Demonstrator für eine dezentralen Regelung von steuerbaren Lasten

Masterarbeit zur Erlangung des akademischen Grades
Master of Science
im Studiengang Erneuerbare Energien
an der Fakultät für Anlagen, Energie- und Maschinensysteme
der Technischen Hochschule Köln

vorgelegt von: Sven Michael Lorre
Adresse: Sven.lorre@smail.th-koeln.de

Betreuer: Prof. Dr. Eberhard Waffenschmidt
Ko-Referent: Prof. Dr. Ingo Stadler

Köln. 30.06.2025

Erklärungen

Name: Sven Michael Lorre
Matrikel-Nummer: 11123546

Erklärung zum eigenständigen Verfassen

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbständig verfasst habe. Ich habe keine anderen außer den von mir angegeben Quellen und Hilfsmittel verwendet
Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Köln, 30.Juni 2025

Sven Michael Lorre

Erklärung zur Nutzung von Hilfsmitteln

Bei der Erstellung dieser Arbeit, besonders beim Erstellen der Software, wurde ChatGPT zur Hilfe beim Debugging eingesetzt. Auch wurde ChatGPT als Formulierungshilfe im Text eingesetzt. Dies ist mit dem Betreuer der Arbeit abgesprochen und genehmigt worden.

Köln, 30.Juni 2025

Sven Michael Lorre

Erklärung zur Veröffentlichung

Ich bin damit einverstanden, dass meine Abschlussarbeit ausgeliehen werden darf. Sie darf von meinem Betreuer im Internet veröffentlicht werden.

Köln, 30.Juni 2025

Sven Michael Lorre

Erklärung zu Bildrechten

Außer den im Folgenden genannten habe ich alle Bilder und Diagramme dieser Abschlussarbeit selbst erstellt.

Köln, 30.Juni 2025

Sven Michael Lorre

Kurzfassung

Diese Arbeit beschreibt den Aufbau eines Demonstrators zur dezentralen Regelung von steuerbaren Lasten. Hierbei wird die Entwicklung einer Softwarelösung zur dezentralen Kommunikation und Regelung der im MicroGrid der TH Köln angeschlossenen regelbaren Lasten und der konkrete Aufbau und Betrieb des Demonstrators dokumentiert. Diese Arbeit stellt einen Teil des Forschungsprojekts GridMaximizer der TH Köln dar. Der Demonstrator besteht aus einem Netzwerk aus 3 Raspberry Pis zur Veranschaulichung der Dezentralität, wobei zwei der Raspberry Pis an regelbare Lasten angeschlossen sind und ein Raspberry Pi an eine nicht steuerbare Last. Alle Raspberry Pis liefern in dieser Konfiguration Messwerte und tauschen diese aus. Ein Raspberry Pi optimiert das Netz daraufhin aufgrund dieser Messwerte. Im Betrieb des Demonstrators können so mithilfe der beiden regelbaren Lasten Ladevorgänge simuliert werden, welche durch das Zuschalten der nicht steuerbaren Last an einem anderen Knoten beeinflusst werden.

Abstract

This thesis documents the construction of a demonstrator for the decentralized control of controllable loads. It details the development of a software solution for decentralized communication and regulation of the controllable loads connected to the TH Köln microgrid, as well as the concrete setup and operation of the demonstrator. This work is part of the TH Köln research project GridMaximizer. The demonstrator consists of a network of three Raspberry Pis to showcase the decentralization. Two of the Raspberry Pis are connected to controllable loads and one to a non-controllable load. In this configuration, all Raspberry Pis measure data and exchange it with one another. One Raspberry Pi then optimizes the network based on these measurements. During operation, the demonstrator can simulate charging processes using the two controllable loads, which are influenced by the activation of the non-controllable load at another node.

