Bei jedem erneutem Netzwerkbeitritt muss dieses Procedere nochmals durchlaufen werden und es kommt zu einer erneuten Schlüsselgenerierung.

Vor der Aktivierung eines Sensors muss sichergestellt sein, dass dieser die DevEUI, AppEUI und den AppKey gespeichert hat. Die DevEUI und die AppEUI sind zwei 64-Bit lange Adressen, welche den Sensor und die Applikation mittels des vom IEEE standardisierten MAC-Adressformats eindeutig identifizieren. Beim AppKey handelt es sich um einen 128-Bit Root-Schlüssel, der zur Berechnung des MIC verwendet wird. Dieser geheime AppKey muss auch dem LNS zugänglich gemacht werden, damit eine OTAA erfolgreich abgeschlossen werden kann.

Sämtliche LoRa Uplink- und Downlink-Nachrichten beinhalten in ihrer PHY-Payload einen 1-Byte langen MAC-Header (MHDR), eine mindestens 7-Byte lange MAC-Payload sowie den 4-Byte langen MIC (siehe Abbildung 17).

Size (bytes)	1	*7..M*	4
PHYPayload	MHDR	MACPayload	MIC

Abbildung 17: Format der PHY-Payload [12][8]

[8] Kapitel 4.1 *MAC-Layer (PHYPayload);* Seite 16

In dem MHDR-Feld codieren die ersten drei Bits (MSB) den Nachrichtentyp (MType). Es wird demnach zwischen acht Nachrichtentypen unterschieden, welche in folgender Abbildung dargestellt sind.

MType	Description
000	Join-request
001	Join-accept
010	Unconfirmed Data Up
011	Unconfirmed Data Down
100	Confirmed Data Up
101	Confirmed Data Down
110	Rejoin-request
111	Proprietary

Abbildung 18: MAC-Nachrichten Typen [12][9]

Sendet ein Endknoten ein Join-Request an einen LNS, dann lässt sich im MType-Feld des MHDR die Bitfolge „000“ finden. In der MAC-Payload einer Join-Request-Nachricht sind die jeweils 8-Byte langen AppEUI und DevEUI, sowie eine 2-Byte lange DevNonce enthalten (siehe Abbildung 19). Bei der DevNonce handelt es sich um eine zufällig generierte, einmalige Nummer. Die Join-Request-Nachricht wird immer unverschlüsselt gesendet.

Size (bytes)	8	8	2
Join-request	AppEUI	DevEUI	DevNonce

Abbildung 19: MAC-Payload einer Join-Request-Nachricht [12][10]

Der 4-Byte lange MIC einer Join-Request-Nachricht wird mit dem geheimen AppKey mittels des AES-CMAC-128-Algorithmus berechnet und nach der MAC-Payload an die Nachricht angehangen (vgl. Abbildung 17). Mit dem 1-Byte langen MHDR, der 18-Byte langen MAC-Payload und dem 4-Byte langen MIC ist die PHY-Payload einer Join-Request Nachricht immer 23-Byte lang. Diese unverschlüsselte Nachricht wird vom LNS empfangen, welcher zunächst mit seinem gespeicherten AppKey die Gültigkeit des MIC überprüft. Ist die Validierung erfolgreich, so generiert der LNS eine 3-Byte lange AppNonce, eine

[9] Kapitel 4.2.1 *Message Type (MType bit field);* Seite 17
[10] geringfügig abgewandelt nach Kapitel 6.2.2 *Join-request message;* Seite 52

3-Byte lange NetID sowie eine 4-Byte lange DevAddr, die fortan als „Name“ des Sensors fungiert. Außerdem erfolgt die Berechnung zweier 128-Bit-Schlüssel, des AppSKey und des NwkSKey, auf Basis der Join-Request-Nachricht und der AppNonce des LNS. Gemäß den Abbildungen 17 und 18 enthält die PHY-Payload der Join-Accept-Nachricht im MHDR die MType-Bitfolge „001“ sowie in der MAC-Payload die AppNonce, NetID, DevAddr, DLSettings, RxDelay und die optionale CFList (siehe Abbildung 20). Die DLSettings beinhalten einige Downlink-Parameter, das RxDelay-Feld definiert die Zeitverzögerung zwischen Empfang und Senden und in der optionalen CFList werden regionale, spezifische Parameter wie Frequenzen etc. festgelegt. Der MIC der Nachricht wird vom LNS berechnet und ebenfalls mit der Nachricht gesendet.

Size (bytes)	3	3	4	1	1	(16) Optional
Join-accept	AppNonce	Home_NetID	DevAddr	DLSettings	RxDelay	CFList

Abbildung 20: MAC-Payload der Join-Accept-Nachricht [12][11]

Das Besondere an der Join-Accept-Nachricht ist außerdem, dass diese verschlüsselt gesendet wird. Zur Verschlüsselung wird der AES-128-Verschlüsselungsalgorithmus mit dem AppKey verwendet. Nachdem der Sensor den MIC überprüft hat, kann dieser die Nachricht mit seiner Version des AppKey wieder entschlüsseln, sodass beide Geräte dieselbe AppNonce und DevNonce hinterlegt haben. Somit kann der Sensor seine eigene Berechnung der Sitzungsschlüssel vornehmen. Als Folge dessen haben der Sensor und der LNS die gleichen Sitzungsschlüssel gespeichert, ohne dass diese jemals gesendet wurden. In der folgenden Abbildung 21 ist die OTAA in einer Übersicht zusammengefasst.

[11] geringfügig abgewandelt nach Kapitel 6.2.3 *Join-accept message;* Seite 53

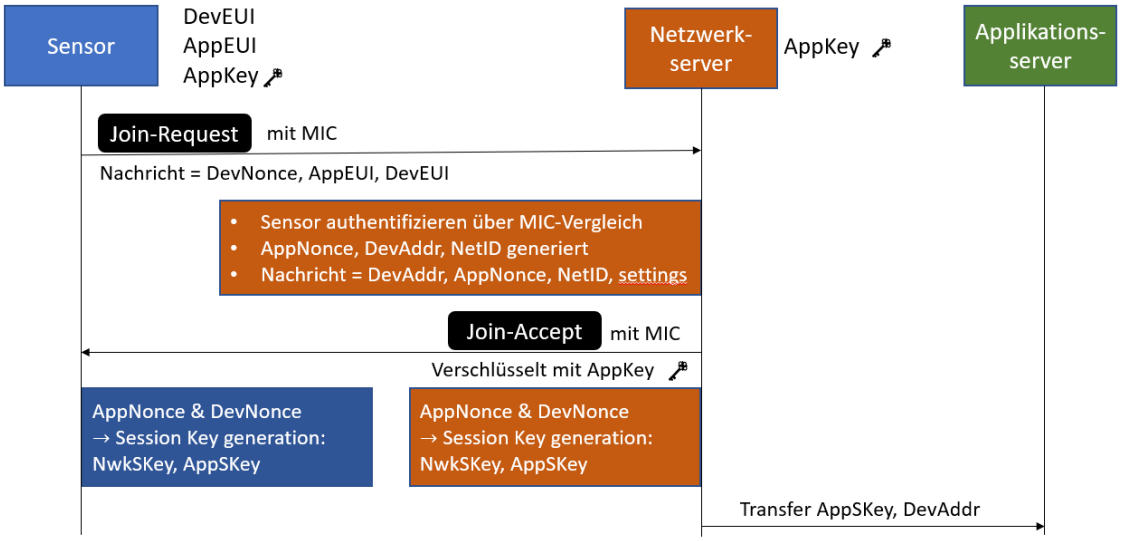


Abbildung 21: Darstellung des OTAA-Ablaufes

Bei der OTAA werden bei jeder Aktivierung Sitzungsschlüssel und Zertifikate dynamisch zur Verfügung gestellt. Dadurch wird die Sicherheit verbessert und ermöglicht bei Bedarf einen Wechsel des Netzwerkes. Der dynamischen Schlüsselgenerierung steht die statische Methode gegenüber, welche im Folgenden beschrieben wird.

ABP

Bei dieser Methode des Netzwerkbeitritts eines Sensors erfolgt der Aufbau einer Sitzung nicht dynamisch, sondern statisch. Dabei werden statische AppSKey und NwkSKey verwendet, die dem LNS in Kombination mit der DevAddr bereits bekannt sind. Somit ist bei der ABP das Vorhalten eines gemeinsamen AppKey nicht notwendig und auch die Join-Prozedur entfällt.

Diese simple Variante des Netzwerkbeitritts erfordert, dass die kryptographischen Schlüssel vom Hersteller oder vom Anwender fest in den Endknoten und den LNS codiert sein müssen. Der Sensor kann demnach nicht das Netzwerk wechseln und keine Schlüssel austauschen.

2.2.6 Daten-Telegramme

Grundsätzlich lassen sich Datenpakete auf zwei unterschiedlichen Wegen senden und empfangen. Man unterscheidet hierbei die Telegrammtypen *Unconfirmed Data Up* und *Confirmed Data Up,* welche gemäß dem MType-Feld in

Abbildung 18 vom LNS erkannt werden. Wie die Namen bereits vermuten lassen, benötigt eine Confirmed Data Up-Nachricht eine Antwort des Netzwerkservers, welche die Ankunft des Paketes bestätigt. Ein solcher Mechanismus ist bei einer Unconfirmed Data Up-Nachricht nicht vorgesehen und daher auch nicht notwendig.

Als Werkseinstellung verwenden die im Rahmen dieses Projektes verwendeten LoRa-Sensoren den Nachrichtenmodus Unconfirmed Data Up. Allerdings lässt sich diese Einstellung mit je nach Hersteller variierendem Aufwand ändern. Eine detailliertere Betrachtung der Parametriermöglichkeiten der Sensoren wird im Abschnitt 3.2 vorgenommen.

Unabhängig von der Notwendigkeit einer Bestätigungsnachricht, ist die MAC-Payload einer Datennachricht identisch aufgebaut. Diese enthält immer einen sogenannten Frame Header (FHDR) und bei einer mit Nutzdaten gefüllten Datennachricht auch stets ein Frame Port-Feld (FPort) sowie eine Frame Payload (FRMPayload), welches in folgender Abbildung 22 nochmals dargestellt ist.

Size (bytes)	7..22	0..1	0..*N*
MACPayload	FHDR	FPort	FRMPayload

Abbildung 22: MAC-Payload einer Datennachricht [12][12]

Der 7- bis 22-Byte lange FHDR beinhaltet neben der 4-Byte langen DevAddr, ein Frame Control-Byte (FCtrl) und einen 2-Byte langen Frame Counter (FCnt). Während im FCtrl je nach Uplink- oder Downlink-Nachricht unterschiedliche Sendeparameter codiert sind, enthält der FCnt einen Zähler, der die einzelnen Datennachrichten hochzählt. Optional können weiterhin bis zu 15 Bytes an MAC-Kommandos im Frame Options Feld (FOpts) übertragen werden (siehe Abbildung 23).

Size (bytes)	4	1	2	0..15
FHDR	DevAddr	FCtrl	FCnt	FOpts

Abbildung 23: Format des FHDR einer Datennachricht [12][13]

[12] Kapitel 4.3.2 *Port Field (FPort);* Seite 25
[13] Kapitel 4.3.1 *Frame Header (FHDR);* Seite 18

2.2.7 Möglichkeiten zur Umsetzung des Netzwerkservers

Wie bereits in Abschnitt 2.2.2 *Netzwerkarchitektur* dargestellt, benötigt jedes LoRaWAN-Netzwerk einen LNS, der die Verwaltung des Netzwerkes übernimmt und dabei Funktionen wie beispielsweise das Herausfiltern redundanter Datenpakete, der Auswahl der verwendeten Kanäle, das Anpassen der Spreizfaktoren sowie das Erzeugen und Verwalten der Sitzungsschlüssel übernimmt. Zur Umsetzung dieses Netzwerkmanagementsystems sind im Wesentlichen drei verschiedene Möglichkeiten denkbar, welche im Folgenden vorgestellt werden. Dabei wird der in dieser Arbeit untersuchte Anwendungsfall berücksichtigt, etwa dass die Sensordaten einer SPS zur Verfügung gestellt werden sollen.

Cloudbasierter LNS

Die klassischste Variante eines LNS ist die Umsetzung mithilfe eines cloudbasierten Netzwerkservers. Als Beispiel für einen solchen Server soll an dieser Stelle *The Things Network* (TTN) als einer der am weitesten verbreiteten Anbieter genannt werden. Diese der Lora-Alliance angehörigen Open Source Lösung bietet offene Programmierwerkzeuge und eine globale Community, mit dem Ziel ein globales, kollaboratives IoT-Netzwerk aufzubauen. Dabei übernehmen Freiwillige die Bereitstellung, Errichtung und Betreuung von Gateways, welche dann bei einer Registrierung im TTN alle LoRa-Datenpakete von Sensoren weiterleitet, die sich innerhalb des Abdeckungsradius befinden. Aktuell sind 20.500 Gateways in 151 Ländern im TTN registriert und es engagieren sich 165.000 Freiwillige bei der Umsetzung des weltweit größten IoT-Netzwerkes. Dabei werden täglich über 46 Millionen Nachrichten versendet (Stand: 22.02.2022) [20].

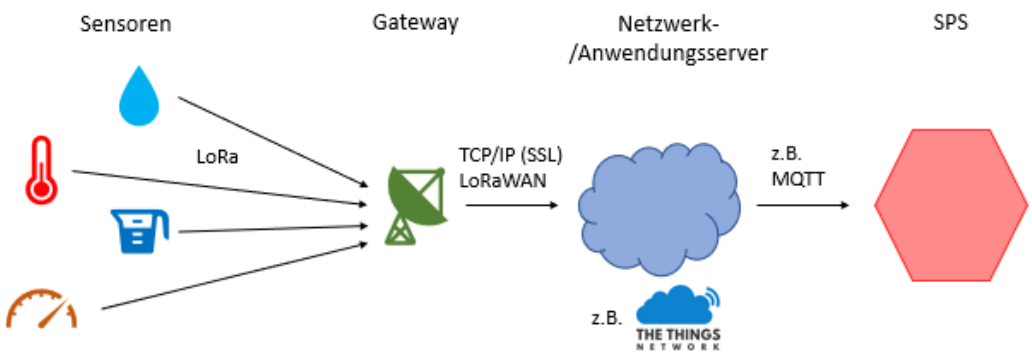


Abbildung 24: cloudbasiertes LoRaWAN-Netzwerk (TTN-Logo [20])

In Abbildung 24 ist die Umsetzung des Anwendungsfalls mittels einer cloudbasierten Lösung dargestellt. Die verwendeten Gateways werden im LNS mit ihrer MAC-Adresse registriert und einem Netzwerk zugeordnet. Anschließend können die Sensoren, welche diesem Netzwerk zugeordnet sind, ihre Daten an den LNS über das Gateway senden. Auf dieser Plattform kann sich die eigentliche IoT-Anwendung befinden, aber es ist auch eine Weiterleitung der entschlüsselten Nutzdaten über entsprechende Broker wie z.B. MQTT möglich. Somit können die Sensordaten an einer SPS verfügbar gemacht werden, allerdings ist eine Internetverbindung der SPS unabdingbar [20].

Self-Hosting des LNS

Es besteht die Möglichkeit ein eigenes Netzwerk zu betreiben, um das öffentliche Netzwerk einer cloudbasierten Lösung zu umgehen. Dabei kann ein solches Netz gänzlich ohne Internetverbindung betrieben werden. *ChirpStack* ist hierbei als eine Open Source Lösung zu nennen, welche eine gebrauchsfertige Lösung für LoRaWAN-Netzwerke anbietet [21].

In Abbildung 25 ist eine Darstellung mit einem selbstbetriebenen LNS dargestellt. Dabei übernehmen die von ChirpStack bereit gestellten modularen Netzwerkkomponenten die Aufgaben des LNS und ermöglichen so ein LoRaWAN-Netzwerk, welches ohne Internetverbindung auskommt.

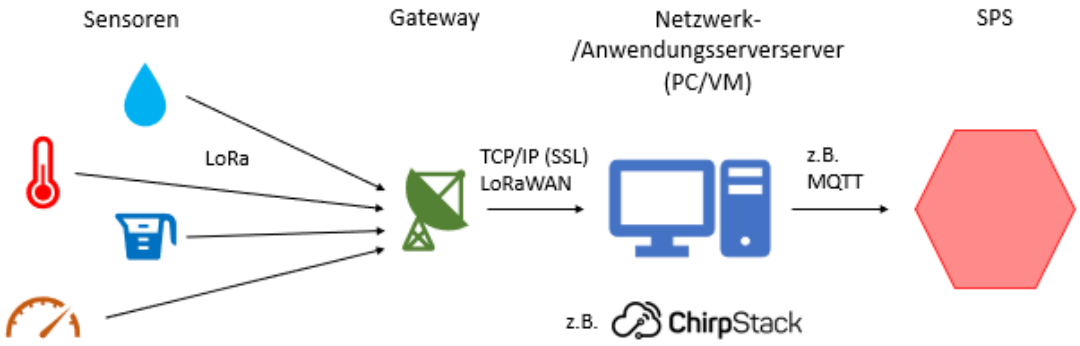


Abbildung 25: Eigenbetrieb des LNS auf Basis von ChirpStack (Logo: [21])

Entwicklung eines eigenen LNS

Für eine maximale Unabhängigkeit von Drittanbietern ist es notwendig, einen eigenen LNS zu entwickeln und zu programmieren. Wird dieser dann auf einer SPS direkt implementiert, so kann ein Gateway mittels Ethernet-Verbindung

direkt an der Steuerungseinheit betrieben werden. Dabei muss das Semtech UDP-Protokoll, welches standardweise bei den meisten LoRaWAN-Gateways als Packet Forwarder eingesetzt wird, implementiert werden. In folgender Abbildung 26 ist eine solche Architektur eines LoRaWAN-Netzwerkes dargestellt.

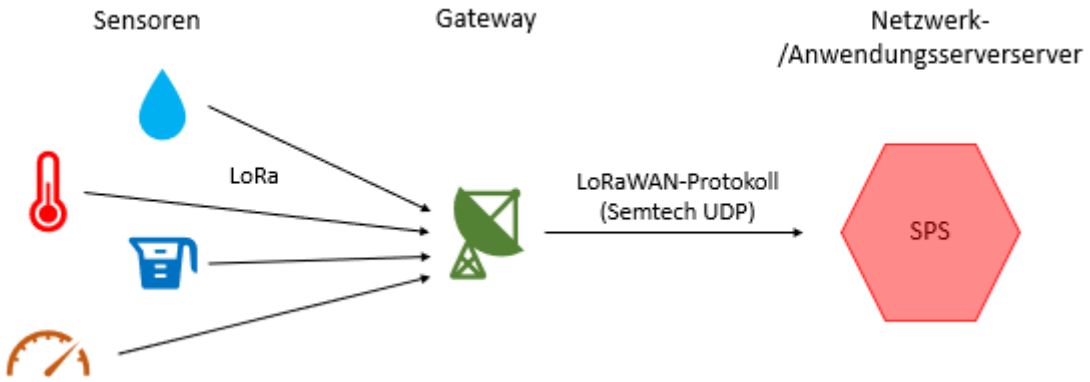


Abbildung 26: LoRaWAN-Netzwerk mit LNS auf einer SPS

2.3 JSON-Format

In diesem Kapitel soll das Datenaustauschformat JavaScript Object Notation, kurz JSON, genauer betrachtet werden. Im Semtech UDP-Protokoll wird die JSON-Datenstruktur genutzt, um Meta- und Nutzdaten zu senden. Während sich Abschnitt 2.3.1 allgemein mit dem Format befasst, wird im Abschnitt 2.3.2 die Verwendung von JSON-Datenstrukturen innerhalb der LoRaWAN-Architektur in den Vordergrund gestellt. Die hier zusammengefassten Informationen stammen sinngemäß aus [40] und [41].

2.3.1 Allgemeine Struktur und Standards

JSON ist ein schlankes, kompaktes Datenaustauschformat in einer für Menschen leicht lesbaren textbasierten Struktur. Dieses Format ist unabhängig von Programmiersprachen und wurde erstmals 1997 von Douglas Crockford spezifiziert, während derzeit der ECMA-404 als Standard gilt, welcher unter [41] abrufbar ist. Trotz der Unabhängigkeit von Programmiersprachen folgt das Textformat vielen Konventionen der C-basierten Sprachen. JSON baut im Wesentlichem auf zwei Strukturen auf. Einerseits ist das eine Sammlung von

Namen-Werte-Paaren, welche in verschiedenen Sprachen häufig als Objekt oder Struct realisiert werden. Andererseits ist das eine geordnete Liste von Werten, welche in den meisten Sprachen z.B. als Array angelegt wird.

Gemäß dem JSON-Standard wird zwischen Objekten, Arrays, Werten, Strings und Nummern unterschieden, welche in diesem genauer spezifiziert werden. Ein Objekt beginnt beispielsweise immer mit einer geschwungenen *Klammer auf* und endet mit einer geschwungenen *Klammer zu*. Innerhalb dieses Objektes ist die ungeordnete Menge von Namen-Werte-Paaren enthalten, wobei nach jedem Namen ein Doppelpunkt folgt, bevor der Wert angegeben ist. Jedes dieser Paare ist durch Kommata getrennt. Tiefergehende Informationen zu der JSON-Datenstruktur können den angegebenen Quellen entnommen werden.

Die Vielfalt der Darstellung von JSON-Objekten sowie z. B. die Tatsache, dass zwischen einzelnen JSON-Objekten beliebig viele Leerzeichen eingefügt werden können, macht ein robustes Parsen mitunter schwierig.

2.3.2 JSON-Datenstruktur im LoRaWAN-Standard

Die JSON-Datenstruktur wird innerhalb des Semtech UDP-Protokolls sowohl im Up- als auch im Downlink-Bereich verwendet. Alle benötigten Informationen zu den verwendeten Datenstrukturen können in der Spezifikation des Standards, welcher unter [35] abgerufen werden kann, nachgelesen werden. Als Beispiel sei an dieser Stelle erwähnt, dass innerhalb des Upstream-Protokolls, also einer Nachricht vom Gateway zum LNS, ein gesendetes JSON-Objekt ein Array namens ‘‘rxpk“ enthält, welches einige Paketinformationen beinhaltet. Diese sind mit den bereits angesprochenen Namen-Werte-Paaren kodiert und beinhalten neben Uhrzeit und Zeitstempel auch Informationen über die verwendeten Kanäle, Frequenzen und Datenraten sowie auch die eigentlichen Datenfelder und noch vieles mehr. Allerdings enthält nicht jedes Upstream-Paket auch das ‘‘rxpk“-Array, sondern es existieren auch Statuspakete, die lediglich mit einem ‘‘stat“-Array befüllt sind, welche den Gateway-Status übermitteln. Darüber hinaus sind auch JSON-Daten mit beiden genannten Arrays möglich.

3. Konkreter Anwendungsfall

Der in dieser Arbeit zu evaluierende Anwendungsfall soll in diesem Kapitel genauer erörtert werden. Wie bereits eingangs erwähnt, sollen LoRa-Raumklimasensoren zur Messdatenerfassung herangezogen werden. Diese sollen in Referenzräumen eingesetzt werden, um mit den gewonnenen Daten eine Heizkreisregelung zu steuern. Dabei müssen die von den Sensoren gemessenen Temperaturwerte der Gebäudeautomatisierungs-SPS zur Verfügung gestellt werden. Mittels der übertragenen Daten, welche als Aktualwerte in die Heizkreisregelung eingehen, sollen die jeweiligen Heizkreise in einem Gebäude energieeffizient und automatisiert geregelt werden. In diesem Kontext sind unter Berücksichtigung der technischen Grundlagen der LoRa- und LoRaWAN-Technologie (siehe Kapitel 2) verschiedene Frage- und Problemstellungen zu lösen, welche in den folgenden Abschnitten systematisch vorgestellt werden.