Inhaltsverzeichnis

Erklärungen ..... **II**
Erklärung zum eigenständigen Verfassen ..... **II**
Erklärung zur Nutzung von Hilfsmitteln ..... **II**
Erklärung zur Veröffentlichung ..... **II**
Erklärung zu Bildrechten ..... **II**
Kurzfassung ..... **III**
Abstract ..... **III**
Inhaltsverzeichnis ..... **IV**
Tabellenverzeichnis ..... **VII**
Abbildungsverzeichnis ..... **VII**
Abkürzungsverzeichnis ..... **IX**
1. Einleitung **11**
2. Stand der Technik **12**
2.1. Steuerbare Lasten im Kontext der Energiewende **12**
2.1.1. Steuerbox im Kontext mit SMGW und dem §14a: Die Grundlage der netzorientierten Steuerung **12**
2.1.2. Schnittstellen der Steuerbox **13**
2.1.3. Steuerungskonzepte der Steuerbox **14**
2.1.4. Anforderungen an eine digitale Schnittstelle **20**
2.1.5. Forschungsprojekte zur SteuerBox nach FNN **26**
2.1.6. Relevanz der Steuerbox für den Aufbaus des Demonstrators **28**
2.2. Experimente und Versuche der Netzbetreiber **29**
2.2.1. Gesteuerte Laden von Elektrofahrzeugen über Preisanzeige **29**
2.2.2. Netzlabor E-Mobility Allee **29**
2.2.3. Netzlabor E-Mobility Carré **31**
2.3. Zellulare Netze **31**
2.4. GridMaximizer **32**
2.5. Gleichzeitigkeitsfaktor **32**
2.6. Bedeutung für diese Arbeit **33**
3. Hintergrund **34**
3.1. Technologische Grundlagen der Systemarchitektur **34**
3.1.1. Steuerbare Lasten und §14a EnWG **34**
3.1.2. Kommunikationsprotokolle zur Gerätesteuerung **35**
3.1.3. Kommunikationsinfrastruktur und Datenaustausch **37**
3.1.4. Kommunikationslogik und Geräteintegration **38**
3.1.5. Docker Container im Kontext der Softwareentwicklung **39**

4. Inhalt 40
4.1. Neuigkeit der Ergebnisse 40
4.2. Eigener Beitrag 40
4.3. Systemkonzept und Anforderungen 40
4.3.1. Übersicht über die benötigte Software und die Anforderungen an diese zur Umsetzung des Demonstrators 40
4.3.2. Hardwarevoraussetzungen zur Umsetzung des Demonstrators 44
4.4. Systemarchitektur und Modulkategorien 45
4.5. Detaillierte Beschreibung der Software 46
4.5.1. Python Bibliothek „devicecom“ 46
4.5.1.1. *Device_register* 47
***4.5.1.2. Selectors* 47**
4.5.1.3. *Controllable* und *Measurement* 47
4.5.1.4. *Device* 48
4.5.2. REST-API-Schnittstelle 48
4.5.2.1. Endpunkt /{device_id}/subscribe 50
4.5.2.2. Endpunkt /{device_id}/latest_measurement 50
4.5.2.3. Endpunkt /{device_id}/goal 51
4.5.2.4. Endpunkt /{device_id}/control 51
4.5.2.5. Endpunkt „send_charging“ 51
4.5.2.6. Endpunkt /meter_values/{cp_id} 52
4.5.2.7. Endpunkt /test/apply_current/{device_id} 52
4.5.2.8. Endpunkt /test/send_charging_profile/{device_id} 52
4.5.3. Integrierter OCPP-Server 53
***4.5.3.1. on_connect* 53**
4.5.3.2. *Boot_notification* 53
4.5.3.3. *on_Heartbeat* 54
4.5.3.4. *on_status_notification* 54
4.5.3.5. *On_authorize* 54
***4.5.3.6. trigger_meter_values* und *on_meter_values* 54**
4.5.3.7. *send_charging_profile* 55
4.5.3.8. *on_start_transaction* und *on_stop_transaction* 55
4.5.4. Central Node 56
4.5.4.1. Broadcast-Zyklus 57
4.5.4.2. Receive Zyklus 57
4.5.4.3. Optimierungszyklus 58
4.5.5. Grafische Oberfläche GUI 60

4.6. Darstellung der zentralen Funktionen der Software und Vergleich mit dem Konzept der Steuerbox 62
5. Aufbau und Funktionsweise des Demonstrators 65
6. Darstellung und Diskussion der Ergebnisse 70
6.1. Darstellung der Ergebnisse 70
6.2. Diskussion der Ergebnisse 71
7. Fazit und Ausblick 73
Literatur 74
Anhang 77

Tabelle 1: Funktionen der digitalen Schnittstelle der Steuerbox, nach [3]. 21
Tabelle 2: Nachrichteninhalt und Länge, vgl. [5]. 41

Tabellenverzeichnis

Tabelle 1: Funktionen der digitalen Schnittstelle der Steuerbox, nach [3]. 21
Tabelle 2: Nachrichteninhalt und Länge, vgl. [5]. 41

Abbildungsverzeichnis

Abbildung 1: Schematische Darstellung der Steuerbox FNN, eigene Darstellung nach [3]. 14
Abbildung 2: Steuerungskonzept 1: 1 SteuVE digital, eigene Darstellung nach [3]. 15
Abbildung 3: Steuerungskonzept 2: 1-4 SteuVE digital, eigene Darstellung nach [3]. . 15
Abbildung 4: Steuerungskonzept 3: EMS Digital, eigene Darstellung nach [3]. 16
Abbildung 5: Steuerungskonzept 4: 1 EZA digital, 1-n SteuVE Relais, eigene Darstellung nach [3]. 16
Abbildung 6: Steuerungskonzept 5: 1 EZA digital, 1 SteuVE digital, 1-n SteuVE Relais, eigene Darstellung nach [3]. 17
Abbildung 7: Steuerungskonzept 6: EMS digital mit 1 SteuVE, eigene Darstellung nach [3]. 17
Abbildung 8: Steuerungskonzept 7: 1-4 SteuVE digital plus EMS, eigene Darstellung nach [3]. 18
Abbildung 9: Steuerungskonzept 8: 1 digitales EMS plus 1-n SteuVE Relais, eigene Darstellung nach [3]. 18
Abbildung 10: Verantwortungsbereiche Messstellenbetreiber und Anlagenbetreiber, eigene Darstellung nach [3]. 22
Abbildung 11: Verantwortungsbereich MSB und Betreiber der Anlage bei Integration der Steuerbox in das SMGW, eigene Darstellung nach [3]. 23
Abbildung 12: Zentrale Funktionen der Steuerbox, eigene Darstellung nach [3] 25
Abbildung 13: Application Layer des EEBUS Protokolls, eigene Darstellung nach [3] 25
Abbildung 14: Protocol layers von ModbusTCP nach [31] 35
Abbildung 15: Funktionen des OCPP, eigene Darstellung nach [32] 36
Abbildung 16: Protocol Layers des OCPP, eigene Darstellung nach [32] 37
Abbildung 17: Ablaufdiagramm der entwickelten Software, eigene Darstellung 46
Abbildung 18: Darstellung der REST-API Endpunkte und deren Gegenspieler, eigene Darstellung. 49
Abbildung 19: OCPP-Server mit Funktionen, eigene Darstellung 53
Abbildung 20: Aussetzung der Validierung, eigene Darstellung 54
Abbildung 21: Darstellung eines OCPP konformen Charging Profiles, eigene Darstellung 55
Abbildung 22: Ablaufdiagramm Central_Node, eigene Darstellung. 56
Abbildung 23: Design der GUI, eigene Darstellung. 60
Abbildung 24: Live Leistungsbezugsanzeige der GUI, eigene Darstellung. 61
Abbildung 25: Strom und Spannung an den Knoten, eigene Darstellung. 61
Abbildung 26: Steuerungskonzept Demonstrator, eigene Darstellung 62
Abbildung 27: Resultierende Protocol Layer aus der Kombination von Modbus TCP und OCPP, eigene Darstellung 63
Abbildung 28: Zentrale Funktionen der entwickelten Software, eigene Darstellung. 64

Abbildung 29: Erfolgreiche Verbindung zwischen Wallbox und OCPP-Server, eigene Darstellung 65
Abbildung 30: Demonstratornetzwerk, eigene Darstellung. 66
Abbildung 31: Bestätigung des Abonnements der Messwerte, eigene Darstellung. 66
Abbildung 32: Logging zur Funktion "meter_values", eigene Darstellung. 66
Abbildung 33: Versand und Empfang der Daten, eigene Darstellung. 67
Abbildung 34: Quittierung der goal Funktion, eigene Darstellung. 67
Abbildung 35: Beispielhafte Geräteparameter für die Nachricht und die Optimierung, eigene Darstellung. 68
Abbildung 36: Versand der optimierten Ströme und setzen desselben, eigene Darstellung. 68
Abbildung 37: Bestätigung der Annahme des gesendeten Charging Profiles, eigene Darstellung. 69
Abbildung 38: Erfolgreiche Ausführung der Testfunktion apply_current, eigene Darstellung. 69
Abbildung 39: Stromverlauf an den Netzknoten in [A], eigene Darstellung 70
Abbildung 40: Verlauf des Gleichzeitigkeitsfaktors für das Demonstratornetz, eigene Darstellung. 71