Ziel dieser Arbeit ist es, ob und mit welchen Einschränkungen man LoRa-Raumklimasensoren zur Steuerung von Regelkreisen auf einer SPS einsetzen kann. Dabei wird ein Testaufbau mit ausgewählten Sensoren und Gateways verwendet.

3.1 Auswahl der Hardware

Zur Evaluierung des beschriebenen Anwendungsfalls wird zunächst betrachtet, welche LoRa-Sensoren und LoRa-Gateways zur Anwendung kommen sollen. Nach ausgiebiger Recherche sind auf Basis der im Folgenden genannten Kriterien die in Tabelle 1 dargestellten vier Sensoren und zwei Gateways ausgewählt worden, die eine möglichst breit gefächerte Produktpalette abdecken sollen:

- Produkteigenschaften wie Batterielebensdauer, Sendeparameter etc.
- Marktanteil (soweit abschätzbar)
- Verfügbarkeit bei gängigen Distributoren
- Verfügbarkeit in Hinblick auf hohe Stückzahlen
- aktuelle Lieferzeiten

Tabelle 1: Übersicht der ausgewählten Sensoren und Gateways

	Hersteller	Modell	Abbildung
Sensoren	Laird Connectivity	Sentrius RS186	 [22]
	Elsys	ERS Lite	 [23]
	Dragino	LHT 65	 [24]
	RAK	7204	 [25]
Gateways	Laird Connectivity	Sentrius RG186	 [26]
	Kerlink	Wirnet iFemtocell Evolution	 [27]

Alle ausgewählten Produkte sind auf unterschiedlichste Weise und mit verschiedenen Hilfsmitteln parametrierbar und in Betrieb zu nehmen. Auf diese Möglichkeiten wird im Folgenden eingegangen.

3.1.1 Sensoren

In diesem Abschnitt sollen die vier ausgewählten Sensoren in Bezug auf ihre Möglichkeiten der Parametrierung beleuchtet werden.

Laird Connectivity – Sentrius RS186

Gemäß des Benutzerhandbuches, welches unter [28] abgerufen werden kann, steht eine Applikation für Smartphones zur Verfügung, mit dessen Hilfe der Sensor konfiguriert werden kann. Die Kommunikation zwischen dem Sensor und der App wird via Bluetooth realisiert und ist folglich mit vergleichsweise geringem Aufwand möglich. Die Applikation bietet vielfältige Möglichkeiten, wie z. B. der Einstellung von Parametern wie Sendeintervalle, einer Livebetrachtung der Sensormesswerte, einer Anpassung von LoRa-Konfigurationselementen wie DevEUI und AppEUI, LoRa-Netzwerk-Informationen wie Datenraten und SNR sowie Firmware-Update- und Alarmoptionen.

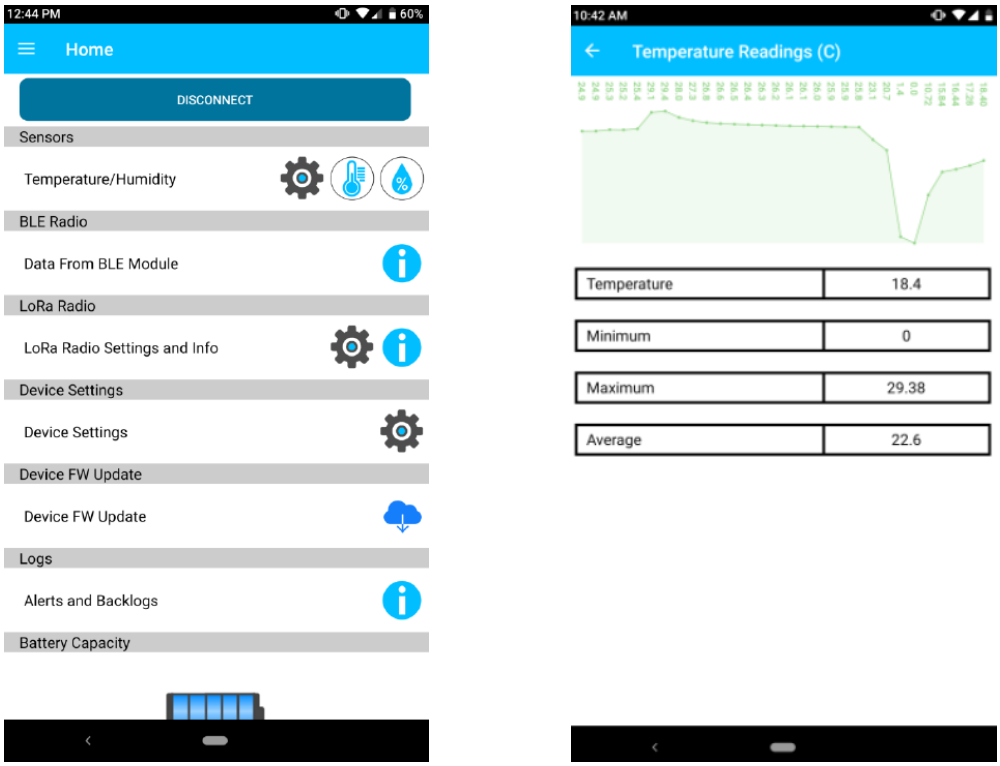


Abbildung 27: Hauptmenü (links) sowie Temperaturverlaufsseite (rechts) der Applikation [28][14]

[14] Kapitel 10 *Mobile Application;* Seiten 25 & 27

Elsys – ERS Lite

Zur Konfiguration des Sensors steht auch bei dem Modell ERS Lite der Firma Elsys eine mobile Applikation namens *Sensor Settings* kostenfrei zur Verfügung. Genauere Informationen sind dem Benutzerhandbuch zu entnehmen [29].

Die Kommunikation zwischen Sensor und Applikation erfolgt hierbei mittels Near Field Communication (NFC), sodass das verwendete Smartphone eine solche Funktionalität unterstützen muss. Die einstellbaren Parameter sind bei dieser Ausführung ebenfalls vielfältig und beinhalten z. B. neben den Sendeintervallen und Sensorschlüsseln auch erweiterte Konfigurations-möglichkeiten wie Datenraten und Ports, aber auch Debug-Berichte können erstellt werden.

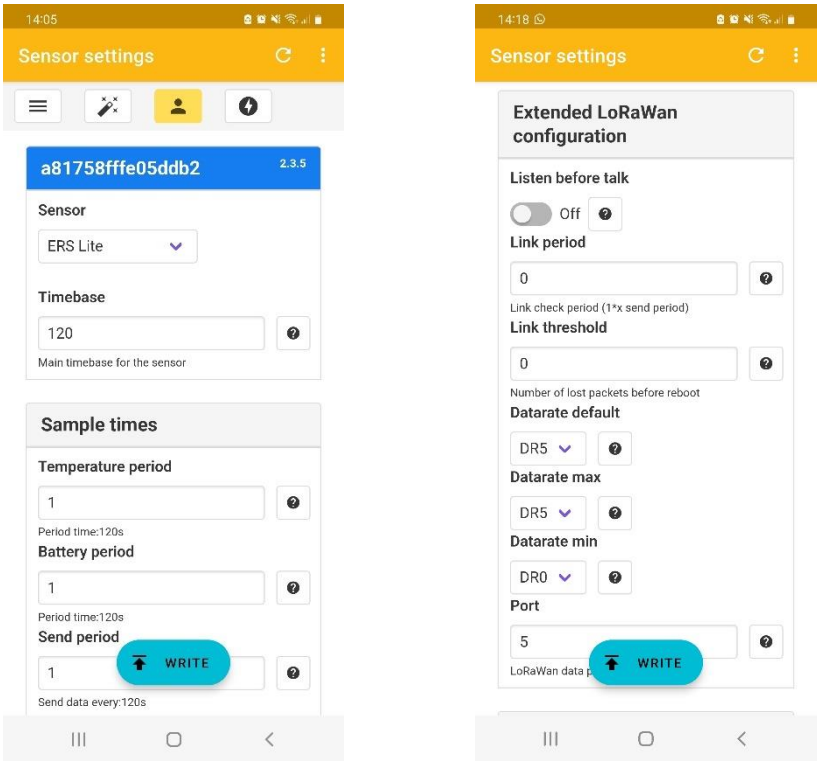


Abbildung 28: Screenshots der Elsys-Sensor-Applikation zur Parametrierung des Sensors

Dragino LHT 65

Gemäß des LHT65-Benutzerhandbuches [30] können AT-Kommandos zur Konfiguration und Informationsabfrage des Sensors benutzt werden. Hierzu muss das mitgelieferte Programmierkabel mit einem passenden (nicht im Lieferumfang enthaltenen) USB-TTL-Umsetzer gemäß der in Abbildung 29a dargestellten Pinbelegung mit einem PC verbunden werden. Auf diesem kann

dann mittels einer Software für textbasierte Terminalsitzungen, eine Verbindung mit dem Sensor hergestellt werden. Anschließend können dann mit der in der Dokumentation befindlichen Kommando-Liste sämtliche Konfigurations-änderungen vorgenommen werden. Der Funktionsumfang ist zwar vergleichbar mit den Möglichkeiten der anderen beiden bereits angesprochenen Sensoren, allerdings ist für eine Verbindung neben der Software noch zusätzliche Hardware wie ein PC und entsprechende Adapter, Jumperkabel usw. notwendig. Im Rahmen dieser Arbeit wird die Windows-Software PuTTY als serielles Werkzeug zur Kommunikation mit dem Dragino-Sensor verwendet (siehe Abbildung 29b).

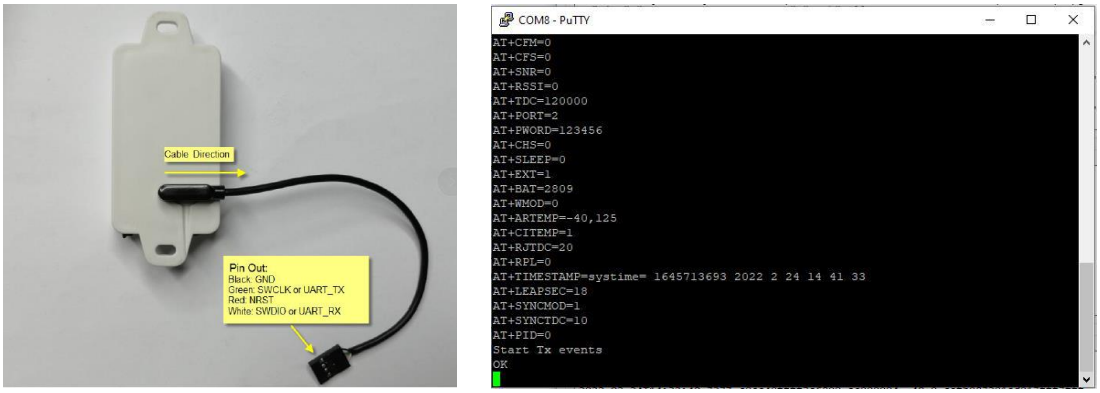


Abbildung 29a und 29b: TTL-Pinbelegung des Programmierkabels [30][15] sowie PuTTY-Screenshot während Sensorkommunikation

RAK – 7204

Zur Inbetriebnahme und Konfiguration des RAK-Sensors ist gemäß der Schnell-Start-Anleitung [31] ein nicht mitgeliefertes USB-A auf Micro-USB-B-Kabel nötig, welches direkt an die Platine des Sensors angeschlossen werden muss, um eine Kommunikation mit einem Windows-PC zu ermöglichen (siehe Abbildung 30a).

Darüber hinaus wird auch bei diesem Sensor ein serielles Werkzeug benötigt. RAK empfiehlt die eigens entwickelte Software *RAK Serial Port Tool*, mit der man ebenfalls unter Nutzung des AT-Befehlssatzes Änderungen an der Konfiguration vornehmen kann. Für die Nutzung der AT-Kommandos steht eine eigene Anleitung zur Verfügung, welche unter [32] abrufbar ist. Mittels der RAK-Software ist eine einfache Parametrierung des Sensors möglich, allerdings werden auch hier zusätzliche Hardware wie ein PC und das entsprechende Kabel

[15] Kapitel 6.1: *How to use AT Command to configure LHT65;* Seite 49

benötigt. In Abbildung 30b ist eine Ansicht des RAK Serial Port Tools bei aktiver Kommunikation mit einem Sensor dargestellt.

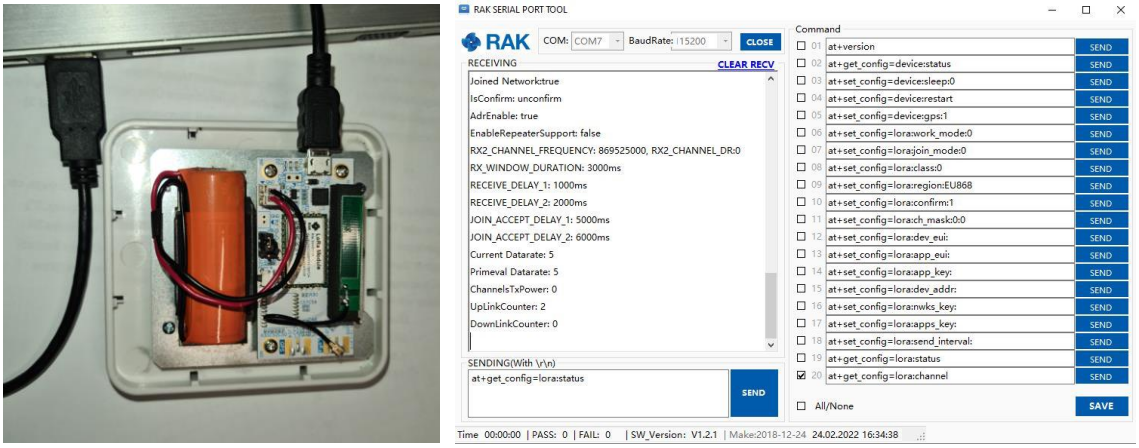


Abbildung 30a und 30b: angeschlossener RAK-Sensor und Screenshot des RAK Serial Tools nach Abfrage des LoRa-Status

3.1.2 Gateways

In diesem Abschnitt sollen die beiden Gateways in Bezug auf ihre Parametrierbarkeit und Debug-Möglichkeiten beleuchtet werden.

Laird Connectivity – Sentrius RG186

Zur Inbetriebnahme des Laird-Gateways wird das online abrufbare Benutzerhandbuch herangezogen [33]. Die Konfiguration des Gateways erfolgt über ein Webinterface. Zu diesem Zwecke muss eine Ethernet-Verbindung zum Netzwerk hergestellt werden, bevor man den Zugang über die URL *https://rg1xx1234AB*, wobei „1234AB“ den letzten 6 Stellen der MAC-Adresse des Gateways entsprechen, oder direkt über die IP-Adresse herstellen kann. In Abbildung 31 ist das Dashboard dieses Webinterfaces dargestellt.

Innerhalb dieser Anwendung lassen sich vielfältige Einstellungen vornehmen, welche von LAN-Einstellungen wie z. B. den iPv4-Adressen, über allgemeinen Einstellungen wie Zugangspasswörtern und Wi-Fi-Konfigurationen bis hin zu den eigentlichen LoRa-Einstellungen reichen. Innerhalb des LoRa-Reiters sind beispielsweise Parameter zum Packet Forwarder, LNS-IP-Adressen, Port-Nummern, Frequenz- und Radiokanälen, Keep-Alive-Traffic-Intervallen und Datenprotokolle der LoRa-Pakete einstell- bzw. einsehbar. Insbesondere die

Informationen über den Datenverkehr sind zur Inbetriebnahme der Sensoren und des Gateways innerhalb des LoRaWAN-Netzwerks sehr hilfreich.

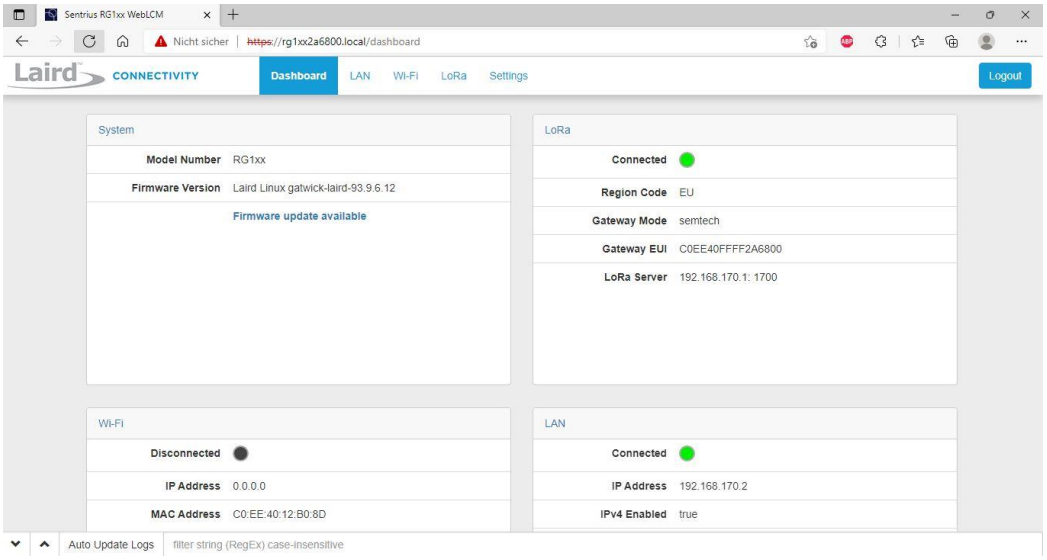


Abbildung 31: Dashboard des Webinterfaces des Laird-Gateways

Kerlink – Wirnet iFemtoCell-evolution

Der Zugang zu den offiziellen Dokumentationen muss über ein Kontaktformular auf der Kerlink-Seite beantragt werden. Nach Prüfung erhält man die Zugangsdaten für das Wiki von Kerlink [34], welches neben einigen Download-optionen auch den Zugang zu einer Vielzahl von weiteren Artikeln bezüglich der Gateways enthält.

Das betrachtete Gerät bietet ebenfalls die Möglichkeit eines Webinterfaces, welches über eine entsprechende URL oder IP-Adresse erreicht wird. (siehe Abbildung 32).

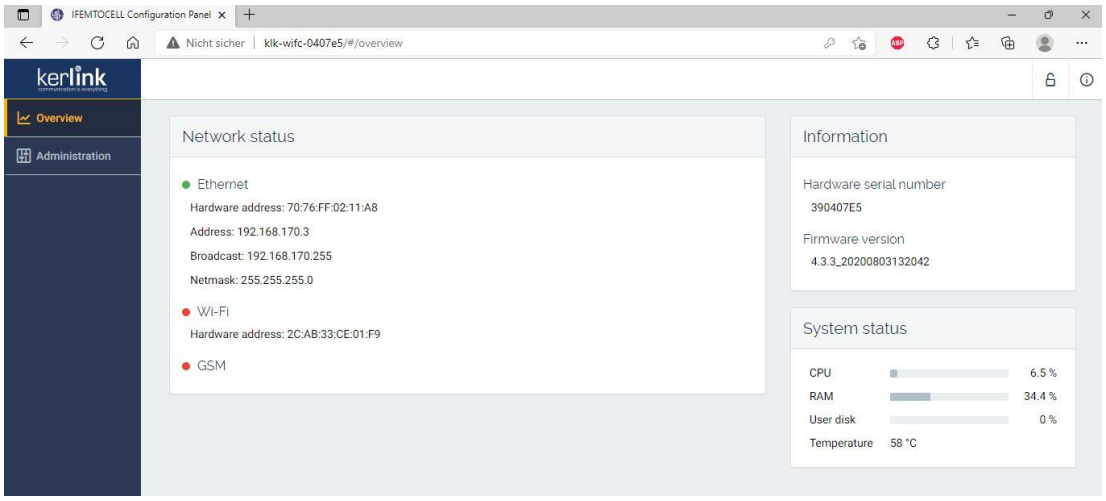
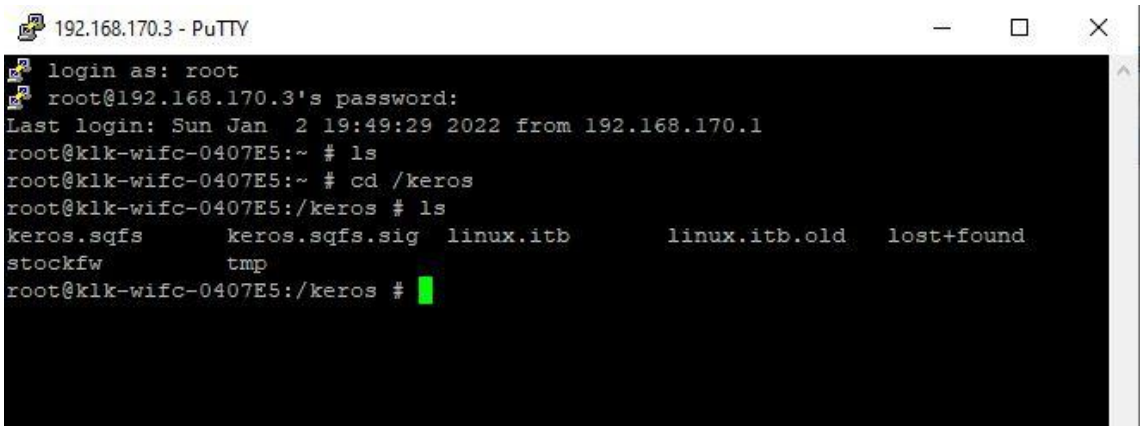


Abbildung 32: Übersichtsseite vom Webinterface des Kerlink-Gateways

Die Konfigurationsmöglichkeiten innerhalb des Webinterfaces beschränken sich allerdings auf rudimentäre Einstellungen wie Passwörter, Firmware-Updates und Reset-Optionen. Ein Großteil der Einstellungen bezüglich des LoRaWAN-Netzwerkes wie die Parametrierung des Packet Forwarders, des LNS und der Frequenzen kann über eine SSH-Verbindung mittels einer entsprechenden Software erfolgen. Im Rahmen dieser Arbeit wird hierfür PuTTY verwendet, sodass mit dem entsprechenden Passwort eine Root-Verbindung zum Linux-basierten KerOS-Betriebssystem hergestellt werden kann (siehe Abbildung 33a). Um Daten mit einer graphischen Oberfläche austauschen und bearbeiten zu können, ist auch die Verwendung eines SFTP-Clients möglich. Hierfür wird WinSCP verwendet, worüber textbasiert Änderungen an den Konfigurations-dateien vorgenommen werden können (siehe Abbildung 33b).