Abkürzungsverzeichnis

5G Fifth Generation Mobile Network 45
A Ampere..... IX, 45, 48, 50, 68, 73, 77
ADU Application Data Unit 36, 66
API Application Programming Interface .....
..... VI, VIII, 39, 40, 41, 42, 46, 48, 49, 50, 51, 61, 63, 64, 70, 71, 72
AR Anwendungsregel..... 23, 25, 27
BDEW Bundesverband der Energie- und Wasserwirtschaft 27, 29, 35
BK Beschlusskammer..... 24, 25
BPL Broadband Powerline 44
BSI Bundesamt für Sicherheit in der Informationstechnik..... 23, 24
BW Baden-Württemberg 13, 29, 30
C/sells Schaufenster-Projekt C/sells – „zellulares Energiesystem für Baden-Württemberg, Bayern und Hessen“ 31, 32
CLS Controllable Local System 23, 24, 25, 28, 32, 65
cp charge-point ..... VI, 52, 54, 63
DECT Digital Enhanced Cordless Telecommunications..... 44
DNS-SD Domain Name System Service Discovery 26, 27
EAF Energiewirtschaftlicher Anwendungsfall..... 28
EEBUS *European Energy BUS* (Kommunikationsstandard) .....
..... VIII, 20, 23, 26, 27, 29, 36
ELBE Projekt ELBE – Elektrisch Laden für Berlin..... 30
EMS Energy Management System..... VIII, 14, 15, 16, 17, 18, 19, 25, 27, 28
EnWG Energie Wirtschaftsgesetz ..... V, 12, 13, 24, 27, 28, 35
ETH Ethernet..... 14, 18, 19, 26
EZA Energieerzeugungsanlage..... VIII, 14, 15, 17, 18, 19
FNN Forum Netztechnik/Netzbetrieb im VDE .....
..... V, VIII, 13, 14, 15, 24, 25, 27, 28, 45, 75
GUI Graphical User Interface.....
..... VI, VIII, 44, 45, 46, 50, 53, 60, 61, 63, 64, 69, 73, 76
HTTP Hypertext Transfer Protocol 39
IDE Integrated Development Environment 40
IEEE Institute of Electrical and Electronics Engineers 37
IP Internet Protocol 20, 26, 27, 36, 46, 48, 49, 66, 68
ISI Fraunhofer-Institut für System- und Innovationsforschung 29
JSON JavaScript Object Notation..... 20, 27, 37, 66
kbit Kilobit..... 44
KEMAL Kundenorientiertes Energiemanagement mit autonomer Lastregelung .....
..... 27, 28
kW Kilowatt..... 13, 30, 31, 35
LamA Laden am Arbeitsplatz..... 30
LoRa Long Range..... 44, 45, 78
LPC Limitation of Active Power Consumption 25, 27
LPP Limitation of Active Power Production..... 25, 27
LTE Long Term Evolution .....
..... 28
mDNS Multicast Domain Name System..... 26, 27
MGCP Monitoring of Grid Connection Point 25, 27
MHz Megahertz 28
MPC Monitoring of Power Consumption..... 25, 27

MQTT Message Queuing Telemetry Transport 20
MSB Messstellenbetreiber ..... VIII, 23, 24, 39
NR New Radio 44
OCPP Open Charge Point Protocol ..... VI, VIII, IX, 12, 20, 29, 30, 34, 36, 37, 38, 39, 40, 41, 45, 46, 50, 53, 54, 55, 56, 57, 61, 63, 64, 65, 66, 68, 71, 79
PDU Protocol Data Unit 36, 66
PLC Powerline Communication 44, 78
PQIDA Power Quality Instrument – Device A 41, 48, 63, 68
PV Photovoltaik ..... II, 13
RAK RAKwireless – Hersteller von LoRa-Modulen 44
Redis Remote DIctionary Server 38, 39, 40, 41, 42, 45, 46, 52, 53, 55, 56, 60, 61, 63, 64, 69, 70
REST Representational State Transfer ..... VIII, 39, 40, 41, 48, 49, 50, 51
RFID Radio Frequency Identification 56
SHIP Smart Home IP 26, 27
SINTEG Schaufenster Intelligente Energie 31
SKI Sicherheitskennzahl für die Identifikation 23
SMGW Smart Meter Gateway ..... V, VIII, 12, 13, 15, 23, 24, 25, 27, 28, 29, 32, 34, 35, 45, 65
soc State of Charge 53, 60, 62
SPINE Smart Premises Interoperable Neutral-message Exchange 26, 27
STB Steuerbox 14
SteuVE Steuerbare Verbrauchseinrichtung ..... VIII, 15, 16, 17, 18, 19, 35
SYP System Integrations Praktikum 41
TCP Transmission Control Protocol ..... VIII, 12, 20, 26, 27, 29, 36, 38, 39, 65, 66, 68, 76
TH Technische Hochschule ..... III, 12, 29, 32, 41, 44, 45, 63, 68, 76
TLS Transport Layer Security 24, 26, 27
TR Technische Richtlinie 24
TTL Time To Live 38
UDP User Datagram Protocol 38, 39, 43, 60, 70
UMD Universal Measurement Device 41, 45, 48, 63
VDE Verband der Elektrotechnik Elektronik Informationstechnik e. V. 23, 24, 25, 27, 29
VDI Verein Deutscher Ingenieure 29
VSI Voltage Source Inverter 50
WAN Wide Area Network 28
WIFI Wireless Fidelity 37
WLAN Wireless Local Area Network 20