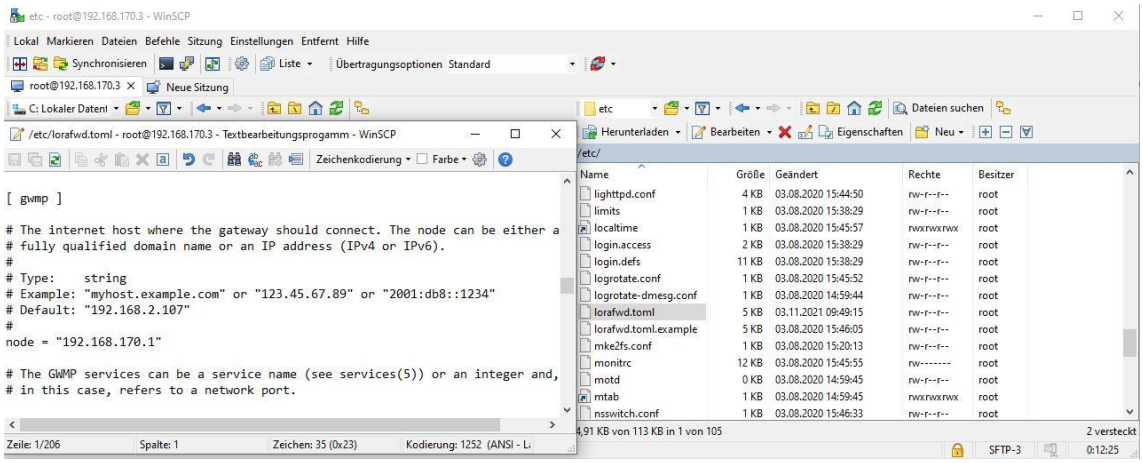


Abbildung 33a und 33b: Putty-Screenshot mit Root-Verbindung zum Kerlink-Gateway (oben) sowie WinSCP-Screenshot mit aufgerufener LNS-Konfigurationsdatei (unten)

3.2 Entscheidung für Netzwerkarchitektur

In dem zu evaluierendem Anwendungsfall ist es zunächst erforderlich, eine Entscheidung über die zu verwendende Netzwerkarchitektur zu treffen. Die drei Möglichkeiten zur Umsetzung wurden im Unterkapitel 2.2.7 vorgestellt. Im Rahmen einer Projektbesprechung fällt die Wahl mit dem Ziel einer maximal unabhängigen Software und der Einschränkung, dass die Gebäude-automatisierungs-SPS nicht in das Internet eingebunden werden soll, auf die Entwicklung eines eigenen LNS. Dabei ist gemäß der Abbildung 26 in dem angesprochenen Kapitel das Semtech UDP-Protokoll sowie das gesamte Netzwerkmanagement inklusive der Berechnung der Sitzungsschlüssel zu entwickeln. Um die Machbarkeit dieses Vorhabens zu prüfen, wird zunächst eine Windows-basierte Desktop-Applikation entwickelt.

3.3 Entwicklung der Desktop-Applikation

Die Entwicklung der Desktop-Applikation wird von Prof. Dr. Jochen Maaß mittels Qt übernommen. Dabei muss das von Semtech entwickelte Packet Forwarder-UDP-Protokoll [35] implementiert werden. Darüber hinaus werden von der Applikation die Berechnung der Sitzungsschlüssel mittels AES-CMAC-Algorithmus, die Abhandlung der Join-Prozedur von OTAA-Sensoren und die Handhabung in Bezug auf den Keep-Alive-Traffic und den Data Up-Nachrichten gemäß LoRaWAN-Spezifikation [12] behandelt.

Während der Entwicklung dient der in Abbildung 34 dargestellte Testaufbau aus den vier Sensoren, den beiden Gateways und einem Laptop als Referenz. Mit Hilfe der Analyse der Datenprotolle mittels Wireshark und den unter 3.1 beschriebenen Debug- und Konfigurationsmöglichkeiten der jeweiligen Geräte kann unter Beteiligung des Autors dieser Arbeit eine Applikation für Windows-PCs erfolgreich entwickelt werden. Eine Ansicht dieser ist in Abbildung 35 dargestellt. Dabei wird zwischen dem Gateway-Protokoll, dem LoRaWAN-Netzwerk und der Applikationsschicht unterschieden. In der Applikationsschicht sind die Nutzdaten der Sensoren mit einigen anderen Metadaten zu erkennen.

Abbildung 34: Testaufbau zur Entwicklung der Desktop-Applikation

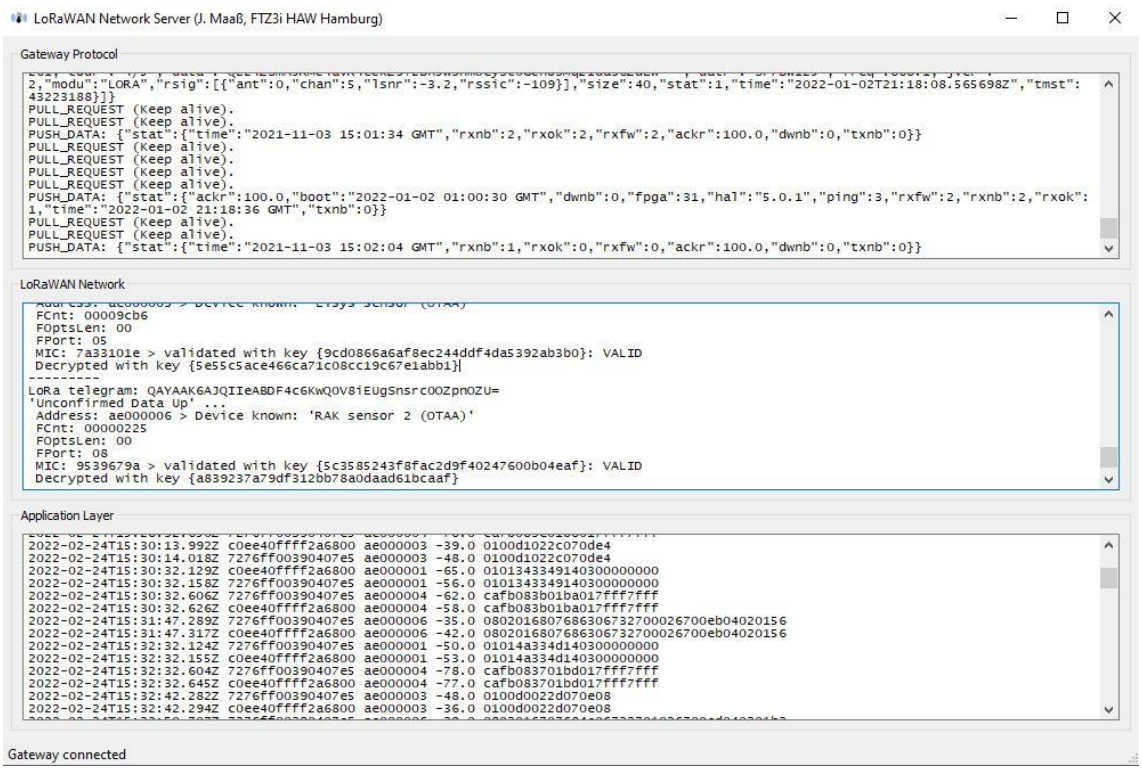


Abbildung 35: Ansicht der Desktop-Applikation

3.4 Möglichkeiten der Implementierung auf einer SPS

Nachdem die Entwicklung der Desktop-Applikation abgeschlossen ist und als Vorlage dient, stellt sich die Frage nach den Implementierungsmöglichkeiten auf einer SPS mit den dort zur Verfügung stehenden Bordmitteln. Dabei ergeben sich grundsätzlich zwei Möglichkeiten, welche im Folgenden vorgestellt werden sollen.

3.4.1 Komplettimplementierung

Eine Möglichkeit der Umsetzung auf einer SPS besteht darin, den gesamten LNS zu implementieren. Dabei muss das gesamte Netzwerkmanagement inklusive des OTAA-Join-Prozesses und der Sitzungsschlüsselberechnung umgesetzt werden. Vorteilhaft an dieser Variante ist, dass keine weitere Hardware zum Aufbau eines LoRaWAN-Netzwerkes benötigt wird. Allerdings ist die Umsetzung aller unter 3.3 beschriebenen und schlussendlich in der Desktop-Applikation entwickelten Funktionen mit den zur Verfügung stehenden Mitteln einer SPS sehr aufwändig.

3.4.2 Minimalimplementierung

Es besteht außerdem die Möglichkeit einer Minimalimplementierung der Netzwerkschicht auf einer SPS. Dabei werden im konkreten Anwendungsfall die Join-Prozedur und das Berechnen der Sitzungsschlüssel von der Desktop-Applikation übernommen. Der dabei generierte AppSKey muss anschließend auf die SPS übertragen werden, sodass ein nicht unwesentlicher Teil des Netzwerkmanagements entfällt. Das Behandeln der UDP-Pakete mit den enthaltenen JSON-Dateien, die Entschlüsselung der Nutzdaten und das sensorspezifische Decodieren der Nutzdaten und damit der Applikationsserver muss in beiden Fällen auf der SPS erfolgen. Den durch diese Variante eingesparten Arbeitsmehraufwand steht allerdings die Tatsache gegenüber, dass bei der Wahl der Minimalimplementierung ein zusätzlicher Laptop und die Desktop-Applikation zum Hinzufügen eines Sensors zu einem LoRaWAN-Netzwerk benötigt werden.

3.5 Entscheidung

Nach Abwägen der jeweiligen Vor- und Nachteile beider Implementierungsmöglichkeiten im Rahmen einer Projektbesprechung und zur genauen Definition der Aufgabenstellung im Zusammenhang mit dieser Thesis, wird sich für die Variante der Minimalimplementierung entschieden. Zur Evaluierung einer Anbindungsmöglichkeit von LoRa-Sensoren an eine SPS wird diese Vorgehensweise als zweckdienlicher und daher als ausreichend angesehen. Damit dient die entwickelte Desktop-Applikation in Teilen als Vorlage aber auch als Ergänzung für die Umsetzung des Arbeitsauftrages.

4. Programmierung

Nachdem die Entscheidung zur Umsetzung des SPS-seitigen LoRaWAN-Netzwerkes auf die Minimalimplementierung gefallen ist, kann mit der eigentlichen Programmierung begonnen werden. Hierzu wird sich zunächst für eine Entwicklungsumgebung entschieden, bevor mit der Definition der einzelnen Arbeitspakete die Verwendung verschiedener Bibliotheken geplant wird und anschließend die eigentliche Codierung beginnt. Diese Vorgehensweise wird in diesem Kapitel ausführlich beleuchtet, wobei innerhalb der Beschreibung zur Generierung des Codes Programmauszüge der finalen Softwareversion verwendet werden. Diese finale Version ist auf der sich im Anhang befindlichen CD abgelegt und kann bei Bedarf eingesehen werden.

4.1 Wahl der Entwicklungsumgebung

Im Rahmen des Gesamtprojektes sollen verschiedene Speicherprogrammierbare Steuerungen eingesetzt werden können, um eine maximale Unabhängigkeit von einzelnen Produzenten zu gewährleisten. Mit diesem Ziel ist die Notwendigkeit verbunden, dass zu codierende Programm mit einer herstellerunabhängigen Software-Plattform zu entwickeln. Dabei ist die Wahl auf die CODESYS-Entwicklungsumgebung gefallen und im Rahmen dieser Arbeit wird die Version V3.5 SP 17 Patch 1 (V3.5.17.10) verwendet, welche unter [36] kostenfrei zum Download zur Verfügung steht.

CODESYS ist eine Software-Plattform für die industrielle Automatisierungstechnik, dessen Basis das IEC-61131-3 Programmiertool „CODESYS Development System“ bildet. Innerhalb des Schichtenmodells des Automatisierungssystems wird zwischen der Geräteebene, der Kommunikationsebene und der Projektierungsebene unterschieden. Mit der Verwendung des Laufzeitmodells kann eine SPS auf der Geräteebene simuliert werden, welche der Kommunikationsserver mit dem Entwicklungssystem auf der Projektierungsebene verbindet [38].

CODESYS ist nach eigenen Angaben bei über 400 Herstellern im Einsatz und damit das führende, herstellerunabhängige IEC-61131-3 Programmierwerkzeug (Stand: 28.02.2022) [37].

4.2 Herangehensweise & Definition der Arbeitspakete

Bei der gewählten Minimalimplementierung des LNS ist es notwendig, wie in Kapitel 3.4.2 beschrieben, die LoRa-Sensoren mittels der entwickelten Desktop-Applikation einem LoRaWAN-Netzwerk zuzuordnen. Die bei der dabei behandelten Join-Prozedur der OTAA-Endknoten berechneten geheimen Sitzungsschlüssel sowie die zugeordneten Geräteadressen (DevAddr) müssen dann auch der SPS bekannt gemacht werden. Alle weiteren beschriebenen Funktionen der Netzwerkschicht und der Anwendungsschicht müssen auf der SPS implementiert werden.

Zur besseren Verwirklichung des Arbeitsauftrages ist eine strukturierte Herangehensweise unabdingbar. Daher werden zunächst gemäß den umzusetzenden Funktionalitäten modularisierte Arbeitspakete definiert, welche im Folgenden in der benötigten Reihenfolge aufgelistet werden:

- allgemeines Aufsetzen des Projektes inkl. Parametrierung der Ethernet-Schnittstelle
- UDP-Pakete der Gateways empfangen bzw. an diese senden können
- Behandlung des Keep-Alive-Traffics zwischen Gateways und LNS
- Behandlung des Upstream-Protokolls, insbesondere PUSH_DATA-Pakete
- anwendungsbezogenes Parsen der JSON-Daten
- BASE64-Decodierung der Daten
- Auswertung der Physikalischen Nutzdaten (PHY-Payload, siehe auch Abbildung 17)
- Auswertung der MAC-Nutzdaten von Datennachrichten (vgl. Abschnitt 2.2.6)
- Decodierung der Nutzdaten mit AES-128-Algorithmus
- sensorspezifische Auswertung/Interpretation der Daten

Im Rahmen einiger der angesprochenen Punkte ist das Herunterladen, Einbinden und die Nutzung von Bibliotheken notwendig. Darüber hinaus werden manche der Funktionalitäten in eigene Funktionsbausteine ausgegliedert, was zum einen aufgrund der speichernden Eigenschaften dieses Bausteintyps alternativlos ist, aber zum anderen auch der besseren

Übersichtlichkeit halber im Hauptprogramm erfolgt. Im folgenden Abschnitt wird die Umsetzung der genannten Arbeitspakete detailliert erörtert.

4.3 Umsetzung und Codierung des SPS-Programms

Bei der Behandlung der einzelnen Arbeitspakete wird zunächst beschrieben, welche Funktionalitäten konkret implementiert werden müssen. Nachfolgend wird die Umsetzung als solches behandelt und abschließend die Verifikation des Funktionsumfangs durchgeführt.

4.3.1 Projektaufsetzung & Konfiguration der Ethernet-Schnittstelle

Gemäß der bereits mehrfach angesprochenen Aufgabenstellung und der damit einhergehenden Netzwerkarchitektur, die eine direkte kabelgebundene Verbindung von den Gateways und der SPS verlangt, ist im aufzusetzenden Projekt der bereits zur Entwicklung der Desktop-Applikation zum Einsatz gekommene Testaufbau (siehe auch Abbildung 34) zu verwenden. Dabei sind beide Gateways über RJ45-Schnittstellen mit einem Switch verbunden, welcher wiederum mit der entsprechenden Schnittstelle des Programmier-Laptops verbunden ist. Der Laptop, dessen Ethernet-Adapter die IP-Adresse 192.168.170.1 mit der Subnetzmaske 255.255.255.0 besitzt, simuliert im Rahmen dieser Arbeit die SPS und somit den Ort des LoRaWAN-Netzwerkservers. Der benutzte UDP-Port ist im Up- und Downlink-Fall mit 1700 gewählt worden. Die IP-Adresse und die genutzten Ports des LNS sind zunächst in beiden Gateways als Netzwerkserver-Adresse mit den in Kapitel 3.2.2 vorgestellten Konfigurationsmöglichkeiten zu hinterlegen.

Beim Aufsetzen des CODESYS-Standardprojektes wird ein entsprechend des verwendeten Betriebssystems passendes Gerät ausgewählt, welches mit der Verwendung des Laufzeitmodells die SPS simuliert. Diesem, dann im Projektbaum der Entwicklungsumgebung hinterlegten Gerät, kann über einen Kontextklick ein weiteres Gerät angehangen werden. Mittels dieser Option ist es möglich, dem Projekt eine entsprechende Ethernet-Schnittstelle als Feldbus hinzuzufügen (siehe Abbildung 36a). Diese Schnittstelle zwischen den Gateways und der SPS ist über das Auswahlmenü konfigurierbar. In den allgemeinen Einstellungen kann die Ethernet-Adapter-IP-Adresse und Subnetzmaske des

Laptops eingetragen werden (siehe Abbildung 36b), sodass die benötigte grundsätzliche Kommunikationsstruktur zwischen den Geräten aufgebaut ist.

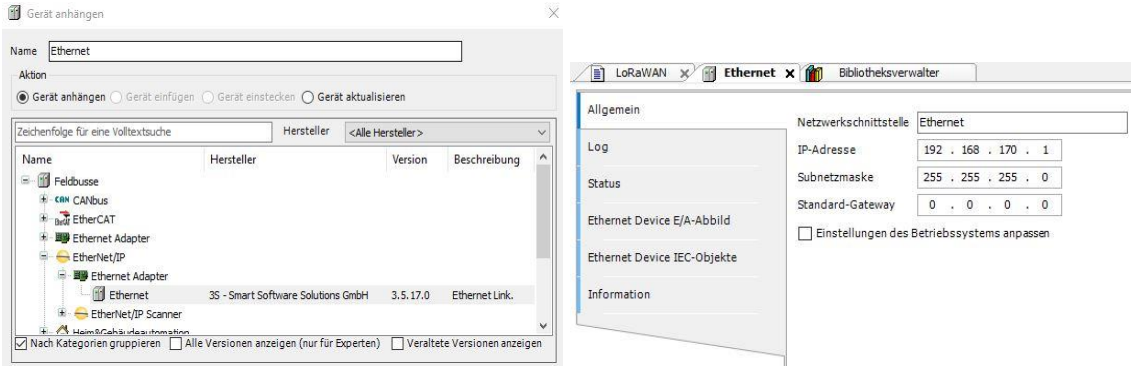


Abbildung 36a und 36b: Hinzufügen von Ethernet als Feldbussystem zum Projekt und Parametrierung der Schnittstelle

4.3.2 Kommunikation mit UDP-Paketen

Nachdem das Softwareprojekt aufgesetzt und die grundlegende Kommunikationsstruktur über die Ethernet-Schnittstelle eingerichtet ist, muss im nächsten Schritt das Empfangen und Versenden von UDP-Nachrichten behandelt werden. UDP ist ein minimales, verbindungsloses Netzwerkprotokoll, welches in IP-basierten Netzwerken den Versand von Datagrammen ermöglicht. Dabei ist es im Vergleich zu TCP ein unzuverlässiges Transportprotokoll, da keine Empfangsbestätigungen für Pakete gesendet werden und jederzeit UDP-Datagramme verloren gehen können. Dies hat aber auch Vorteile, da z. B. das Fehlen von Paketen bei TCP zu einem ständigen Nachfordern fehlender Daten führt und somit die Übertragungen nicht mehr so energieeffizient sind [39]. Daher hat die Firma Semtech UDP als Transportprotokoll für die Verwendung in dem ersten entwickeltem Packet Forwarder verwendet, welcher heute noch in nahezu allen verfügbaren Gateways vorinstalliert ist und daher auch als LoRaWAN-Standard bezeichnet werden kann.

Zum Aufbau einer UDP-basierten Kommunikation ist ein sogenannter UDP Peer nötig, welcher innerhalb einer von CODESYS zur Verfügung gestellten Bibliothek namens *Net Base Services* (kurz: NBS) implementiert ist. Diese Bibliothek kann im Bibliotheksverwalter über den Reiter „Bibliothek hinzufügen“ und der entsprechenden Volltextsuche dem Projekt hinzugefügt werden. Dabei werden die Funktionsbausteine *UDP_Peer*, *UDP_Receive* sowie

IPv4Address verwendet. Ein Peer kapselt alle notwendigen Dinge zum Senden und Empfangen von Daten über das UDP-Protokoll. In folgender Abbildung ist die Übersicht der Ein- und Ausgänge des FBs dargestellt.

InOut:

Scope	Name	Type	Comment	Inherited from
Input	xEnable	BOOL	TRUE: Activates the defined operation FALSE: Aborts/resets the defined operation	LConC
Output	xBusy	BOOL	TRUE: Operation is running	LConC
	xError	BOOL	TRUE: Error condition reached	LConC
	eErrorID	ERROR		LConC
Input	itfAsyncProperty	IAsyncProperty		
	itfIPAddress	IIPAddress	IP Address of the specific Adpater for Receiving/Sending (The related Networkmask and Broadcast adresses for receiving will be calculated)	
	uiPort	UINT	Receive Port	
	itfMulticast	IIPAddressSet	One or more Multicast Groups (optional)	
Output	xActive	BOOL	TRUE if the peer context is established	
	itfPeer	IPeer	The established context of this peer Valid, as long xActive = TRUE	

- UDP_Peer.Receive (METH)
- UDP_Peer.Send (METH)
- UDP_Peer.SetInitialValue (METH)

Abbildung 37: CODESYS-Dokumentation des FB UDP_Peer aus der Net Base Services Bibliothek

Über den Eingang *uiPort* kann der Empfangs-Port, welcher wie zuvor erwähnt auf 1700 festgelegt wurde, eingestellt werden. Darüber hinaus wird über *itfIPAdress* die IP-Adresse vorgegeben. Hierzu wird wiederum der FB IPv4Address eingesetzt, welcher mit seiner implementierten Methode *IPv4Address.SetInitialValue* die Möglichkeit einer dynamischen Initialisierung einkapselt.

Um entsprechende Daten im UDP-Format zu erhalten und einige Nebeninformationen wie z. B. Anzahl der zu lesenden Bytes oder der Sender-IP-Adresse zu speichern, wird der Funktionsbaustein UDP_Receive verwendet, dessen Ein- und Ausgänge in Abbildung 38 dargestellt sind. Der Pointer *pData* gibt den Speicherbereich an, in dem die Daten geschrieben werden sollen, während *udiSize* die maximale Anzahl der zu lesenden Bytes definiert. Über den Eingang *itfPeer*, welcher vom Typ *IPeer*, also einem Interface, ist, kann der zuvor initialisierte FB UDP_Peer (hat das Interface IPeer implementiert) mit dem FB UDP_Receive verbunden werden. Wenn Daten erfolgreich gelesen werden, wird die boolesche Ausgangsvariable *xReady* auf TRUE gesetzt und die Variable

udiCount zeigt die tatsächlich gelesene Anzahl von Bytes an. Außerdem können über die Variablen *itfIPAddressFrom* und *uiPortFrom* die IP-Adresse und die Portnummer des Gerätes abgefragt werden, welches das Datagramm übermittelt hat.

InOut:

Scope	Name	Type	Comment	Inherited from
Input	xEnable	BOOL	TRUE: Activates the defined operation FALSE: Aborts/resets the defined operation	LConC
Output	xBusy	BOOL	TRUE: Operation is running	LConC
	xError	BOOL	TRUE: Error condition reached	LConC
	eErrorID	ERROR		LConC
Input	itfPeer	IPeer		
	pData	__XWORD		
	udiSize	UDINT		
Output	xReady	BOOL		
	itfIPAddressFrom	IIPAddress		
	uiPortFrom	UINT		
	udiCount	UDINT		

Abbildung 38: CODESYS-Dokumentation des FBs UDP_Receive

Da es für ein komplettes LoRaWAN-Netzwerk ebenfalls nötig ist, UDP-Pakete vom LNS an die Gateways zu senden, wird außerdem die im FB UDP_Peer enthaltene Methode UDP_Peer.Send verwendet. Diese kann über vier Eingänge parametriert werden und über einen Ausgang die tatsächliche Anzahl der gesendeten Bytes ausgeben (siehe Abbildung 39).

UDP_Peer.Send (METH)

METHOD Send : ERROR

InOut:

Scope	Name	Type
Return	Send	ERROR
Input	itfIPAddress	IIPAddress
	uiPort	UINT
	pData	__XWORD
	udiSize	UDINT
Output	udiCount	UDINT

Abbildung 39: CODESYS-Dokumentation der Methode UDP_Peer.Send

Die korrekte Funktionalität dieser UDP-Kommunikationsstruktur wird zunächst mit dem kostenlosen, von Microsoft zur Verfügung gestellten Entwicklungstool *UDP – Sender/Receiver* verifiziert. Nachdem diese Testläufe nach einigen

Anpassungen am Code erfolgreich abgeschlossen sind, wird diese Struktur innerhalb der LoRaWAN-Anwendung implementiert. Dazu werden die entsprechenden Variablen angelegt und gemäß den Anforderungen im Programmcode verknüpft.

```
PROGRAM LoRaWAN
VAR
    udpPeer : NBS.UDP_Peer;
    ipAddress : NBS.IPv4Address;

    initialized : BOOL := FALSE;
    udpReceiver : NBS.UDP_Receive;
    receivedMessageByteArray: ARRAY[0..350] OF BYTE;
    newPacketArrived : BOOL := FALSE;
    packetSizeReceived : UDINT;
    packetSizeSend : UDINT;
```

```
IF NOT initialized THEN
    udpPeer.uiPort := 1700;
    udpPeer.xEnable := TRUE;
    ipAddress.SetInitialValue('192.168.170.001');
    udpPeer.itfIPAddress := ipAddress;
    initialized := TRUE;

    udpReceiver.itfPeer := udpPeer;
    udpReceiver.pData := ADR(receivedMessageByteArray);
    udpReceiver.udiSize := SIZEOF(receivedMessageByteArray);
    udpReceiver.xEnable := true;
END_IF

udpPeer();
udpReceiver();
```

Abbildung 40a und 40b: Variablen zum Aufsetzen der UDP-Kommunikation (oben) sowie Konfiguration dieser im Programmcode (unten) (Codeauszüge)

Für die zu empfangenden Daten wird ein Byte-Array von ausreichender Größe angelegt und der Daten-Pointer pData des FBs UDP_Receive auf diesen Adressbereich verlinkt. Mit der Software Wireshark und dem angelegten Byte-Array kann unter Zuhilfenahme von Haltepunkten die korrekte Funktion der aufgesetzten UPD-Kommunikation verifiziert werden. In Abbildung 41 ist das Paket 264, welches Wireshark als UDP-Paket von der IP-Adresse 192.168.170.3 (Kerlink Gateway) auf die IP-Adresse 192.168.170.1 und den Port 1700 aufgezeichnet hat, in Teilen dargestellt. Dieses Paket umfasst insgesamt 323

Bytes, wovon 281 Bytes als Daten deklariert sind. Die ersten 24 Bytes der Daten sind *02 5b c4 00 72 76 ff 00 39 04 07 e5 7b 22 72 78 70 6b 22 3a 5b 7b 22 61*. Dieselbe Bytefolge lässt sich in Abbildung 43 nachlesen, welche ein Auszug aus der Variablentabellen des CODESYS-Programms ist, in der die Aktualwerte angezeigt werden.

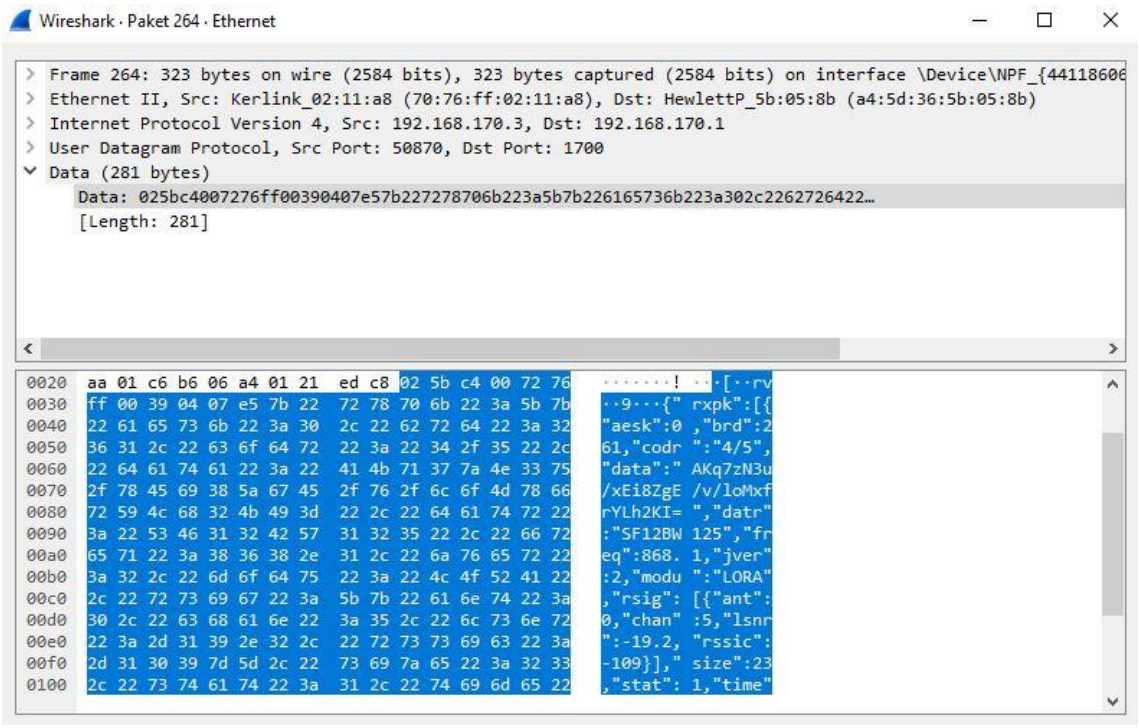


Abbildung 41: Wireshark-Aufzeichnung eines UDP-Paketes

Über die Ausgangsvariable udiSize des Funktionsbausteins UDP_Receive (vgl. auch Abbildung 38) wird die empfangene Paketgröße abgefragt und in einer entsprechenden Variablen gespeichert. Diese Funktion ist in folgender Abbildung 42 dargestellt, welcher aufgrund der aktiven Online-Verbindung ebenfalls die aktuellen Werte der Variablen darstellt. Die als Paketgröße angezeigte 119 ist die hexadezimale Darstellung der Dezimalzahl 281 ($1 \cdot 16^2 + 1 \cdot 16^1 + 9 \cdot 16^0$).

```
17  IF udpReceiver.xReady FALSE THEN
18      packetSizeReceived 16#00000119 := udpReceiver.udiCount 16#00000000 ;
19      newPacketArrived TRUE := TRUE;
```

Abbildung 42: Bestimmung der erhaltenen Paketgröße (Codeauszug)

Device.Application.LoRaWAN

Ausdruck	Datentyp	Wert
receivedMessageByteArray	ARRAY [0..350] OF ...	
receivedMessageByteArray[0]	BYTE	16#02
receivedMessageByteArray[1]	BYTE	16#5B
receivedMessageByteArray[2]	BYTE	16#C4
receivedMessageByteArray[3]	BYTE	16#00
receivedMessageByteArray[4]	BYTE	16#72
receivedMessageByteArray[5]	BYTE	16#76
receivedMessageByteArray[6]	BYTE	16#FF
receivedMessageByteArray[7]	BYTE	16#00
receivedMessageByteArray[8]	BYTE	16#39
receivedMessageByteArray[9]	BYTE	16#04
receivedMessageByteArray[10]	BYTE	16#07
receivedMessageByteArray[11]	BYTE	16#E5
receivedMessageByteArray[12]	BYTE	16#7B
receivedMessageByteArray[13]	BYTE	16#22
receivedMessageByteArray[14]	BYTE	16#72
receivedMessageByteArray[15]	BYTE	16#78
receivedMessageByteArray[16]	BYTE	16#70
receivedMessageByteArray[17]	BYTE	16#6B
receivedMessageByteArray[18]	BYTE	16#22
receivedMessageByteArray[19]	BYTE	16#3A
receivedMessageByteArray[20]	BYTE	16#5B
receivedMessageByteArray[21]	BYTE	16#7B
receivedMessageByteArray[22]	BYTE	16#22
receivedMessageByteArray[23]	BYTE	16#61

Abbildung 43: Auszug der Variablentabelle im eingeloggten Zustand mit den ersten 24 Bytes des Beispielpakets

Anhand des dokumentierten, exemplarischen UDP-Pakets kann die korrekte Funktion der programmierten Kommunikationsstruktur nachgewiesen werden. Als Ergebnis dieses Arbeitspaketes ist festzuhalten, dass der entstehende LNS in der Lage ist, von beiden Gateways entsprechende UDP-Pakete über die parametrierte Ethernet-Schnittstelle zu empfangen und diese korrekt in einem Byte-Array abzulegen.

4.3.3 Keep-Alive-Traffic

Die empfangenen UDP-Datagramme unterscheiden sich signifikant in ihrer Paketgröße. Ein wichtiger Teil der Kommunikation zwischen dem Gateway und dem LNS ist der sogenannte Keep-Alive-Traffic. Das Gateway sendet je nach

Konfiguration zyklisch nach einer festen Zeitspanne ein sogenanntes *PULL_DATA*-Paket, auf das der Server mit einem *PULL_ACK*-Paket antworten muss [35]. Diese, in Abbildung 44 dargestellte, Kommunikationssequenz wird in diesem Abschnitt in das LoRaWAN-Software-Projekt eingebettet.

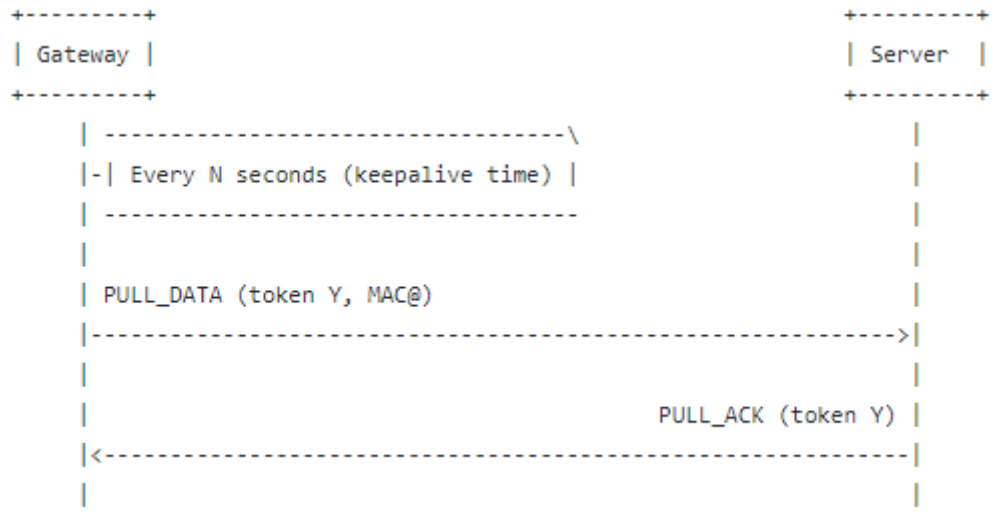


Abbildung 44: Sequenzdiagramm des Keep-Alive-Traffics [35][16]

Das PULL_DATA-Paket ist ein Pakettyp, welches vom Gateway genutzt wird, um Daten vom Server abzurufen. Dieser Datenaustausch wird vom Gateway initiiert, um zu überprüfen, ob das Netzwerk erreichbar ist. Ein PULL-DATA-Paket besitzt stets eine Größe von 12 Bytes, in der eine 1-Byte lange Protokoll-Version, 2-Bytes lange Zufallstoken, dem PULL-DATA-Identifizierungsbyte sowie der 8-Byte langen MAC-Adresse des Gateways gemäß dem Semtech-UDP-Protokoll enthalten sind (siehe auch Abbildung 45) [35].

Bytes	Function
0	protocol version = 2
1-2	random token
3	PULL_DATA identifier 0x02
4-11	Gateway unique identifier (MAC address)

Abbildung 45: Zusammensetzung des PULL_DATA-Pakets zur Realisierung des Keep-Alive-Traffics [35][17]

[16] Kapitel 5: *Downstream protocol*
[17] Kapitel 5.2: *PULL_DATA packet*

Dem Sequenzdiagramm des Keep-Alive-Traffics ist zu entnehmen, dass nach einem gesendeten PULL_DATA-Pakets das Gateway eine Antwort in Form eines passendem PULL_ACK-Pakets des zu programmierenden LNS erwartet. Dabei beinhaltet das Antwortpaket dieselben Zufallstoken, sowie neben der Protokollversion auch das entsprechende Identifizierungsbyte. Die genaue Zusammensetzung des PULL_ACK-Paketes ist in folgender Abbildung dargestellt.

Bytes	Function
0	protocol version = 2
1-2	same token as the PULL_DATA packet to acknowledge
3	PULL_ACK identifier 0x04

Abbildung 46: Zusammensetzung des PULL_ACK-Pakets zur Realisierung des Keep-Alive-Traffics [35][18]

Demnach sind für das Erkennen eines vom Gateway initiiertem Keep-Alive-Traffics zwei Parameter maßgeblich. Zum einen ist das die Länge des empfangenen Pakets und zum anderen ist das das PULL_DATA-Identifizierungsbyte. Darüber hinaus kann auch die MAC-Adresse der eigenen Gateways miteinbezogen werden, worauf allerdings im Rahmen dieses Projektes aus Gründen des fehlenden Mehrwerts verzichtet wird. Mittels der bereits in Abbildung 42 eingeführten Integer-Variablen *packetSizeReceived*, welche über die Ausgangsvariablen udiCount des FBs UDP_Receive zugewiesen wird, kann in einer bedingten Anweisung die Größe des Keep-Alive-Traffic Paketes von 12 Bytes abgefragt werden. Außerdem wird die Paketgröße mit dem Identifizierungsbyte verundet, sodass beide angesprochenen Parameter abgefragt werden (siehe Abbildung 47).

```
24      IF packetSizeReceived=12 AND receivedMessageByteArray[3]=2 THEN
25          KeepAliveTraffic:=TRUE;
26          returnMessageByteArray[0]:=receivedMessageByteArray[0];
27          returnMessageByteArray[1]:=receivedMessageByteArray[1];
28          returnMessageByteArray[2]:=receivedMessageByteArray[2];
29          returnMessageByteArray[3]:=4;
30          ELSE KeepAliveTraffic:=FALSE;
31      END_IF
```

Abbildung 47: Handhabung des Keep-Alive-Traffics (Codeauszug)

[18] Kapitel 5.3: *PULL_ACK packet*

Wird ein ankommendes UDP-Paket als PULL_DATA-Paket erkannt, so muss der LNS eine gemäß dem Semtech-UDP-Protokoll festgelegte Antwort formulieren. Das PULL_ACK-Paket ist demnach stets 4-Byte lang und wird im Programmcode ebenfalls über ein Byte-Array repräsentiert, welches in den Zeilen 26 bis 29 gemäß der Spezifikation des Kommunikationsprotokolls (vgl. Abbildung 46) befüllt wird. Zum Senden eines UDP-Paketes mit diesem Inhalt wird an dieser Stelle die bereits unter 4.3.2 angesprochene Methode UDP_Peer.Send verwendet. Dabei werden die benötigten vier Eingangsparameter der Methode entsprechend zugewiesen, sodass die Antwort an die IP-Adresse und den Port gesendet wird, von dem das PULL_DATA-Paket gesendet wurde. Diese Parameter sind über die Ausgänge uiPortFrom und itfIPAdressFrom des FBs UDP_Receive zugänglich. In Abbildung 48 ist in den dargestellten Codezeilen 153 und 154 exakt diese Vorgehensweise nachzuvollziehen. Darüber hinaus wird das befüllte Antwort-Byte-Array als Adresse dem Datenpointer pData der Methode übergeben (Zeile 155), während in Zeile 156 die Anzahl der zu sendenden Bytes in Abhängigkeit von der Größe des Byte-Arrays zugewiesen wird.

```
153     udpPeer.Send(   itfIPAddress:= udpReceiver.itfIPAddressFrom,
154                     uiPort:= udpReceiver.uiPortFrom,
155                     pdata:=ADR(returnMessageByteArray),
156                     udiSize:=SIZEOF(returnMessageByteArray));
```

Abbildung 48: Senden des PULL_ACK-Pakets (Codeauszug)

Zur Überprüfung der korrekten Funktion des implementierten Keep-Alive-Traffics wird erneut die Software Wireshark herangezogen, welche die beiden entsprechenden PULL_DATA- und PULL_ACK-Pakete aufzeichnen kann und sich somit ideal zu Debug-Zwecken eignet. In Abbildung 49 ist der Keep-Alive-Traffic ausgehend vom Kerlink-Gateway (192.168.170.3) zum LNS (192.168.170.1) und wieder zurück dargestellt. Über die sich im Info-Feld der Pakete befindlichen Länge der übermittelten Daten und den dort dargestellten Portnummern ist grundsätzlich eine korrekte Kommunikation verifizierbar.

No.	Time	Source	Destination	Protocol	Length	Info
58	15.375884	192.168.170.3	192.168.170.1	UDP	60	47987 → 1700 Len=12
59	15.386908	192.168.170.1	192.168.170.3	UDP	46	1700 → 47987 Len=4

Abbildung 49: Wireshark-Aufzeichnung des Keep-Alive-Traffics

Zur genaueren Betrachtung der Zusammensetzung der beiden Pakete können weitere Informationen über einen Doppelklick angezeigt werden. Diese sind in den folgenden Abbildungen mit jeweils blau markierten Datenfeldern dargestellt.

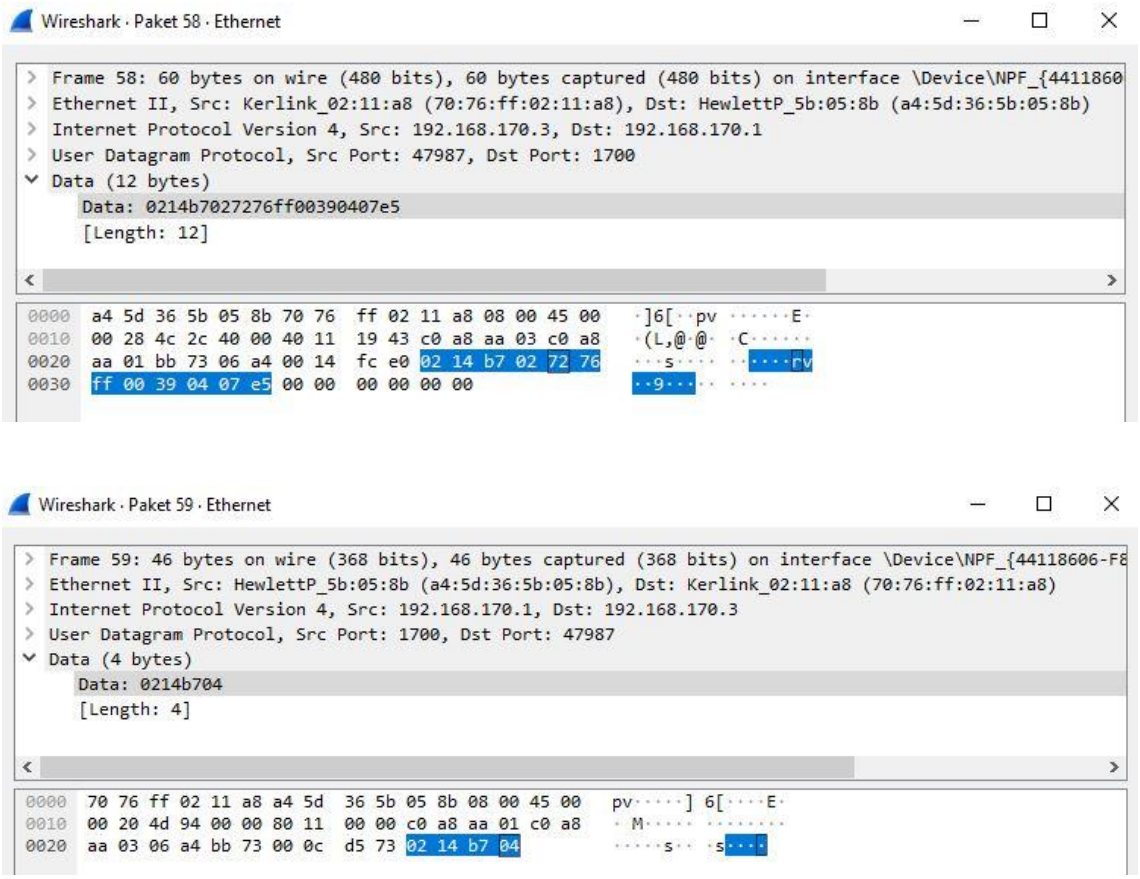


Abbildung 50a und 50b: Wireshark-Ansicht des PULL_DATA-Pakets (oben) sowie des PULL_ACK-Pakets (unten)

Vergleicht man die Wireshark-Ansicht des PULL_DATA-Pakets mit der Spezifikation des Semtech UDP-Protokolls (vgl. Abbildung 45), so erkennt man den Aufbau des Paketes deutlich. Neben der Größe des Pakets von 12 Byte ist im ersten Byte die Protokoll-Version 2 und nach den beiden Zufallstoken an dritter Position das Identifizierungsbyte *0x02* festzustellen. Anschließend folgt die 8-Byte lange MAC-Adresse des Kerlink-Gateways.

Es kann anhand der Wireshark-Ansicht des PULL_ACK-Pakets ebenfalls die korrekte Funktion des in Abbildung 48 dargestellten Codeauszuges zur Formulierung der Antwort nachgewiesen werden. Das Spiegeln der Protokoll-Version (Byte 1) sowie der beiden Zufallstoken (Byte 2 und 3) gelingt ebenso wie

das Hinzufügen des korrekten Identifizierungsbytes (Byte 4). Somit ist nach Beendigung dieses Arbeitspakets auch die Handhabung des Keep-Alive-Traffics und erstmals eine Kommunikation des entstehenden LNS mit dem Gateway erfolgreich umgesetzt.

4.3.4 Behandlung des Upstream-Protokolls

Ein weiterer wichtiger Pakettyp, welcher vom Gateway zum Server gesendet wird, sind die PUSH_DATA-Pakete, die Teil des Upstream Protokolls sind. Das in folgender Abbildung dargestellte Sequenzdiagramm spiegelt den Vorgang wider und lässt erkennen, dass der LNS auf dieses Paket mit einem entsprechenden PUSH_ACK-Paket antworten muss.

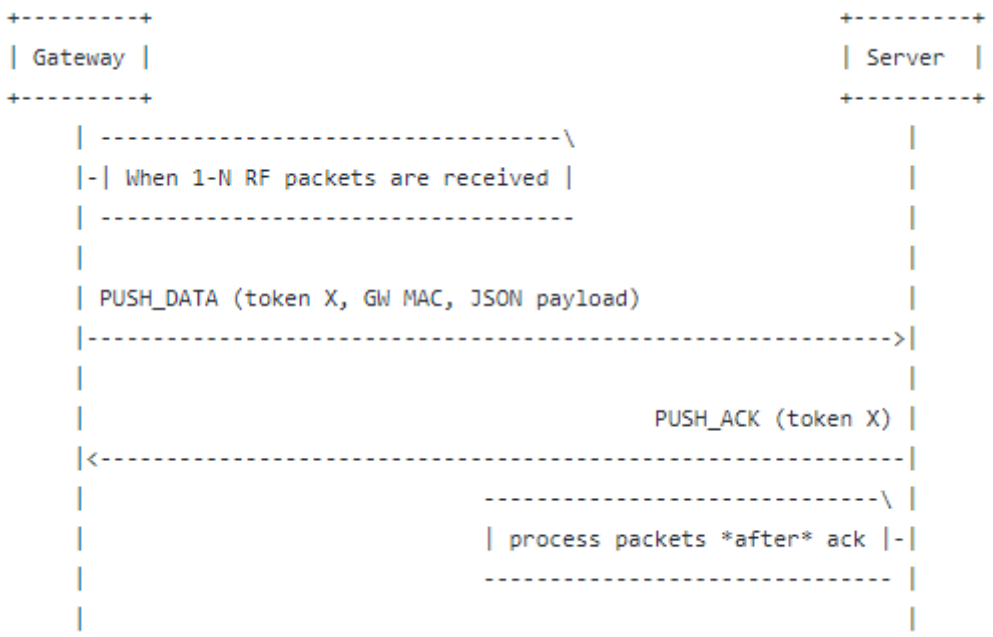


Abbildung 51: Sequenzdiagramm des Upstream-Protokolls [35][19]

Ein PUSH_DATA-Paket wird vom Gateway verwendet, um Datenpakete an den LNS zu senden, die neben den bereits vom PULL_DATA-Paket bekannten ersten 12 Bytes noch zusätzlich ein JSON-Objekt mitsendet, in dem Metadaten wie z. B. Frequenz und Datenrate aber auch die eigentlichen Nutzdaten der Sensoren enthalten sind. Da das Versenden dieser Metadaten je nach Gateway variiert und

[19] Kapitel 3.1.: *Sequence diagram*

auch die Sensoren-Nutzdaten unterschiedliche Längen besitzen, ist die Gesamtgröße des PUSH_DATA-Paketes sehr unterschiedlich. Über Wireshark wurde eine maximale Paketgröße von 291 Bytes festgestellt, in jedem Fall liegt diese aber über 12 Byte. Der konkrete Aufbau des Datagramms ist in folgender Abbildung 52 dargestellt.

Bytes	Function
0	protocol version = 2
1-2	random token
3	PUSH_DATA identifier 0x00
4-11	Gateway unique identifier (MAC address)
12-end	JSON object, starting with {, ending with }, see section 4

Abbildung 52: Zusammensetzung des PUSH_DATA-Pakets [35][20]

Ein wichtiger Parameter zur Identifizierung eines PUSH_DATA-Pakets ist demnach neben der Paketgröße das Identifizierungsbyte *0x00*, welches als viertes Byte übertragen wird.

Gemäß des dargestellten Sequenzdiagramms muss der LNS jedes Paket dieses Typs mit einem PUSH_ACK-Paket beantworten. Diese erfüllen die Funktion, dass der Netzwerkserver sofort den Erhalt jeglicher PUSH_DATA-Pakete bestätigt. In Abbildung 53 ist die genaue Struktur der Bestätigungsnachricht dargestellt. Ein solches Paket beinhaltet demnach neben der Protokollversion und zwei Zufallstoken erneut das entsprechende Identifizierungsbyte des Pakettyps, in diesem Fall das Byte *0x01*.

Bytes	Function
0	protocol version = 2
1-2	same token as the PUSH_DATA packet to acknowledge
3	PUSH_ACK identifier 0x01

Abbildung 53: Zusammensetzung des PUSH_ACK-Pakets [35][21]

[20] Kapitel 3.2: *PUSH_DATA packet*
[21] Kapitel 3.3: *PUSH_ACK packet*

Nachdem die Struktur des Upstream-Protokolls nachvollzogen ist, muss der beschriebene Mechanismus in das Softwareprojekt integriert werden. Zur Identifizierung des PUSH_DATA-Pakets werden die bereits angesprochenen zwei Parameter, die Länge und das Identifizierungsbyte, verwendet. Die Formulierung der PUSH_ACK-Nachricht wird, ähnlich der Vorgehensweise zur Behandlung des Keep-Alive-Traffics, über das Spiegeln der ersten drei Bytes und das Hinzufügen des Bytes *0x01* realisiert (siehe Abbildung 54).

```
37      IF packetSizeReceived>12 AND receivedMessageByteArray[3]=0 THEN
38          PushDataArrived:=TRUE;
39          returnMessageByteArray[0]:=receivedMessageByteArray[0];
40          returnMessageByteArray[1]:=receivedMessageByteArray[1];
41          returnMessageByteArray[2]:=receivedMessageByteArray[2];
42          returnMessageByteArray[3]:=1;
43          ELSE PushDataArrived:=FALSE;
```

Abbildung 54: Formulierung des PUSH_ACK-Pakets (Codeauszug)

Zum Senden der Nachricht wird erneut die bereits erfolgreich im Rahmen des Keep-Alive-Traffics implementierte Methode UDP_Peer.Send des verwendeten Funktionsbausteins genutzt (vgl. Abbildung 48). Mittels Wireshark kann analog zum vorherigen Abschnitt die korrekte Funktionalität des programmierten Mechanismus nachgewiesen werden. Somit ist an dieser Stelle im Entwicklungsprozess eine weitere und zunächst letzte Kommunikationsstruktur zwischen dem LNS und den Gateways realisiert.

4.3.5 JSON-Parser für die Nutzdaten

Nachdem das PUSH_DATA-Paket vom entstehendem Netzwerkserver bestätigt werden konnte, wird sich im Folgenden mit den in diesem Paket befindlichen Daten befasst. Wie bereits in Kapitel 2.3 erwähnt, verwendet das Semtech UDP-Protokoll die JSON-Datenstrukturen zur Weiterleitung der Sensordaten. Dabei ist für den betrachteten Anwendungsfall vor allem das Upstream-Protokoll von Interesse und in diesem Zusammenhang insbesondere die JSON-Dateien, welche das ´´rxpk´´-Array enthalten haben. Innerhalb dieses Arrays sind neben einigen Metadaten auch die eigentlichen Base64 verschlüsselten Daten enthalten, welche der Packet Forwarder der Gateways in dieses Feld eingetragen hat (siehe auch Abbildung 55).

Name	Type	Function
time	string	UTC time of pkt RX, us precision, ISO 8601 'compact' format
tmms	number	GPS time of pkt RX, number of milliseconds since 06.Jan.1980
tmst	number	Internal timestamp of "RX finished" event (32b unsigned)
freq	number	RX central frequency in MHz (unsigned float, Hz precision)
chan	number	Concentrator "IF" channel used for RX (unsigned integer)
rfch	number	Concentrator "RF chain" used for RX (unsigned integer)
stat	number	CRC status: 1 = OK, -1 = fail, 0 = no CRC
modu	string	Modulation identifier "LORA" or "FSK"
datr	string	LoRa datarate identifier (eg. SF12BW500)
datr	number	FSK datarate (unsigned, in bits per second)
codr	string	LoRa ECC coding rate identifier
rssi	number	RSSI in dBm (signed integer, 1 dB precision)
lsnr	number	Lora SNR ratio in dB (signed float, 0.1 dB precision)
size	number	RF packet payload size in bytes (unsigned integer)
data	string	Base64 encoded RF packet payload, padded

Abbildung 55: Bestandteile des rxpk-Arrays eines JSON-Objektes im Upstream-Protokoll [35][22]

Nicht alle der dargestellten Felder sind in jeder Nachricht enthalten, diese variieren je nach Konfiguration der verwendeten Hardware. Allerdings beinhalten sämtliche Pakete mit dem rxpk-Array das Datenfeld, welches für die zu untersuchende Anwendung von größtem Interesse ist. Eine typische JSON-Datenstruktur mit dem angesprochenen Array wird beispielsweise wie folgt übertragen:

```
{‘‘rxpk‘‘: [
        {
            ‘‘tmst‘‘:185110660
            ‘‘freq‘‘:868.100000
            ‘‘modu‘‘:‘‘LORA‘‘
            ‘‘datr‘‘:‘‘SF7BW125‘‘
            …
            ‘‘data‘‘:‘‘AKq7zN3u/xEi+JgE/v/loMxkSNpIk8E=‘‘
        }
        ]
}
```

[22] Kapitel 4: *Upstream JSON data structure*

Das auszugsweise dargestellte JSON-Datenpaket enthält im Feld *data* die Nutzdaten der Sensoren. Diese sollen innerhalb dieses Arbeitspakets aus der Datenstruktur herausgeschnitten werden; man spricht dabei auch von einem Parser. Das empfangene Rohdatenpaket liegt im Proramm in dem Byte-Array *receivedMessageByteArray* vor, innerhalb dessen sich das angesprochene Daten-Feld befindet. Es wird sich dafür entschieden, das Parsen des gewünschten Feldes mittels einer FOR-Schleife zu realisieren, welche dann in verschachtelten IF-Bedingungen das Vorhandensein der ASCII-Zeichen in der richtigen Reihenfolge überprüft. Da beide im Rahmen dieser Thesis verwendeten Gateways exakt die oben im Beispielpaket dargestellte Form des Datenfeldes haben, werden lediglich die in der folgenden Tabelle dargestellten Bitmuster benötigt.

Tabelle 2: ASCII-Codierung der benötigten Zeichen zum Parsen der JSON-Daten [42]

ASCII-Zeichen	Dezimal	Hexadezimal	Binär
“	34	0x22	00100010
:	58	0x3A	00111010
a	97	0x61	01100001
d	100	0x64	01100100
t	116	0x74	01110100

Mit Hilfe dieser Tabelle kann das Vorhaben mittels der FOR-Schleife und verschachtelten IF-Bedingungen das Datenfeld zu lokalisieren, umgesetzt werden. Der entsprechende Codeauszug ist in folgender Abbildung 56 dargestellt.

```
45          FOR iCount:=0 TO TO_INT(SIZEOF(receivedMessageByteArray)) BY 1 DO
46              IF receivedMessageByteArray[iCount]=100 THEN
47                  IF receivedMessageByteArray[iCount+1]=97 THEN
48                      IF receivedMessageByteArray[iCount+2]=116 THEN
49                          IF receivedMessageByteArray[iCount+3]=97 THEN
50                              IF receivedMessageByteArray[iCount+4]=34 THEN
51                                  IF receivedMessageByteArray[iCount+5]=58 THEN
52                                      IF receivedMessageByteArray[iCount+6]=34 THEN
```

Abbildung 56: Abfrage der korrekten Zeichenfolge zum Parsen der JSON-Daten (Codeauszug)

Dabei wird das gesamte Byte-Array Schritt für Schritt durchlaufen und überprüft, ob an der jeweiligen Stelle iCount der Dezimalwert 100, also das kleine *d*, enthalten ist. Falls dies der Fall sein sollte, wird an der nächsten Stelle (iCount+1, Zeile 47) auf die dezimale 97 geprüft, dem kleinen *a*. Mit dieser Methodik wird in sieben IF-Bedingungen die exakte Zeichenfolge *data*":" ermittelt, sofern diese in der jeweiligen Nachricht enthalten ist. Ist das der Fall, befindet man sich an Position iCount+6 im Byte-Array *receivedMessageByteArray*. Somit beginnen ab der Position iCount+7 die eigentlichen Datenbytes des Pakets, welche in ein eigens zu diesem Zweck angelegtem Byte-Array namens *receivedDataByteArray* transferiert werden soll. Dieser Kopiervorgang der Bytes wird so lange durchgeführt, bis erneut die Anführungsstriche, also die dezimale 34, erkannt wird und ist in folgender Abbildung 57 im Codeauszug dargestellt.

```
54      help1:=0;
55      help2:=iCount+7;
56      FOR iCount2:=0 TO TO_INT(SIZEOF(receivedDataByteArray)) BY 1 DO
57      receivedDataByteArray[iCount2]:=0;
58      END_FOR
59      WHILE receivedMessageByteArray[help2] <> 34 DO
60      receivedDataByteArray[help1]:= receivedMessageByteArray[help2];
61      help1:=help1+1;
62      help2:=help2+1;
63      END_WHILE
64      SizeDataByteArray:=help1;
```

Abbildung 57: Herauskopieren der Datenbytes aus der Nachricht (Codeauszug)

In den Codezeilen 54 und 55 werden zunächst zwei Integer-Hilfsvariablen angelegt, wobei die zweite den Wert iCount+7 zugewiesen bekommt. Anschließend wird mittels einer FOR-Schleife das zu beschreibende Daten-Array zurückgesetzt und somit die letzte Nachricht „gelöscht" (Zeilen 56 - 58). Dies ist nötig, da bei einem simplen Überschreiben der Bytes im Falle, dass die aktuellen Daten kürzer sind als die vorigen, weitere Bytes im Array stehen, die nicht Bestandteil des derzeit behandelten Pakets sind. Ab der Zeile 59 wird der eigentliche Kopiervorgang mittels einer WHILE-Schleife realisiert, die so lange mit Hilfe der Hilfsvariablen die jeweiligen Bytes kopiert, bis mit der dezimalen 34 das Ende des Datenfeldes detektiert wird. In Zeile 64 wird die Anzahl der übertragenen Bytes unter Verwendung der ersten Hilfsvariablen gespeichert, da diese Anzahl im späteren Programmablauf noch benötigt wird. Die korrekte Funktion dieses Parsens der Daten aus dem JSON konnte unter Zuhilfenahme

der Variablen-Aktualwerte-Ansicht im eingeloggten Zustand von CODESYS sowie mit Wireshark bestätigt werden. Wie bereits eingangs dieses Abschnitts erwähnt und auch in Abbildung 55 unter „data“ spezifiziert, sind diese Daten Base64 codiert und müssen im nächsten Schritt decodiert werden.

4.3.6 Base64-Decodierung

Die mit Base64 codierten Datenbytes, welche im Array receivedDataByteArray abgelegt sind, sollen jetzt entschlüsselt werden. Base64 ist ein Verfahren zur Codierung von 8-Bit-Binärdaten in eine nur aus ASCII-Zeichen bestehende Zeichenfolge. Hierfür werden jeweils drei 8-Bit-Blöcke des Bytestromes in vier 6-Bit-Blöcke aufgeteilt, wobei jeder dieser Blöcke eine Zahl zwischen 0 bis 63 bildet. Über den Base64-Zeichensatz werden diese in ASCII-Zeichen übersetzt und bilden die Base64-Repräsentanz der drei ursprünglichen Bytes. Das Gleichheitszeichen dient dabei als Füllbyte (besteht nur aus Nullen), falls die Anzahl der Eingangsbytes nicht durch drei teilbar ist. Dieser Vorgang wird als *Padding* bezeichnet, wobei dadurch bis zu zwei Gleichheitszeichen am Ende einer Base64-codierten Nachricht enthalten sein können. Solch eine auf diese Art und Weise codierte Nachricht ist mindestens um den Faktor 4/3 länger als die ursprüngliche Nachricht. Eine Decodierung erfolgt genau umgekehrt zu dem erklärten Algorithmus. Für weitergehende Informationen sei an dieser Stelle auf den Standard RFC 4648 [43] verwiesen, welcher als Quelle für diesen Abschnitt dient und in dem der Algorithmus sowie auch das Base64-Alphabet beschrieben sind.

Zur Implementierung eines solchen Decodier-Bausteins wird nach Recherche die im CODESYS-Store herunterladbare Bibliothek *OSCAT NETWORK* [44] eingesetzt, da in dieser ein Funktionsbaustein zur Decodierung eines beliebig langen BASE64 Byte-Datenstroms vorhanden ist (siehe Abbildung 58).

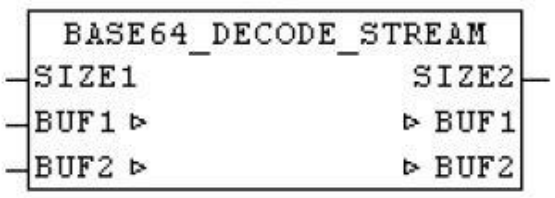


Abbildung 58: FB zur Decodierung von Base64-Daten aus der OSCAT NETWORK-Bibliothek [45]

Der Funktionsbaustein verfügt über einen Eingang SIZE1, welcher als Integer-Variable die Anzahl der zu decodierenden Bytes und einen Ausgang SIZE2, welcher wiederum die Anzahl der Bytes im Ergebnis ausgibt. Als Ein- und Ausgangsparameter dienen BUF1 und BUF2, wobei ersterer ein Byte-Array mit einer Größe von 64 Byte und zweiterer ein Byte-Array mit 48 Byte Speicherplatz ist. Im BUF1 befinden sich die Base64-Daten zum Decodieren, während im BUF2 die konvertierten Daten zu finden sind. Im Verhältnis der beiden Array-Größen ist der bereits angesprochene Faktor 4/3 wiederzufinden.

Nach dem Einbinden der Bibliothek sind Kompatibilitätsprobleme in Bezug auf das im Rahmen dieser Thesis verwendete 64-Bit-System aufgetaucht. Da diese nach Rücksprache mit dem CODESYS- und OSCAT-Supports nicht gelöst werden konnten, wird der Quellcode des Bausteins kopiert und im einen eigenen FB verwendet. Der Aufruf des FBs erfolgt im Hauptprogramm und ist in folgender Abbildung 59 dargestellt.

```
71      BASE64_DECODE_STREAM(BUF1:=receivedDataByteArray,
72                           BUF2:=DataByteArrayBase64Decoded,
73                           SIZE1:=SizeDataByteArray);
74      SizeDataByteDecoded:= BASE64_DECODE_STREAM.SIZE2;
```

Abbildung 59: Aufruf des Base64-Decodier-FBs mit Parameterübergabe (Codeauszug)

Zur Überprüfung der korrekten Funktion des implementierten Bausteins werden online abrufbare Konvertierer herangezogen, wobei an dieser Stelle eine Internetseite [46] hervorgehoben werden soll, da diese auch in späteren Entwicklungsphasen herangezogen wird. Bei Eingabe eines Base64-codierten Pakets erfolgt dabei die in folgender Abbildung 60 dargestellte Ausgabe, welche mit der Variablenansicht in CODESYS unter Verwendung von entsprechenden Haltepunkten verifiziert werden kann.

```
Assuming base64-encoded packet
QAEAAK6A8xEBZQ015o9SE38xIjMVxWRr

Message Type = Data
  PHYPayload = 40010000AE80F31101650D35E68F52137F31223315C5646B
```

Abbildung 60: Ausgabe eines Online-Base64- Konvertierers zur Überprüfung der Software-Implementierung [46]

4.3.7 Handhabung der PHY-Payload

Nach der erfolgten Base64-Decodierung der Daten liegt die sogenannte PHY-Payload der LoRa-Nachricht vor. Der konkrete Aufbau dieser wurde bereits in Kapitel 2.2.5 diskutiert und ist in Abbildung 17 dargestellt.

Innerhalb dieses Arbeitspakets soll die PHY-Payload in die drei Bestandteile, MHDR, MACPayload und MIC, zerlegt werden. Der MHDR ist das erste Byte des dekodierten Byte-Arrays „DataByteArrayBase64Decoded“ und beinhaltet in den Bits 7 bis 5 den MAC-Nachrichtentypen (vgl. Abbildung 18). Mit dieser Information kann an dieser Stelle erkannt werden, ob es sich z. B. um einen Join-Request oder eine Unconfirmed Data Up-Nachricht handelt, indem man die Bitfolge abfragt. Dieser Vorgang wird in der Software über eine CASE-Anweisung realisiert und so zwischen eben diesen Nachrichtentypen unterschieden (siehe Abbildung 61).

```
80      MHDR:=DataByteArrayBase64Decoded[0];
81      CASE MHDR OF
82          16#40:UnconfirmedDataUp:=TRUE;JoinRequest:=FALSE;
83          16#00:UnconfirmedDataUp:=FALSE; JoinRequest:=TRUE;
84          ELSE
85              UnconfirmedDataUp:=FALSE;JoinRequest:=FALSE;
86      END_CASE
```

Abbildung 61: Herauslesen des MHDR und Feststellung des Nachrichtentypen (Codeauszug)

Gemäß des durch den LoRaWAN-Standard spezifizierten Telegrammaufbaus, beinhalten die letzten vier Bytes den MIC, welcher an dieser Stelle aus dem Paket in ein hierfür angelegtes Byte-Array kopiert wird, um eine Validierung zu ermöglichen. Auf einen solchen Mechanismus wird im Rahmen dieser Arbeit allerdings verzichtet. Der Kopiervorgang wird über die durch den FB BASE64_DECODE_STREAM zur Verfügung gestellte Variable SizeDataByteDecoded realisiert (siehe Abbildung 62).

```
87      MICfromPacket[0]:=DataByteArrayBase64Decoded[SizeDataByteDecoded-4];
88      MICfromPacket[1]:=DataByteArrayBase64Decoded[SizeDataByteDecoded-3];
89      MICfromPacket[2]:=DataByteArrayBase64Decoded[SizeDataByteDecoded-2];
90      MICfromPacket[3]:=DataByteArrayBase64Decoded[SizeDataByteDecoded-1];
```

Abbildung 62: Herauslesen des MIC aus der PHY-Payload (Codeauszug)

Der Byte-Kopiervorgang zum Herauslesen der MAC-Payload wird mittels einer FOR-Schleife realisiert, welche erneut auf die Variable SizeDataByteDecoded zurückgreift, die die Anzahl der der Schleifendurchläufe unter Berücksichtigung der bereits behandelten Felder MHDR und MIC steuert (siehe Abbildung 63).

```
94      FOR iCount4:=0 TO SizeDataByteDecoded-6 BY 1 DO
95      MACPayload[iCount4]:=DataByteArrayBase64Decoded[iCount4+1];
96      END_FOR
97      SizeMACPayload:=iCount4;
```

Abbildung 63: Herauslesen der MAC-Payload aus der PHY-Payload (Codeauszug)

Zur Überprüfung der korrekten Funktion dieses Arbeitspakets wird die Variablenansicht in CODESYS bemüht, wobei der bereits im vorigen Abschnitt verwendete Online-Konvertierer die Angaben bestätigt. Das bereits in Abbildung 60 dargestellte Beispielpaket führt zu der in folgender Abbildung 64 dargestellten erweiterten Ausgabe und trägt somit zur Validierung des Codes bei.

```
Assuming base64-encoded packet
QAEAAK6A8xEBZQ015o9SE38xIjMVxWRr

Message Type = Data
  PHYPayload = 40010000AE80F31101650D35E68F52137F31223315C5646B

( PHYPayload = MHDR[1] | MACPayload[..] | MIC[4] )
        MHDR = 40
  MACPayload = 010000AE80F31101650D35E68F52137F312233
         MIC = 15C5646B
```

Abbildung 64: Darstellung der PHY-Payload des Beispielpaket des Online-Konvertierers zu Validierungszwecken

4.3.8 Handhabung der MAC-Payload

Zur weiteren Verarbeitung einer mit Daten befüllten LoRa-Nachricht, muss auch die soeben aus der PHY-Payload herausgetrennte MAC-Payload weiter in seine Bestandteile unterteilt werden. Hierzu wird der bereits in Abbildung 22 dargestellte, grundsätzliche Aufbau einer MAC-Payload herangezogen. Das bereits im Abschnitt 2.2.6 als optional vorgestellte FOpts-Feld ist in der im

Rahmen dieser Arbeit verwendeten Konfiguration der Hardware nicht enthalten. Insofern bilden die ersten sieben Bytes der MAC-Payload den FHDR, welche in folgender Abbildung 65 in den Zeilen 111 bis 113 mittels einer FOR-Schleife herauskopiert werden. Im achten Byte befindet sich das FPort-Feld, sodass ab dem neunten Byte der MAC-Payload die eigentliche FRMPayload beginnt. Auch diese wird unter Zuhilfenahme einer FOR-Schleife mit den entsprechenden Parametern in ein zu diesem Zwecke angelegtes Byte-Array kopiert (Zeilen 118 bis 120).

```
110   IF UnconfirmedDataUp THEN
111       FOR i:=0 TO 6 BY 1 DO
112       FHDR[i]:=MACPayload[i];
113       END_FOR
114       FPort:=MACPayload[7];
115       FOR i:=0 TO TO_INT(SIZEOF(FRMPayload)) BY 1 DO
116       FRMPayload[i]:=0;
117       END_FOR
118       FOR i:=0 TO SizeMACPayload-8 BY 1 DO
119       FRMPayload[i]:=MACPayload[i+8];
120       END_FOR
121       SizeFRMPayload:=i-1;
```

Abbildung 65: Zerlegung der MAC-Payload in die einzelnen Felder (Codeauszug)

Die sieben Bytes des FHDR enthalten gemäß der im Abschnitt 2.2.6 dargestellten Abbildung 23 neben der DevAddr die Felder FCtrl und FCnt. Auch diese werden in den folgenden Codezeilen aus dem Byte-Array in die jeweiligen Variablen übertragen. Gerade die DevAddr besitzt im folgenden Abschnitt, welcher sich mit der Entschlüsselung der FRMPayload auseinandersetzt, eine entscheidende Bedeutung. Denn durch die sensorspezifische DevAddr kann die Auswahl des richtigen Schlüssels zur Decodierung gesteuert werden.

```
124       FOR i:=0 TO TO_INT(SIZEOF(DevAddr)) BY 1 DO
125       DevAddr[i]:=FHDR[i];
126       END_FOR
127       FCtrl:=FHDR[4];
128       FCnt[0]:=FHDR[5];
129       FCnt[1]:=FHDR[6];
```

Abbildung 66: Zerlegung des FHDR in seine Bestandteile (Codeauszug)

Zur Verifizierung der Korrektheit der implementierten Codezeilen wird auch bei diesem Arbeitspaket die Variablenansicht der CODESYS-Entwicklungsumgebung unter Zuhilfenahme geeigneter Haltepunkte verwendet. Darüber hinaus kann die bereits zuvor verwendete Internetseite herangezogen werden, dessen Ausgabe bei Eingabe des Beispielpakets in folgender Abbildung 67 dargestellt ist.

```
( MACPayload = FHDR | FPort | FRMPayload )
        FHDR = 010000AE80F311
       FPort = 01
  FRMPayload = 650D35E68F52137F312233

      ( FHDR = DevAddr[4] | FCtrl[1] | FCnt[2] | FOpts[0..15] )
     DevAddr = AE000001 (Big Endian)
       FCtrl = 80
        FCnt = 11F3 (Big Endian)
       FOpts =

Message Type = Unconfirmed Data Up
   Direction = up
        FCnt = 4595
   FCtrl.ACK = false
   FCtrl.ADR = true
```

Abbildung 67: weiterführende Ausgabe der MAC-Payload mit dem Online-Konvertierer unter Eingabe des Beispielpakets [46]

4.3.9 Entschlüsselung der FRMPayload

Die Entschlüsselung der FRMPayload, in der die eigentlichen Sensordaten enthalten sind, wird in diesem Arbeitspaket behandelt. Als Verschlüsselungsalgorithmus verwendet LoRaWAN, wie bereits in Abschnitt 2.2.4 erwähnt, den AES-128 im leicht modifiziertem *Cipher Block Chaining*-Modus (CBC). Der AES-128 ist ein symmetrisches Verfahren, welches 128-Bit lange Schlüssel verwendet. Beim CBC-Modus handelt es sich um ein Blockchiffre-Verfahren, bei dem vor dem Verschlüsseln eines Klartextblocks dieser zunächst mit einem im vorhergehenden Schritt erzeugten Geheimtextblock per XOR verknüpft wird. Zur Entschlüsselung der mit dem AppSKey Ende-zu-Ende-verschlüsselten Payload der Sensoren, wird die Bibliothek *CmpCrypto* in der Version 3.5.17.0, welche aus dem CODESYS-Store heruntergeladen werden

kann, verwendet. Darüber hinaus wurde die Plattform CODESYS Forge, welche eine offene Website für alle CODESYS bezogenen Open-Source-Projekte mit kostenloser Software und Beispielen ist, herangezogen. Dabei ist vor allem das Beispielprojekt *Crypto Example*, welches unter [48] heruntergeladen werden kann, zu nennen. Dieses Beispielprojekt demonstriert, welche kryptographischen Funktionen in der Bibliothek enthalten sind und wie diese zu verwenden sind, wobei vor allem der implementierte AES-256-Algorithmus als Vergleichsquelle herangezogen wurde. Zur Entschlüsselung muss die in folgender Abbildung 68 dargestellte Funktion *CryptoSymmetricEncrypt* verwendet werden.

FUNCTION CryptoSymmetricEncrypt : RTS_IEC_RESULT

Perform a symmetric encryption using the algorithm handle.

InOut:

Scope	Name	Type	Comment
Return	CryptoSymmetricEncrypt	RTS_IEC_RESULT	Result of the operation
Input	hAlgo	RTS_IEC_HANDLE	Handle to the algorithm.
	pPlainText	POINTER TO RtsByteString	Data to be encrypted
	key	RtsCryptoKey	Key to encrypt the data. Has to be a KeyType_Key key.
	pInitVector	POINTER TO RtsByteString	Init vector of the ecryption
	xEnablePadding	BOOL	Enables padding. If this is not enabled the plaintext length has to match a multiple of the block length.
	pCipherText	POINTER TO RtsByteString	Encrypted data.

Abbildung 68: Dokumentation der zur Entschlüsselung verwendeten Funktion aus der CmpCrypto-Bibliothek

Bei dem Datentyp des Rückgabewertes handelt es sich um eine in dem Interface *SysTypes2Interfaces* enthaltene Datenstruktur. Ähnlich verhält es sich mit der Variablen *hAlgo*, welche für die Auswahl des Algorithmus zuständig ist. Der dabei angegebene Datentyp *RTS_IEC_HANDLE* ist ebenfalls eine in dem soeben vorgestellten Interface enthaltene Managementstruktur. Über die Funktion *CryptoGetAlgorithByID* (siehe Abbildung 69) kann dieser über den Typ *RtsCryptoID* den gewünschten Algorithmus auswählen.

FUNCTION CryptoGetAlgorithmById : RTS_IEC_HANDLE

Get a handle to the algorithm using a specific ID

InOut:

Scope	Name	Type	Comment
Return	CryptoGetAlgorithmById	RTS_IEC_HANDLE	Handle to the crypto algorithm
Input	ui32CryptoID	RtsCryptoID	ID of the algorithm
	pResult	POINTER TO RTS_IEC_RESULT	Result of the operation. Can be NULL.

Abbildung 69: Dokumentation der Funktion zur Festlegung des Algorithmus

Die ID des Algorithmus wird nach Abbildung 69 durch den Datentyp *RtsCryptoID* repräsentiert. Dieser ist ein Enum aus der CmpCrypto Interfaces-Bibliothek und wird gemäß über die in folgender Abbildung 70 dargestellten Zusammenhänge repräsentiert.

TYPE RtsCryptoID :

Attributes:

qualified_only

InOut:

Name	Initial	Comment
AES_128_CBC	16#1001	AES 128 bit key in CBC mode
AES_128_CFB	16#1002	AES 128 bit key in CFB mode
AES_256_CBC	16#1003	AES 256 bit key in CBC mode
DES_CBC	16#1004	DES 64 bit key in CBC mode
AES_256_CTR	16#1005	AES 256 bit key in CTR mode
RSA	16#2001	Plain RSA asymmetric encryption. Not recommended.

Abbildung 70: Dokumentation der unter CODESYS zur Verfügung stehenden Verschlüsselungsalgorithmen (Auszug)

Diese verschachtelte Struktur wird angewendet, um über die in Abbildung 68 dargestellten Eingang hAlgo den gewünschten Algorithmus AES-128_CBC auszuwählen (siehe Abbildung 71).

```
_hAlgo : CmpCrypto_Interfaces.RTS_IEC_HANDLE :=
    CryptoGetAlgorithmById(ui32CryptoID:=RtsCryptoID.AES_128_CBC, pResult:=0);
```

Abbildung 71: Anlegen und Zuweisung der Variablen zur Auswahl des Algorithmus (Codeauszug)

Nachdem die erste Eingangsvariable der zu verwendeten Funktion CryptoSymmetricEncrypt erfogreich angelegt werden konnte, wird mit den anderen Variablen ähnlich verfahren. Beim Typ *RtsByteString* handelt es sich um einen Struct, welcher einer Repräsentanz eines Byte-Strings entspricht. Diese Struktur beinhaltet vier Parameter, wie der aktuellen Länge der Daten im Buffer, dem allokierten Speicher sowie dem Pointer zum Datenpuffer als auch die Möglichkeit, den Speicher dynamisch allokieren zu lassen (siehe Abbildung72).

TYPE RtsByteString : STRUCT

This a representation of a byte string.

InOut:

Name	Type	Comment
ui32Len	UDINT	Current length of data stored in the buffer.
ui32MaxLen	UDINT	The allocated size of the buffer.
pByData	POINTER TO BYTE	Pointer to buffer
xDynamic	BOOL	Indicates if the data was allocated dynamically

Abbildung 72: Dokumentation des STRUCTs RtsByteString

Eine weitere Schwierigkeit bei der Umsetzung der Entschlüsselung liegt in der Bereithaltung des notwendigen Schlüssels. Gemäß der Dokumentation der kryptographischen Funktion ist dieser vom Typ *RtsCryptoKey*, einem Struct zur Verwaltung des Schlüssels (siehe Abbildung 73).

TYPE RtsCryptoKey : STRUCT

A cryptographic key.

InOut:

Name	Type	Comment
keyType	RtsCryptoKeyType	The type of the key.
key	RtsCryptoKeyStorage	Information of the key. Actually only the KeyType_Key is supported.

Abbildung 73: Aufbau des STRUCTs RtsCryptoKey

Innerhalb dieser Struktur werden zwei neue Datentypen sichtbar, zum einem *RtsCryptoKeyType* und zum anderen *RtsCryptoKeyStorage*. Erstgenannter ist ein Enum der CmpCrypto Interfaces, welche drei Möglichkeiten zur Identifizierung der Schlüssel bereithält (z. B. direkte Verfügbarkeit als RTS_BYTESTRING für symmetrsiche Verschlüsselung wie in diesem Fall). Zweitgenannte Datenstruktur ist ein Union, welcher u. a. die Speicherung des Schlüssels verwaltet. Somit wird die Struktur zur Schlüsselverwaltung wie folgt implementiert, wobei der *ckKey* an die Verschlüsselungsfunktion übergeben wird.

```
bsKey: RtsByteString:=(ui32MaxLen:=SIZEOF(AppSKey_Laird),
                       ui32Len:=TO_UDINT(_szKey),
                       pByData:=ADR(AppSKey_Laird));
ksStorage: RtsCryptoKeyStorage:=(byteString:=bsKey);
ckKey : RtsCryptoKey:=(keyType:=RtsCryptoKeyType.KeyType_Key,
                       key:=ksStorage);
```

Abbildung 74: Struktur zur Verwaltung des Schlüssels

In Abbildung 74 ist zu erkennen, dass der Applikationssitzungsschlüssel des Laird-Sensors verwendet wird. Es ist in jedem Fall notwendig, für alle verwendeten Sensoren den jeweiligen AppSKey vorzuhalten und die Struktur zur Schlüsselverwaltung an das gerade sendende Gerät anzupassen. Daher wird sich entschieden, die Entschlüsselung der LoRa-Pakete in einen eigenen Funktionsbaustein auszugliedern, welche die jeweiligen Schlüssel vorhalten kann und in der dann die für die Kryptographie notwendigen Variablen und Funktionen abgearbeitet werden. Diesem FB können vom Hauptprogramm neben dem zu decodierenden Byte-Array und der Größe der Daten auch der FCnt und die DevAddr übergeben werden, sodass über eine entsprechende CASE-Anweisung die Schlüsselzuweisung erfolgt (siehe Abbildung 75).

```
IF DeviceKnown THEN
CASE DevAddr[0] OF
    16#01:bsKey.ui32MaxLen:=SIZEOF(AppSKey_Laird);
          bsKey.ui32Len:=TO_UDINT(_szKey);
          bsKey.pByData:=ADR(AppSKey_Laird);
          ksStorage.byteString:=bsKey;
          ckKey.keyType:=RtsCryptoKeyType.KeyType_Key;
          ckKey.key:=ksStorage;
    16#06:bsKey.ui32MaxLen:=SIZEOF(AppSKey_RAK);
          bsKey.ui32Len:=TO_UDINT(_szKey);
          bsKey.pByData:=ADR(AppSKey_RAK);
          ksStorage.byteString:=bsKey;
          ckKey.keyType:=RtsCryptoKeyType.KeyType_Key;
          ckKey.key:=ksStorage;
    16#03:bsKey.ui32MaxLen:=SIZEOF(AppSKey_Elsys);
          bsKey.ui32Len:=TO_UDINT(_szKey);
          bsKey.pByData:=ADR(AppSKey_Elsys);
          ksStorage.byteString:=bsKey;
          ckKey.keyType:=RtsCryptoKeyType.KeyType_Key;
          ckKey.key:=ksStorage;
    16#04:bsKey.ui32MaxLen:=SIZEOF(AppSKey_Dragino);
          bsKey.ui32Len:=TO_UDINT(_szKey);
          bsKey.pByData:=ADR(AppSKey_Dragino);
          ksStorage.byteString:=bsKey;
          ckKey.keyType:=RtsCryptoKeyType.KeyType_Key;
          ckKey.key:=ksStorage;
END_CASE
```

Abbildung 75: CASE-Anweisung zur Auswahl der sensorspezifischen Schlüssel (Codeauszug)

Zur Entschlüsselung der Sensornutzdaten werden im CBC-Modus zwei 16-Byte-Blöcke über ein XOR miteinander verknüpft, wobei der eine aus den

verschlüsselten Bytes besteht und der andere das Ergebnis des beschriebenen Algorithmus ist. Das im Programm als *aBlock* bezeichnete Byte-Array wird gemäß der folgenden Tabelle mit den entsprechenden Werten aus dem FB-Aufruf verknüpft.

Tabelle 3: Zuweisung des aBlocks mit den dargestellten Werten

aBlock[i]	Wert	aBlock[i]	Wert
aBlock[0]	1	aBlock[8]	DevAddr[2]
aBlock[1]	0	aBlock[9]	DevAddr[3]
aBlock[2]	0	aBlock[10]	FCnt[0]
aBlock[3]	0	aBlock[11]	FCnt[1]
aBlock[4]	0	aBlock[12]	0
aBlock[5]	0	aBlock[13]	0
aBlock[6]	DevAddr[0]	aBlock[14]	0
aBlock[7]	DevAddr[1]	aBlock[15]	counterBlock

Der Wert der Variable *counterBlock* ist im ersten Durchgang der Entschlüsselung gleich eins, und inkrementiert sich in jedem weiteren Durchlauf. Mehrere Durchläufe sind dann nötig, wenn die zu entschlüsselnde Payload länger als die 16 möglichen Bytes ist. Bei den im Rahmen dieses Projektes eingesetzten Sensoren ist dies lediglich bei dem Modell der Firma RAK der Fall.

Dieser so aufgebaute aBlock dient der Funktion CryptoSymmetricEncrypt als Datenbasis, auf den der Algorithmus angewendet werden soll, wobei das Ergebnis der Funktion in das Byte-Array *sBlock* geschrieben wird, welche wie in folgender Abbildung 76 dargestellt definiert sind. Hierbei muss gemäß der Spezifikation der Funktion (vgl. Abbildung 68) erneut die Datenstruktur RtsByteString (vgl. Abbildung 72) verwendet werden.

```
PlainText:RtsByteString:=(ui32MaxLen:=16,ui32Len:=16,pByData:=ADR(sBlock));
CipherText:RtsByteString:=(ui32MaxLen:=16,ui32Len:=16,pByData:=ADR(aBlock));
```

Abbildung 76: Verknüpfung der beiden Blöcke mit der erforderlichen Datenstruktur der Funktion (Codeauszug)

Somit sind alle für den Aufruf benötigten Parameter erstellt und es wird die Funktion unter Verwendung von hAlgo, CipherText, ckKey und dem PlainText

aufgerufen. Darüber hinaus wird das Padding ausgeschaltet und ein nur mit Nullen initialisierter InitVector verknüpft (siehe Abbildung 77).

```
Result := CryptoSymmetricEncrypt(
    hAlgo:=_hAlgo,
    pPlainText:=ADR(CipherText),
    key:=ckKey,
    pInitVector:=ADR(InitVector),
    xEnablePadding:=FALSE,
    pCipherText:=ADR(PlainText)
);
```

Abbildung 77: Aufruf der Kryptographie-Funktion (Codeauszug)

Der durch diese Funktion erhaltene sBlock wird mit den verschlüsselten Sensordaten, welche im Programm durch das Byte-Array *DataEncrypted* abgebildet werden, mittels XOR-Verknüpfung verbunden. Das bei dieser Operation erhaltene Ergebnis-Byte-Array (*DecryptedSensorPayload*) enthält die entschlüsselten Daten der Endknoten. Diese Funktionalität ist über eine FOR-Schleife realisiert und in folgender Abbildung 78 dargestellt.

```
FOR i:=0 TO SizeDataToBeEncrypted-1 BY 1 DO
DecryptedSensorPayload[bufferIndex +i ]:=DataEncrypted[bufferIndex + i] XOR sBlock[i];
END_FOR
```

Abbildung 78: Verknüpfung der beiden Blöcke über XOR zum Erhalt der entschlüsselten Daten

Die im Array-Laufindex enthaltene Variable *bufferIndex* dient dabei der Entschlüsselung der Daten des RAK-Sensors, da dieser wie bereits erwähnt zwei Durchgänge des in diesem Unterkapitel beschriebenen Algorithmus benötigt, da dessen Nutzdaten 19 Byte umfassen. Im Falle, dass dieser entschlüsselt werden soll, wird der erste Schritt in einer anderen Schleife behandelt. Dabei wird die Variable SizeDataToBeEncrypted auf einen Wert größer als 16 geprüft und in dessen Zuge die Variable bufferIndex um den Wert 16 erhöht, sodass alle Daten decodiert werden können.

Die korrekte Funktionalität des hier umgesetzten Arbeitspakets wird erneut mit dem online abrufbaren LoRa-Konvertierer und der Variablenansicht in CODESYS überprüft und kann erfolgreich abgeschlossen werden.

4.3.10 Interpretation der Sensordaten

Nachdem die Daten der Endknoten erfolgreich entschlüsselt werden können, müssen diese nunmehr sensorspezifisch interpretiert werden. Hierzu wird dem Programm erneut ein Funktionsbaustein hinzugefügt, welcher mit der entschlüsseltem Payload aus dem vorigen Abschnitt und der DevAddr als Eingänge versorgt wird, wobei die Temperatur und die Luftfeuchtigkeit als Ausgänge im REAL-Format zur Verfügung gestellt werden. Innerhalb dieses FBs wird erneut mittels eines CASE-Statements und der Überprüfung der mitgegebenen DevAddr die sensorspezifische Interpretation der Daten ermöglicht. Die Vorgehensweise und Umsetzung der Übersetzung der vier verschiedenen Nutzdaten-Paketen wird im Folgenden nacheinander vorgestellt.

Laird-Sensor

Zur Auswertung der Laird-Nutzdaten wird das *RS1xx LoRa Protocol – Application Note* Dokument, welches unter [49] abrufbar ist, verwendet. Die Länge der Daten beträgt demnach bei allen Sensor-zu-LNS-Nachrichten genau 11 Bytes. Innerhalb dieser Bytes gibt es Nachrichtentyp-, Options-, Temperatur-, Luftfeuchtigkeits-, Batteriekapazitäts-, Alarmnachrichten-Zähl- und Backlognachrichten-Zähl-Felder. Im Rahmen dieser Arbeit werden lediglich die Temperatur- und Luftfeuchtigkeitswerte betrachtet.

Die Temperatur wird hierbei mittels zwei Bytes dargestellt, wobei sich an Indexposition 5 die Ganzzahlstelle und an Indexposition 4 die Nachkommastelle befindet. Ähnlich verhält es sich bei der Luftfeuchtigkeit, dabei ist an Position 3 die Vorkomma- und an Position 2 des Arrays die Nachkommastelle zu finden. Der Codeauszug zur Berechnung der beiden Werte beim Endknoten der Firma Laird ist in folgender Abbildung 79 dargestellt.

```
16#01:  Temperature:=TO_REAL(SensorPayloadDecrypted[5])
                      +TO_REAL(SensorPayloadDecrypted[4])/100;
        Humidity:=TO_REAL(SensorPayloadDecrypted[3])
                      +TO_REAL(SensorPayloadDecrypted[2])/100;
```

Abbildung 79: Berechnung der Temperatur- und Luftfeuchtigkeitswerte aus den Nutzdaten des Laird-Sensors (Codeauszug)

Bei einer beispielhaften Nachricht der Bytefolge 01 01 2D 43 5F 15 05 00 00 00 00 ergeben sich somit eine Temperatur von 21,95 Grad Celsius und eine Luftfeuchtigkeit von 67,45 Prozent.

RAK-Sensor

Die Nutzdaten des Produkts der Firma RAK werden über die bereits herangezogene Schnell-Start-Anleitung [31] interpretiert. Die Länge der Daten beträgt hierbei stets 19 Bytes, wobei auch bei diesem Sensor neben der Temperatur und Luftfeuchtigkeit zusätzliche Informationen wie Batteriespannung, Luftdruck und Gasdruck enthalten sind. Ein Datenfeld besitzt gemäß der Spezifikation immer ein 2-Byte langes Datenflag, welches die eigentlichen Werte einleitet. Das verwendete Flag für die Luftfeuchtigkeit ist die Bytefolge 07 68, während das der Temperatur die Bytes 02 67 enthält. Der Luftfeuchtigkeitswert befindet sich somit permanent an der Indexposition 6 des Byte-Arrays, wobei der dortige Wert nach mit dem Faktor 0,5 Prozent multipliziert werden muss. Die Temperaturdaten sind an der Position 13 und 14 des Arrays zu finden, wobei die entsprechen Stellen im Programm mit einer Wertigkeit versehen sind und am Ende die Summe der beiden noch mit dem Faktor 0,1 Grad Celsius multipliziert werden muss. Die Umsetzung dieses Mechanismus im Code sind in folgender Abbildung 80 dargestellt.

```
16#06:  Temperature:=( (TO_REAL(SensorPayloadDecrypted[13]))*256
                              + TO_REAL(SensorPayloadDecrypted[14]))/10;
        Humidity:=TO_REAL(SensorPayloadDecrypted[6])/2;
```

Abbildung 80: Berechnung der Temperatur- und Luftfeuchtigkeitswerte aus den Nutzdaten des RAK-Sensors (Codeauszug)

Bei einer beispielhaften Nachricht der Bytefolge 08 02 01 6D 07 68 77 06 73 27 05 02 67 00 E9 04 02 08 B7 ergeben sich somit eine Temperatur von 23,3 Grad Celsius und eine Luftfeuchtigkeit von 59,5 Prozent.

Elsys-Sensor

Gemäß des Dokuments *Senso_payload.pdf* der Firma Elsys, welches unter [50] zur Verfügung steht, können die Nutzdaten interpretiert werden. Die Länge der Daten beträgt bei diesem Sensor stets acht Byte, wobei auch hier neben den Temperatur- und Luftfeuchtigkeitswerten zusätzlich die Batteriespannung mitgesendet wird. Das Erkennen der jeweiligen Datenfelder ist dabei ebenfalls über Datenflags realisiert. Nach dem Byte 01 folgen zwei Bytes Temperaturdaten, hinter dem Byte 02 schließt sich ein Byte Luftfeuchtigkeitswert an und anschließend an die 07 existieren zwei Bytes mit der Batteriespannung. Gemäß

des im Rahmen dieser Arbeit verwendeten Sensors befinden sich die Temperaturwerte an Indexposition 1 und 2, wobei nach erfolgter Gewichtung der Faktor 0,1 Grad Celsius miteinberechnet werden muss und der Luftfeuchtigkeitswert an Position 4 des Byte-Arrays. Die folgende Abbildung 81 zeigt die Umsetzung im Code.

```
16#03:  Temperature:=( (TO_REAL(SensorPayloadDecrypted[1]))*256
                            + TO_REAL(SensorPayloadDecrypted[2]))/10;
        Humidity:=TO_REAL(SensorPayloadDecrypted[4]);
```

Abbildung 81: Berechnung der Temperatur- und Luftfeuchtigkeitswerte aus den Nutzdaten des Elsys-Sensors (Codeauszug)

Bei einer beispielhaften Nachricht der Bytefolge 01 00 EE 02 38 07 0E 3D ergeben sich somit eine Temperatur von 23,8 Grad Celsius und eine Luftfeuchtigkeit von 56 Prozent.

Dragino-Sensor

Die Interpretation der Daten des Geräts der Firma Dragino wird anhand des Benutzerhandbuchs [24] durchgeführt. Demnach besitzt jede Uplink-Payload stets acht Byte, wobei auch hier neben den Temperatur- und Luftfeuchtigkeitswerten auch der Batteriestatus und eventuelle Werte eines weiteren, optional extern anzuschließenden Sensors enthalten sind. Während die Temperatur aus den Bytes an Position 2 und 3 des Arrays nach entsprechender Gewichtung und unter Einbeziehung eines Faktors von 0,01 Grad Celsius berechnet werden kann, ist die Luftfeuchtigkeit an Position 4 und 5 enthalten. Dabei ist ebenfalls nach der Gewichtung der Stellen ein Faktor von 0,1 Prozent miteinzuberechnen (siehe Abbildung 82).

```
16#04:  Temperature:=( (TO_REAL(SensorPayloadDecrypted[2]))*256
                            + TO_REAL(SensorPayloadDecrypted[3]))/100;
        Humidity:=( (TO_REAL(SensorPayloadDecrypted[4]))*256
                            + TO_REAL(SensorPayloadDecrypted[5]))/10;
```

Abbildung 82: Berechnung der Temperatur- und Luftfeuchtigkeitswerte aus den Nutzdaten des Dragino-Sensors (Codeauszug)

Bei einer beispielhaften Nachricht der Bytefolge CB 7F 07 FE 02 02 01 7F FF 7F FF ergeben sich somit eine Temperatur von 20,46 Grad Celsius und eine Luftfeuchtigkeit von 51,4 Prozent.

Der FB, der die sensorspezifische Interpretation übernimmt, wird im Hauptprogramm unter Parameterübergabe aufgerufen. Über die angelegten Ausgänge kann innerhalb des Programms über den Punktoperator auf die Instanz zugegriffen werden und somit die Werte der Temperatur und Luftfeuchtigkeit als REAL-Variable abgerufen werden (siehe auch Abbildung 83). Somit ist ein Application Layer umgesetzt und erfolgreich implementiert, welches über etwaige Statusbytes aber vor allem durch die plausiblen Wertepaare bestätigt wird.

```
TranslatePayload(SensorPayloadDecrypted:=SensorPayloadDecrypted,DevAddr:=DevAddr);
Temperature:=TranslatePayload.Temperature;
Humidity:=TranslatePayload.Humidity;
```

Abbildung 83: Aufruf des FBs zur sensorspezifischen Interpretation der Daten (Codeauszug)

Nachdem mit der Implementierung der Applikationsschicht alle in Abschnitt 4.2 definierten Arbeitspakete innerhalb dieses Kapitels erfolgreich umgesetzt werden konnten, ist an dieser Stelle die Programmierung eines minimalen LoRaWAN-Netzwerkservers auf der SPS abgeschlossen. Eine Diskussion und Bewertung der Ergebnisse erfolgt nach Betrachtung und Durchführung eines Reichweitentests (Kapitel 5) innerhalb des Kapitel 6.

5. Reichweiten- bzw. Abdeckungstest

Die LoRa-Technologie gibt, wie bereits erwähnt, eine Reichweite von ca. zwei Kilometern in urbaner Umgebung an. Hierbei wirken sich Einflussparameter wie z. B. die Reflexion, Brechung und Dämpfung von Gebäuden negativ auf die Funkreichweite aus. In diesem Kapitel soll ein dem Anwendungsfall entsprechender gerätebasierter Verbindungstest durchgeführt werden, welcher zur Untersuchung der Netzabdeckung für spezielle Anwendungen dient. Hierzu wird an den potentiellen Standorten getestet, wie gut das Linkbudget vom Sensor zum Gateway ist [8][23].

Dafür werden die beiden Gateways an einem festen Standort platziert, während die Sensoren mobil an mehreren Messpunkten für ein festgelegtes Zeitintervall Daten liefern, welche dann anschließend statistisch ausgewertet werden können. Zur Auswertung der Empfangsqualität bzw. der Empfangsstärke wird der mit jeder Datennachricht im JSON enthaltene *Received Signal Strength Indicator* (RSSI) herangezogen. Dabei gilt das Prinzip, je höher der RSSI ist, desto besser ist der Empfang und umgekehrt.

5.1 Planung des Tests

Die Planung des Abdeckungstests wird in zwei Bestandteile unterteilt, wobei sich im ersten Abschnitt der rein physikalischen Durchführung gewidmet wird, bevor sich der mathematischen Planung des Experimentes angenommen wird.

5.1.1 Räumliche Planung

Zur Abbildung des konkreten Anwendungsfalls soll die Abdeckung innerhalb eines Gebäudes simuliert werden. Da das LoRaWAN-Gateway an einer SPS der Gebäudeautomatisierung betrieben werden soll, welche sich üblicherweise in den Schaltschränken der Gebäudetechnik innerhalb der Kellerräume befindet, soll dieses Szenario abgebildet werden. Um den Einfluss der Gebäudebauart zu berücksichtigen, wird der Test in zwei verschiedenen Gebäuden der HAW Hamburg durchgeführt. Zum einem wird das Hochhaus BT-7 mit seiner

[23] Kapitel 3.3: *LoRaWAN und Reichweite;* Seite 28

Stahlbetonbauweise und den aluminiumbedämpften Fenstern gewählt. Zum anderen fällt die Wahl auf den massiven Backsteinbau BT-21. Während es sich beim BT-7 eher um ein hohes und nicht sehr breites Gebäude handelt, ist das BT-21 im Vergleich dazu als niedrig, aber deutlich breiter gebaut anzusehen. Daher werden zur Festlegung der Messpunkte im 14-stöckigen Hochhaus BT-7 die Etagen 2, 5, 8 und 14 gewählt, wobei im 5-stöckigen BT-21 die Etagen 2 und 5 gewählt werden. Zur Überprüfung des horizontalen Einflusses werden im letzten Messpunkt, also dem 5. Stock des BT-21, die zwei extremen Punkte zur Messung herangezogen, welche maximal voneinander entfernt sind. Somit sind insgesamt sieben Stationen in den Abdeckungstest integriert und können nach dem Erfassen der Daten ausgewertet werden.

5.1.2 Mathematische Planung

Nachdem die Anzahl der Messstationen definiert ist, muss die Anzahl der Datenpunkte festgelegt werden. Dabei ist aufgrund der Parametrier-möglichkeiten der verwendeten Sensoren ein minimales Sendeintervall von zwei Minuten bei allen Endknoten einstellbar.

Zur Auswertung der Daten soll neben der Berechnung des Mittelwertes und der Standardabweichung das Konfidenzintervall berechnet werden. Dieses lässt sich nach [47] unter der Annahme einer normalverteilten Größe durch

$$I_{Konf} = \left[\bar{x} - c \cdot \frac{\sigma}{\sqrt{n}} \; ; \bar{x} + c \cdot \frac{\sigma}{\sqrt{n}} \right]$$

darstellen, wobei die verwendeten Variablen wie folgt definiert sind:

- $\bar{x}$: Mittelwert
- c: Konstante (abhängig vom Konfidenzniveau)
- σ: Standardabweichung
- n: Stichprobenumfang

Zur Planung des Stichprobenumfangs wird angenommen, dass es sich beim Mittelwert und der Standardabweichung um nahezu konstante Größen handelt. Zusammen mit der Konstanten c wird bei Betrachtung der Formel zur Berechnung des Konfidenzintervalls deutlich, dass nur die Größe des Stichprobenumfangs n Einfluss auf dieses hat. Ausgehend davon, wird in

folgender Abbildung 84 der Verlauf der Funktion $f(x) = \sqrt{n}$ sowie deren Ableitung $f'(x) = -\frac{1}{2 \cdot \sqrt{n^3}}$ betrachtet, um eine fundierte Aussage zum Einfluss verschiedener Stichprobenumfänge treffen zu können.

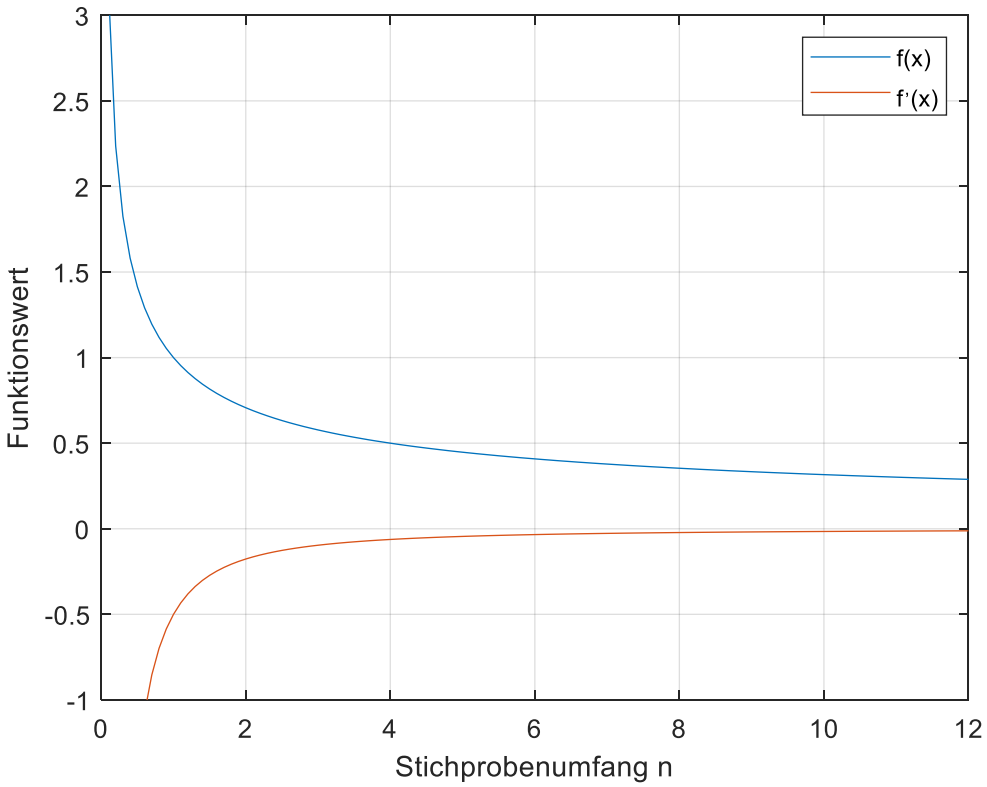


Abbildung 84: geplottete Funktionen zur Festlegung des Stichprobenumfangs

Betrachtet man den Verlauf der Funktion f(x), so erkennt man deutlich, dass bereits ab einem Stichprobenumfang von ungefähr 4 der Einfluss einer höheren Anzahl von Messwerten immer geringer wird Dies ist auch im Verlauf der Ableitung der Funktion zu erkennen, die dabei abgebildete Änderungsrate des Funktionswertes geht offenbar sehr schnell gegen Null. Zur besseren Einschätzung soll der Verlauf der Änderungsrate nochmals in einem vergrößerten Bereich betrachtet werden (siehe Abbildung 85). Anhand dieser soll dann eine Ableitung zur Bestimmung der aufzunehmenden Datenpunkte erfolgen.

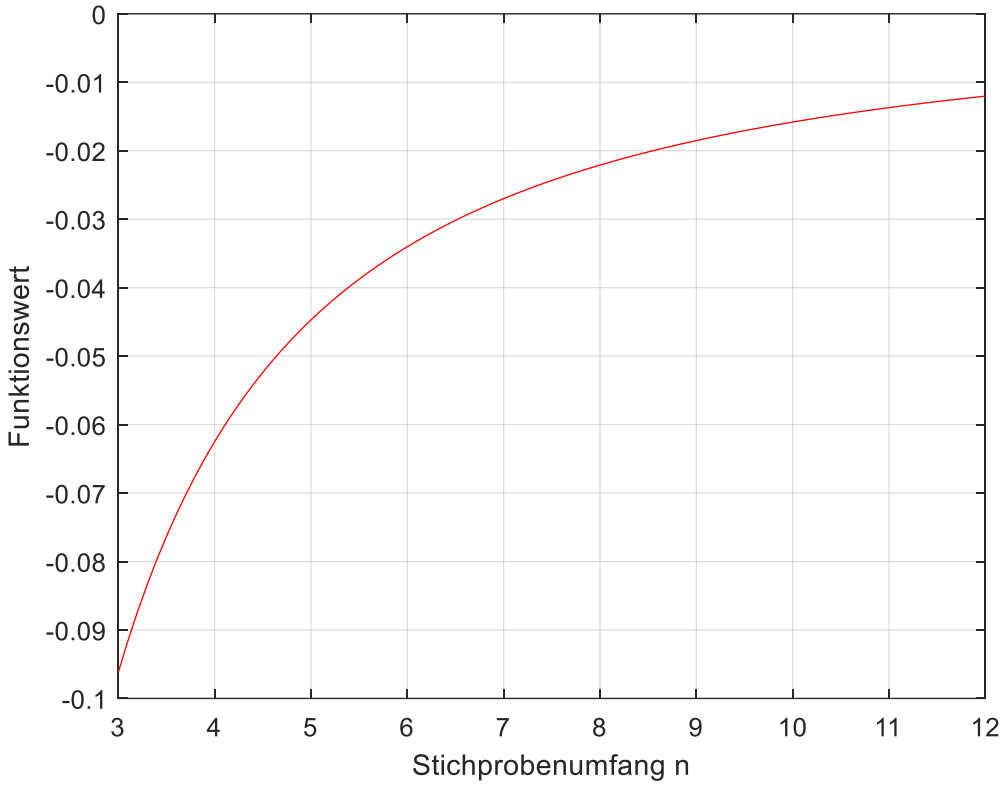


Abbildung 85: vergrößerter Ausschnitt der Änderungsrate der Funktion zum Abschätzen des benötigten Stichprobenumfangs

Bei Betrachtung des hier vergrößert dargestellten Ausschnittes der Änderungsrate kann festgestellt werden, dass bei dem zuvor diskutiertem Stichprobenumfang von 4 die Änderungsrate noch über 6 Prozent beträgt, sodass eine größere Datenmenge angestrebt wird. In einer Abwägung zwischen zeitlichem Aufwand (jeder Datenpunkt benötigt zwei Minuten Wartezeit) und einer vertretbaren Änderungsrate wird sich für einen Stichprobenumfang von 9 entschieden. Bei diesem Wert sinkt die Änderungsrate erstmals unter 2 Prozent und es ist bei jedem Standort mit einer Messdauer von ca. 20 Minuten zu rechnen. Somit ergibt sich eine theoretische Dauer des Tests bei 7 geplanten Messstationen von ungefähr 140 Minuten, wobei die Standortwechsel noch nicht einberechnet sind. Durch diese Entscheidung wird ein guter Kompromiss zwischen Aufwand und Nutzen für eine erste Aussage in Bezug auf den anwendungsbezogenen Abdeckungstest erwartet.

5.2. Durchführung des Tests

Für die praktische Durchführung des Abdeckungstests ist zum einen ein Laptop mit der entwickelten Desktop-Applikation notwendig. Diese beinhaltet in den Feld Application Layer (vgl. Abbildung 35) neben den eigentlichen Daten einen Zeitstempel, die MAC-Adresse des Gateways, die DevAddr des Sensors und den RSSI, welche von der Applikation in einem Log gespeichert werden und somit zur Auswertung der Signalstärke herangezogen werden können. Zum anderen wird zur Umsetzung des Testkonzepts eine zweite Person benötigt. Eine Person befindet sich in dem jeweiligen Gebäude, indem der Test stattfindet, in den Kellerräumen an einem möglichst zentral gelegenen Punkt, betreut den Testaufbau, welcher die Gateways über einem Switch mit dem Laptop verbindet und behält die Ausgabe der Applikation im Blick. Die weitere Person bewegt sich mit den vier Sensoren zu den festgelegten Stationen und notiert sich die Anfangs- und Endzeiten der Messdatenaufnahme. Zur besseren Kommunikation und Abstimmung wird eine Mobilfunkverbindung aufrechterhalten. Es ist bei der Durchführung darauf zu achten, dass alle Sensoren 9-mal die Möglichkeit haben, ihre Daten über die Gateways an den LNS zu senden. Diese Vorgehensweise wird über alle Stationen in beiden Gebäuden möglichst gleichbleibend angewendet, um eine maximale Vergleichbarkeit der Daten zu erhalten.

5.3 Auswertung des Tests

Zur Auswertung der Messpunkte werden aus den aufgenommenen RSSI-Werten die Mittelwerte, Standardabweichungen und das Konfidenzintervall nach [46] berechnet. Die Analyse beinhaltet außerdem die absolute Anzahl der empfangenen Pakete und ist in Abhängigkeit der beiden Testgebäude getrennt voneinander dargestellt.

In der folgenden Tabelle 4 ist die zusammenfassende Auswertung der Datenpunkte aus dem ersten Objekt BT-7 abgebildet. Zur Berechnung des Konfidenzintervalls wird die in 5.1.2 vorgestellte Formel bei einem Konfidenzniveau von 95 %, welches einer Konstanten c von 1,96 entspricht, verwendet.

Tabelle 4: Zusammenfassende Auswertung der Messdaten aus dem BT-7

Raum	Gateway	Mess-werte	Sensoren			
			Laird	Elsys	Dragino	RAK
BT-7 2. OG	Kerlink	Anzahl	9	9	9	3
		$\bar{x}$	-103,0	-98,6	-109,2	-107,7
		σ	3,4	5,6	2,0	1,5
		I_{Konf}	[-105,2;-100,8]	[-102,2; -94,9]	[-110,6;-107,9]	[-109,4;-105,9]
	Laird	Anzahl	9	9	9	3
		$\bar{x}$	-104,4	-96,8	-107,1	-106,7
		σ	3,4	5,0	0,6	0,6
		I_{Konf}	[-106,6;-102,3]	[-100,0;-93,5]	[-107,5;-106,7]	[-107,3;-106,0]
BT-7 5. OG	Kerlink	Anzahl	9	9	8	0
		$\bar{x}$	-108,1	-107,1	-110,4	-
		σ	3,6	2,2	2,1	-
		I_{Konf}	[-110,5;-105,8]	[-108,6;-105,7]	[-111,9;-108,9]	-
	Laird	Anzahl	9	9	7	0
		$\bar{x}$	-106,2	-106,0	-107,3	-
		σ	2,0	1,7	1,3	-
		I_{Konf}	[-107,6;-104,9]	[-107,1;-104,9]	[-108,2;-106,4]	-
BT-7 8. OG	Kerlink	Anzahl	9	7	8	0
		$\bar{x}$	-109,2	-109,0	-109,9	-
		σ	2,6	0,8	2,1	-
		I_{Konf}	[-110,9;-107,5]	[-109,6;-108,4]	[-111,3;-108,4]	-
	Laird	Anzahl	8	9	2	1
		$\bar{x}$	-105,6	-106,7	-108,5	-107,0
		σ	2,3	1,4	0,7	-
		I_{Konf}	[-107,2;-104,1]	[-107,6;-105,7]	[-109,5;-107,5]	-
BT-7 14. OG	Kerlink	Anzahl	6	6	0	0
		$\bar{x}$	-108,5	-114,5	-	-
		σ	2,1	0,7	-	-
		I_{Konf}	[-110,2;-106,8]	[-115,1;-113,9]	-	-
	Laird	Anzahl	6	6	0	0
		$\bar{x}$	-103,5	-107,5	-	-
		σ	6,4	2,1	-	-
		I_{Konf}	[-108,6;-98,4]	[-109,2;-105,8]	-	-

In der folgenden Tabelle 5 sind nunmehr die Ergebnisse der Auswertung für das zweite Testgebäude BT-21 zusammengefasst.

Tabelle 5: Zusammenfassende Auswertung der Messdaten aus dem BT-21

Raum	Gateway	Mess-werte	Sensoren			
			Laird	Elsys	Dragino	RAK
BT-21 3. OG	Kerlink	Anzahl	8	8	9	8
		$\bar{x}$	-95,9	-80,8	-101,8	-97,6
		σ	13,3	4,9	5,3	4,1
		I_{Konf}	[-105,1;-86,6]	[-84,2;-77,3]	[105,2;-98,3]	[-100,5;-94,8]
	Laird	Anzahl	8	8	9	9
		$\bar{x}$	-86,8	-79,9	-101,1	-92,9
		σ	8,0	2,9	6,0	4,0
		I_{Konf}	[-92,3;-81,2]	[-81,9;-77,8]	[-105,1;-97,2]	[-95,5;-90,3]
BT-21 5. OG, vor R511	Kerlink	Anzahl	8	4	0	0
		$\bar{x}$	-108,6	-107,3	-	-
		σ	0,5	4,2	-	-
		I_{Konf}	[-109,0;-108,3]	[-111,4;-103,1]	-	-
	Laird	Anzahl	4	5	0	0
		$\bar{x}$	-109,8	-108,8	-	-
		σ	1,5	2,9	-	-
		I_{Konf}	[-111,2;-108,3]	[-111,3;-106,3]	-	-
BT-21 5. OG, vor R526	Kerlink	Anzahl	8	9	9	9
		$\bar{x}$	-94,2	-101,5	-108,3	-105,7
		σ	6,3	10,2	2,3	2,8
		I_{Konf}	[-98,3;-90,1]	[-108,2;-94,8]	[-109,8;-106,9]	[-107,5;-103,8]
	Laird	Anzahl	8	9	7	6
		$\bar{x}$	-98,6	-101,8	-109,0	-103,8
		σ	6,8	5,7	1,6	2,1
		I_{Konf}	[-103,1;-94,1]	[-105,5;-98,1]	[-110,2;-107,8]	[-105,2;-102,4]

Bei Betrachtung der Auswertungsergebnisse aus dem BT-7 ist auf den ersten Blick erkennbar, dass der Sensor der Firma RAK am schlechtesten abschneidet. Bereits im 2. Obergeschoss empfangen beide Gateways lediglich 3 von 9 möglichen LoRa-Paketen, wohingegen die anderen drei Sensoren jeweils alle Datagramme erfolgreich versenden. Ab dem 5. Stockwerk des Gebäudes erreichen nur noch 1 von 54 möglichen Paketen des RAK-Produkts den LNS auf dem sich im Keller befindlichen Laptop. Ebenfalls sind ab dieser Messstation erste Ausfälle des Dragino-Sensors festzustellen, welche dann ab der 8. Etage immer häufiger werden und im 14. Stock darin münden, dass keines von 18 möglichen Datenpaketen empfangen wird. Analysiert man die Empfänge der

beiden anderen Sensoren, so ist festzustellen, dass beide auch noch in der obersten Etage des Hochhauses BT-7 in 12 von 18 möglichen Fällen ihre Daten an die Gateways im Keller senden. Bezieht man den Mittelwert des RSSI in die Auswertung mit ein, so lässt sich erkennen, dass diese beim Laird-Sensor etwas besser ausfallen als beim Sensor der Firma Elsys. Allerdings ist dabei auch die Standardabweichung des Laird-Sensor fast durchgehend höher als die des Elsys-Geräts. Der einzige signifikante Unterschied beim Vergleich der beiden Gateways im BT-7 ist bei der Anzahl der angekommenen Pakete des Dragino-Sensors im 8. OG festzustellen, wobei das Verhältnis 8 zu 2 für den Kerlink spricht.

Die Analyse der Auswertung aus dem BT-21 ergibt, dass sich in diesem Gebäude aus der ersten Messstation heraus bis auf sehr wenige Ausnahmen (67 von 72 möglichen Paketen) fast alle Daten empfangen lassen. Dabei gibt es allerdings signifikante Unterschiede bei den RSSI-Werten, welche beim Gerät der Firma Elsys um die -80, beim Laird-Sensor ca. -90, beim RAK-Endknoten ungefähr -95 und beim Produkt von Dragino bereits -101 betragen. Diese Tendenz bestätigt sich bei der Betrachtung des nächsten Messpunktes, denn an diesem kann kein einziges der möglichen 36 Pakete der beiden zuvor am schlechtesten abschneidenden Sensoren erfolgreich empfangen werden. Im Vergleich dazu werden beim Laird-Sensor 12 von 18 und beim Elsys-Gerät 9 von 18 möglichen Datagrammen, welche allesamt mit betragsmäßig hohen RSSI-Werten versehen sind, empfangen. Interessanterweise ist bei einer horizontalen Verlegung des Messortes innerhalb desselben Stockwerkes ein völlig anderes Bild zu sehen, denn an diesem Standort sind 65 der 72 möglichen Pakete zu empfangen, wobei sich auch hierbei die bereits zuvor bemerkten Unterschiede in der Signalstärke feststellen lassen.

Beim Vergleich der Ergebnisse der beiden Gebäude ist augenscheinlich, dass die Standardabweichungen und somit auch die Konfidenzintervalle im BT-21 deutlich höher bzw. größer ausfallen als im BT-7.

Zieht man die Daten beider Gateways vergleichend heran, so sind bis auf wenige Ausnahmen (z. B. BT-7 8. OG) keine allzu großen Leistungsunterschiede erkennbar. Einzig die Tatsache, dass der RSSI-Wert unabhängig von Messstation und Sensor beim Laird-Gateway fast immer geringfügig besser, also betragsmäßig niedriger, als beim Produkt der Firma Kerlink ausfällt, ist erwähnenswert.

6. Fazit und Bewertung

Im Kapitel 3 wurde der konkrete Anwendungsfall detailliert beleuchtet und in diesem Zuge die Aufgabenstellung herauskristallisiert. Das Ziel dieser Arbeit war es, herauszufinden, ob und mit welchen Einschränkungen man einen LoRaWAN-Netzwerkserver auf einer SPS betreiben kann. Dabei wurde auf eine Minimalimplementierung zurückgegriffen, welche ohne Internetverbindung und Software von Drittanbietern auskommt. Im Rahmen dieser Arbeit konnte das innerhalb des Kapitel 3 erarbeitete Konzept zur Umsetzung mit der ausgewählten Hardware in einem Testaufbau erfolgreich implementiert werden. Das entwickelte SPS-Programm ist in der Lage, nachdem ein Sensor einem Netzwerk hinzugefügt wurde, die vom Gateway weitergeleiteten Pakete zu verarbeiten, zu entschlüsseln und entsprechend zu interpretieren. Somit ist die prinzipielle Durchführbarkeit des Vorhabens als belegt anzusehen. Obwohl der Proof of Concept damit erfolgt ist, so sind in der Umsetzung des Programms noch einige Schwächen bzw. Verbesserungspotentiale enthalten, welche nicht vorenthalten werden sollen und auf die im Folgenden eingegangen wird.

1. Das Parsen der JSON-Daten kann robuster erfolgen. Entweder man greift auf bereits vorhandene JSON-Parser in verfügbaren Bibliotheken zurück oder man modifiziert den programmierten Parser. In der derzeitigen, finalen Programmversion gelingt das Herauslesen der Daten nur dann, wenn exakt die überprüfte Zeichenfolge vorkommt. Dies ist bei den beiden verwendeten Gateways der Fall, muss aber bei Modellen anderer Firmen nicht zwingend so sein, denn der JSON-Standard erlaubt auch andere Varianten. Darüber hinaus würde ein gekapselter Parser die Übersichtlichkeit des Programmcodes erhöhen, da dadurch die verschachtelten IF-Bedingungen entfallen könnten.
2. Die Base64-Decodierung funktioniert nur bei Datenarrays mit maximal 64 Bytes Länge. Der implementierte FB zur Decodierung nach dem Base64-Algorithmus arbeitet auf einem 64-Byte großen Array aus welchem er 48 Bytes Daten gewinnt. Bei den im Rahmen dieser Arbeit verwendeten Sensoren ist die maximale Größe der Daten geringer als die 64 Bytes, aber auch das muss keinesfalls für alle Modelle gelten. Bei größeren Datenmengen ist ein mehrfacher Aufruf und entsprechendes

Auseinanderschneiden und Wiederzusammenfügen der Arrays notwendig.

3. Die Speicherung der geheimen AppSKeys kann verbessert werden. Aktuell sind die Schlüssel in Byte-Arrays innerhalb des Kryptographie-FBs gespeichert. Um einen ungewollten Zugriff oder Auslesemöglichkeiten zu minimieren sind passwortgeschützte Strukturen oder von außen nicht konfigurierbare Bausteine zur Schlüsselverwaltung denkbar.
4. Ein objektorientierter Ansatz könnte zur besseren Übersichtlichkeit und Funktionalität beitragen. Dabei könnten Interfaces implementiert werden und die Variablenübergaben via Getter und Setter erfolgen. Als Folge dessen wird die Modularisierung und der Aufbau von größeren Netzwerken vereinfacht.
5. Die Handhabung des MAC-Payloads kann robuster erfolgen. Die in Abschnitt 4.3.8 vorgestellte Lösung zum Auslesen der FRMPayload basiert auf der Tatsache, dass das optionale FOpts-Feld in der im Rahmen dieser Arbeit verwendeten Hardwarekonfiguration nicht enthalten ist. Bei Vorhandensein dieses Feldes muss das Herauskopieren der Payload angepasst werden. Darüber hinaus ist dieses Feld lt. LoRaWAN-Standard verschlüsselt, demnach wäre bei Vorhandensein eine Entschlüsselung notwendig.
6. Die Arraylängen sind statisch und nicht dynamisch. Die Arrays, die zur Speicherung der ankommenden Nachrichten genutzt werden, besitzen eine statisch vorgegebene Länge, welche für die untersuchte Hardware ausreichend ist. Dies kann sich bei anderen Produkten aber auch bei abweichenden Konfigurationen ändern, sodass diese entweder angepasst werden müssen oder eben eine dynamische Speicherplatz-Allokierung erfolgen muss.

Unabhängig von den angesprochenen Verbesserungsmöglichkeiten des Programmcodes konnte die Aufgabenstellung erfolgreich abgeschlossen werden. Der Aufbau eines LoRaWAN-Netzwerks im betrachteten Anwendungsfall könnte unter Berücksichtigung der Erkenntnisse aus dieser Thesis wie folgt ablaufen. Ein Sensor wird mittels der entwickelten Desktop-Applikation einem Netzwerk hinzugefügt und die dabei berechneten

Sitzungsschlüssel sind dem Anwender durch die von diesem Programm erzeugte Textdatei zugänglich. Der dabei generierte AppSKey (eventuell auch der NwkSKey für spätere Erweiterungen) wird dem Programm auf der SPS zur Verfügung gestellt, welches mit den entwickelten Funktionalitäten anschließend die benötigten Teile des Netzwerkmanagements sowie die Applikationsebene enthält. Die vom Programm gelieferten Daten können dann den für die Heizkreisregelungen verantwortlichen Programmen verfügbar gemacht werden und somit ist die Datenbasis für eine intelligente Heizkreisreglung mit Hilfe von LoRa-Sensoren und eines selbstentwickelten LNS auf einer SPS ohne Internetverbindung möglich.

Bezieht man die in dieser Thesis verwendete Hardware in die Bewertung mit ein, so sind unter Einbeziehung aller Faktoren wie z. B. Art der Konfiguration, Parametrierbarkeit, Umfang und Qualität der Dokumentation, mechanische und technische Robustheit einige Aussagen abzuleiten.

Betrachtet man zunächst die beiden verwendeten Gateways, so sind vor allem bei der Art der Konfiguration Unterschiede festzustellen. Während das Produkt der Firma Laird ausnahmslos über das Webinterface zu parametrieren ist, besitzt dieses beim Kerlink-Gateway eher darstellenden Charakter und kommt rudimentär daher. Allerdings bietet das Linux-basierte Betriebssystem vielfältige Möglichkeiten, welche die des Laird übersteigen, es sind allerdings weitergehende Kenntnisse dafür nötig. Der Reichweitentest wies in Hinblick auf die beiden Gateways wenig Unterschiede auf, sodass unter Berücksichtigung aller Faktoren eine Umsetzung des zu evaluierenden Anwendungsfalls mit diesen Geräten uneingeschränkt erfolgen kann.

Bei der Analyse der Sensoren unter Berücksichtigung der genannten Faktoren kann eine solche Aussage nicht getroffen werden. Neben einigen (eventuell noch vertretbaren) Schwächen bei den Produkten der Firma Dragino und RAK wie z. B. komplizierte Parametrierbarkeit und teilweise unzureichende Dokumentation, haben diese beiden Sensoren vor allem beim Reichweitentest schlecht abgeschnitten. Dabei ist zu erwähnen, dass der RAK-Endknoten sich als noch deutlich unperformanter gezeigt hat. Daher wird von einer Verwendung dieser Sensoren abgeraten, wobei die Produkte der Firmen Elsys und Laird auch in Bezug auf den Reichweitentest als ähnlich leistungsfähig angesehen werden können, sodass die Verwendung dieser beiden empfohlen werden kann.

Der anwendungsbezogene Abdeckungstest hat darüber hinaus feststellen lassen, dass die angestrebte Nutzung der LoRa-Technologie im Rahmen des Anwendungsfalls möglich ist. Mit der Verwendung eines Gateways in den Kellerräumen und den beiden angesprochenen Sensoren konnte im BT-7 fast uneingeschränkt das gesamte Gebäude bis hin zum 14. Stock abgedeckt werden. Der Verlust einzelner Pakete ist im Rahmen der Verwendung als Referenz für Heizkreisregelungen vertretbar, da es sich dabei ohnehin um träge Systeme mit hohen Reaktionszeiten handelt. Der Unterschied zum BT-21 ist augenscheinlich, denn hierbei ist eine Abdeckung des gesamten Gebäudes zwar ebenfalls möglich, allerdings sind die Unterschiede zwischen den Messstandorten einer Etage auffällig. Die baulichen Unterschiede verschiedener Gebäudeteile haben offensichtlich erheblichen Einfluss auf die Abdeckung und beeinflussen die Umsetzung eines solchen Projekts maßgeblich. Es ist also ratsam, bei der Konzipierung und Planung den Standort des Gateways (sofern möglich) und vor allem der Referenzräume für die Erfassung der Datenpunkte über die Sensoren sorgfältig zu prüfen und gegebenenfalls über einen eigenen, größer ausgelegten Abdeckungstest zu validieren.

Das von LoRaWAN verwendete symmetrische Verschlüsselungsverfahren AES gilt allgemeinhin immer noch als sicher und wird von angesehen Kryptologen verwendet. Einzig die verwendete Schlüssellänge von 128 Bit ist negativ anzumerken, denn der AES mit 256 Bit Schlüssellänge gilt langfristig als sicherer. Der dabei verwendete CBC-Modus ist zwar weit verbreitet, weist aber einige Schwächen auf und führen in der Realität zu sogenannten POODLE-Angriffen. Die Blockgrößen sind fest vorgegeben, sodass diese nicht integritätsgeschützt sind und sich somit zum Durchprobieren bestimmter Entschlüsselungs-algorithmen missbrauchen. Außerdem können die Verschlüsselung großer Datenmengen nicht parallelisiert werden [51]. Zusammenfassend kann man sagen, dass zwar bessere Algorithmen zur Verfügung stehen aber die IT-Sicherheit des LoRaWAN-Standards durchaus als hoch angesehen werden kann.

Nachdem die grundsätzliche Durchführbarkeit nachgewiesen ist, auf die Schwächen im Code hingewiesen wurde, eine Auswertung des Abdeckungstests in die Bewertung der Hardware miteingeflossen ist und die IT-Sicherheit des von LoRaWAN verwendeten Algorithmus betrachtet wurde, soll im Folgenden ein

Ausblick erfolgen. Dabei sollen Erweiterungen zur Verbesserung des Programms und des Konzeptes allgemein vorgeschlagen werden. Die folgenden Aufzählungen bilden den Abschluss dieser Thesis.

1. Hinzufügen der MIC-Berechnung zur Validierung der Pakete: Hierbei ist der NwkSKey und der AES-CMAC zu verwenden. Mit diesem Mechanismus können erhaltene MICs validiert und eigene MICs zum Signieren von Paketen berechnet werden.
2. Erweiterung der behandelten Nachrichten: Neben den in dieser Arbeit behandelten Unconfirmed Data Up-Nachricht könnten auch Confirmed Data Up-Nachrichten oder Downlink-Pakete behandelt werden.
3. Komplettimplementierung des LNS: Dieser Punkt beinhaltet die Komplettimplementierung des Servers auf der SPS, sodass eine Nutzung der Desktop-Applikation entfallen könnte.
4. Herausfiltern redundanter Datenpakete implementieren: Kommerziell verfügbare LNS haben diese Funktion enthalten, um den Traffic so gering wie möglich zu halten.
5. Neben dem Semtech UDP-Protokoll andere Packet Forwarder implementieren: Fast alle Hersteller von Gateways besitzen eigene Packet Forwarder, welche ebenfalls verarbeitet werden könnten.
6. Hinzufügen einer automatischen Regelung der Spreizfaktoren und Datenraten: Auch diese Funktionen sind in kommerziellen LNS enthalten, da durch Anpassen dieser eine Optimierung der Energiemenge und Reichweite erzielt werden kann.
7. Erweiterung des Application Layers: Bei diesem Punkt sind vielfältige Möglichkeiten denkbar, z. B. kann der Batteriestatus, welchen alle Sensoren mitliefern, ausgewertet werden und bei kritischen Werten eine automatisierte Benachrichtigung generieren.

7. Literatur- & Quellenverzeichnis

[1] Lueth, Knud L.: *State of the IoT 2020: 12 billion IoT connections, surpassing non-IoT for the first timeaccelerating* https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/ Version: November 2020. – [Online: Stand 07.02.2022]

[2] powunity: *Sigfox, LoRa – Unabhängige Funknetze für den Datenaustausch? Das steckt dahinter!* https://powunity.com/sigfox-lora-unabhaengige-funknetze-fuer-den-datenaustausch [Online: Stand 07.02.2022]

[3] O´Dea, S.: *Global LPWAN connections 2017-2023, by technology* https://www.statista.com/statistics/880822/lpwan-ic-market-share-by-technology/ Version: Mai 2021. [Online: Stand 07.02.2022]

[4] Jung, Tom Martin: *Vergleich aktueller LPWAN-Technologien im Internet der Dinge unter Einbindung von Energy-Harvesting* https://opus.hs-offenburg.de/frontdoor/deliver/index/docId/2297/file/TomJung_Bachelorthesis_final.pdf [Online: Stand 08.02.2022]

[5] Telekom: *NB-IoT, LoRaWAN, Sigfox: ein aktueller Vergleich* https://iot.telekom.com/resource/blob/data/570954/0ddb2c4808e68427035d415001fc645d/mobile-iot-netzwerk-vergleich-nb-iot-lorawan-sigfox.pdf
[Online: Stand 09.02.2022]

[6] LoRaWAN: *Was ist LoRaWAN?* https://www.lora-wan.de/ [Online: Stand: 09:02.2022]

[7] SEMTECH: *What is LoRa?* https://www.semtech.com/lora/what-is-lora [Online: Stand 09.02.2022]

[8] Linnemann, M.; Sommer, A.; Leufkes,R.: *Einsatzpotentiale von LoRaWAN in der Energiewirtschaft* , SpringerVieweg (2019)

[9] Barz, Felix: *Sichere LPWAN-Übertragungen und IoT-Updates am Beispiel eines Smarten Briefkastens* https://www.h-brs.de/files/related/masterthesis-felix-barz-hbrs.pdf Version: 22.06.2019 [Online: Stand 08.02.2022]

[10] Wenck, Florian; Vorlesungsskript: *Bussysteme – 01 Grundlagen*, https://emil.haw-hamburg.de/course/view.php?id=53446 , [Online: Stand 10.02.2022]

[11] LoRa Alliance: *What is LoRaWAN Specifications* https://lora-alliance.org/about-lorawan/ [Online: Stand 10.02.2022]

[12] LoRa Alliance: *lorawantm_specification_-v1.1.pdf;* Version 1.1 Oktober 2017; abrufbar unter: https://lora-alliance.org/resource_hub/lorawan-specification-v1-1/ [Online: Stand 10.02.2022]

[13] LoRa Alliance: LoRaWAN Regional Parameters *RP002-1.0.3.pdf;* Version 1.0.3 Mai 2021; abrufbar unter: https://lora-alliance.org/resource_hub/rp2-1-0-3-lorawan-regional-parameters/ [Online: Stand 10.02.2022]

[14] LoRa Alliance: *About LoRa Alliance* https://lora-alliance.org/about-lora-alliance/ [Online: Stand 10.02.2022]

[15] LoRa Alliance; https://lora-alliance.org/ [Online: Stand 15.02.2022]

[16] LoRa Alliance: *what-is-lorawan.pdf;* Version: November 2015; abrufbar unter https://lora-alliance.org/resource_hub/what-is-lorawan/ [Online: Stand 15.02.2022]

[17] The Things Network: *Packet Forwarders;* https://www.thethingsnetwork.org/docs/gateways/packet-forwarder/ [Online: Stand 15.02.2022]

[18] Telekom Whitepaper: *Vergleich und Analyse der Sicherheitsaspekte von LoRaWAN und NB-IoT;* abrufbar unter https://iot.telekom.com/de/blog/iot-security-nb-iot-lorawan [Online: Stand 16.02.2022]

[19] Ghosliya,S. Blog: *All About LoRa and LoRaWAN – LoRa Decoding;* http://www.sghoslya.com/p/lora_9.html [Online: Stand 16.02.2022]

[20] The Things Network: https://www.thethingsnetwork.org/ [Online: Stand 22.02.2022]

[21] ChirpStack: https://www.chirpstack.io/ [Online: 24.02.2022]

[22] Laird Connectivity: *Using the Sentrius RS1xx Sensor App*; https://www.lairdconnect.com/using-sentrius-rs1xx-sensor-app [Online: Stand 24.02.2022]

[23] Elsys Shop: https://www.elsys.se/shop/product/ers-lite/?v=f003c44deab6 [Online: Stand 24.02.2022]

[24] Dragino: *LHT65 LoRaWAN Temperature & Humidity Sensor User Manual;* https://www.dragino.com/products/temperature-humidity-sensor/item/151-lht65.html [Online:Stand 24.02.2022]

[25] RAK Shop: https://store.rakwireless.com/products/rak7204-lpwan-environmental-node [Online: 24.02.2022]

[26] Laird Connectivity: *Sentrius RG1xx LoRaWAN Gateway + Wi-Fi / Ethernet + Optional LTE (US Only);* https://www.lairdconnect.com/wireless-modules/lorawan-solutions/sentrius-rg1xx-lorawan-gateway-wi-fi-ethernet-optional-lte-us-only [Online: Stand 24.02.2022]

[27] kerlink: *Wirnet iFemtoCell-evolution;* https://www.kerlink.com/product/wirnet-ifemtocell-evolution/ [Online: Stand 24.02.2022]

[28] Laird Connectivity: *User Guide Sentrius RS1xx v2_2.pdf;* Version 2.2; abrufbar unter: https://www.lairdconnect.com/wireless-modules/lorawan-solutions/sentrius-rs1xx-lora-enabled-sensors [Online: Stand 24.02.2022]

[29] Elsys: *Operating Manual ERS Lite.pdf;* abrufbar unter: https://www.elsys.se/en/documents-firmware/ [Online: Stand 24.02.2022]

[30] Dragino: *LHT65_Temperature_Humidity_Sensor_UserManual_v1.8.5.pdf;* https://www.dragino.com/downloads/index.php?dir=LHT65/UserManual/ [Online: Stand 24.02.2022]

[31] RAK: *RAK7204 Quick Start Guide;* https://docs.rakwireless.com/Product-Categories/WisNode/RAK7204/Overview/ [Online: Stand 24.02.2022]

[32] RAK: *RAK7204 AT Command Manual;* https://docs.rakwireless.com/Product-Categories/WisNode/RAK7204/Overview/ [Online: 24.02.2022]

[33] Laird Connectivity: *CS-GUIDE-RG1xx_RG191+LTE v6_1.pdf;* Version 6.1; abrufbar unter: https://www.lairdconnect.com/wireless-modules/lorawan-solutions/sentrius-rg1xx-lorawan-gateway-wi-fi-ethernet-optional-lte-us-only [Online: Stand 25.02.2022]

[34] Kerlink-Wiki: https://wikikerlink.fr/wirnet-productline/doku.php?id=wiki:quickstart:quickstart_ifemto [Online: Stand 25.02.2022]

[35] GitHub: *LoRa packet_forwarder;* https://github.com/Lora-net/packet_forwarder/blob/master/PROTOCOL.TXT [Online: 25.02.2022]

[36] CODESYS Store: *Download CODESYS Development System V3;* https://store.codesys.com/de/codesys.html [Online: Stand 27.02.2022]

[37] CODESYS: *CODESYS – Das System;* https://de.codesys.com/das-system.html [Online:28.02.2022]

[38] Maaß, Jochen.: Vorlesungsunterlagen „Prozessautomatierung“; Wintersemester 2021; abrufbar unter: https://emil.haw-hamburg.de/course/view.php?id=53440 [Online: Stand 28.02.2022]

[39] Mandl, Peter: *TCP und UDP Internals – Protokolle und Programmierung;* SpringerVieweg (2018)

[40] Webseite: *Introducing JSON;* https://www.json.org/json-en.html [Online: Stand 03.03.2022]

[41] ecma international: *ECMA-404 – The JSON data interchange syntax*; 2. Edition (Dezember 2017); abrufbar unter: https://www.ecma-international.org/publications-and-standards/standards/ecma-404/ [Online: Stand 03.03.2022]

[42] Heinrich,Berthold; Linke, Petra; Glöckler, Michael: *Grundlagen Automatisierung*; 3. Auflage (2020); SpringerVieweg, Seite 272

[43] Josefsson,S.: *RFC 4648 - The Base16, Base32, Base64 Data Encodings;* Oktober 2006; https://datatracker.ietf.org/doc/html/rfc4648#section-4 [Online: Stand 04.03.2022]

[44] CODESYS Store: *OSCAT NETWORK;* Version 1.3.5.2; https://store.codesys.com/oscat-network.html [Online: Stand 04.03.2022]

[45] OSCAT: Dokumentation Network Library; *oscat_netlib121_de.pdf;* Version 1.21; http://www.oscat.de/component/jdownloads/summary/3-oscat-network/12-oscat-netlib121-de.html [Online: Stand 04.03.2022]

[46] LoRaWAN 1.0.x packet decoder; https://lorawan-packet-decoder-0ta6puiniaut.runkit.sh/ [Online: Stand 05.03.2022]

[47] Papula, Lothar: *Mathematik für Ingenieure und Naturwissenschaftler – Band 3;* 5. Auflage (2008); Vieweg+Teubner

[48] CODESYS Forge: *Crypto Example;* https://forge.codesys.com/prj/codesys-example/crypto-example/home/Home/ [Online: 06.03.2022]

[49] Laird Connectivity - Application Note: *CS-AN-RS1xx-LoRa-Protocol v2_12_1.pdf;* https://www.lairdconnect.com/documentation/application-note-rs1xx-lora-protocol; [Online: Stand 06.03.2022]

[50] Elsys: *Sensor_payload.pdf*; Version 1.11 abrufbar unter: https://elsys.se/public/documents/Sensor_payload.pdf [Online: Stand 07.03.2022]

[51] heise Security; Blogartikel; Schmidt, Jürgen: *Kryptographie in der IT – Empfehlungen zu Verschlüsselung und Verfahren;* https://www.heise.de/security/artikel/Kryptographie-in-der-IT-Empfehlungen-zu-Verschluesselung-und-Verfahren-3221002.html#mobile_detect_force_desktop [Online: Stand 08.03.2022]

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, den _____ _____