1. Einleitung

Im Rahmen der Energie- und Mobilitätswende wird der Hauptanteil des Wärme- und Verkehrssektors zunehmend elektrifiziert [1]. Diese Entwicklung führt insbesondere durch Haushaltslasten und Wallboxen im Niederspannungsnetz zu einer steigenden Auslastung der Betriebsmittel, die durch die zunehmende Zahl an steuerbaren Lasten wie Ladeinfrastruktur für Elektrofahrzeuge, Wärmepumpen und Batteriespeicher zusätzlich verschärft werden [2]. Um einem kostspieligen und zeitintensiven Netzausbau entgegenzuwirken, bedarf es intelligenter Lösungen zur netzdienlichen Steuerung solcher Lasten.
Ein Ansatz für eine netzdienliche Steuerung ist die zentrale Steuerung über Smart-Meter-Gateways (SMGW) in Kombination mit Forum Netztechnik/Netzbetrieb-konformen Steuerboxen, [3], wie sie im Rahmen des §14a Energiewirtschaftsgesetz (EnWG) [4] vorgesehen ist. Diese Lösung bringt jedoch erhebliche Koordinations- und Infrastrukturanforderungen auf Seiten der Netzbetreiber mit sich.
Im Gegensatz dazu verfolgt das Forschungsprojekt GridMaximizer der Technischen Hochschule Köln einen dezentralen Regelungsansatz, bei dem die Regelung der Lasten direkt an den Netzknoten, an denen die Haushaltsanschlüsse angeschlossen sind, erfolgt. Ziel dieses Projekts ist der Aufbau eines funktionalen Demonstrators zur verteilten Steuerung von Lasten im Niederspannungsnetz, der im institutseigenen MicroGrid der TH Köln realisiert und getestet wird [5].
In der vorliegenden Masterarbeit wird die dezentrale Kommunikation und Steuerung der Lasten im MicroGrid realisiert. Ein Schwerpunkt liegt dabei auf der Entwicklung und Dokumentation der dezentralen Kommunikationsinfrastruktur zwischen den verwendeten Microcomputern (Raspberry Pi), der Einbindung der im MicroGrid vorhandenen steuerbaren Lasten in die dezentrale Kommunikation und Steuerung, sowie der Integration des Open Charge Point Protocol (OCPP) in die vorhandene Softwarearchitektur zur Steuerung von Ladeeinrichtungen.
Klar abgegrenzt vom Inhalt dieser Arbeit ist die konkrete Entwicklung und Umsetzung des Optimierungsalgorithmus zur netzdienlichen Steuerung. Stattdessen liegt der Beitrag dieser Arbeit in der Bereitstellung und dem Test der kommunikativen Infrastruktur, einschließlich der Datenerfassung, Vorverarbeitung und Übermittlung an die Optimierung sowie der praktischen Anbindung realer Geräte über standardisierte Schnittstellen. Auch die Einbindung einer Netzzustandsschätzung (Grid State Estimation) soll nicht in dieser Arbeit umgesetzt werden.
Ziel dieser Arbeit ist es einen Demonstrator zu entwickeln, der auf einem dezentralen Regelungskonzept basiert. Mithilfe von Raspberry-Pi-basierten Steuerknoten, die über ein eigenes Kommunikationsnetzwerk Informationen austauschen, werden die Grundvoraussetzungen für eine Optimierung an einem der Knoten geschaffen. Der Demonstrator zeigt, dass ein verteiltes Steuerungssystem auch ohne zentrale Steuerbox oder SMGW funktionsfähig und skalierbar ist. Eine eigens entwickelte Python-Bibliothek *devicecom* übernimmt dabei die Kommunikation mit den angeschlossenen Geräten über etablierte Schnittstellen wie OCPP und Modbus TCP.
Im Folgenden werden die relevanten technischen Grundlagen sowie bestehende Lösungen zur Regelung dezentraler Lasten vorgestellt. Dabei wird ein besonderer Fokus auf die

technologische Einordnung der verwendeten Schnittstellen und Kommunikationsmethoden gelegt.

2. Stand der Technik

2.1. Steuerbare Lasten im Kontext der Energiewende

Insbesondere die zunehmende Verbreitung von Elektrofahrzeugen in Privathaushalten führt zu dynamischeren Lastgängen und teils erheblichen Lastspitzen im Tagesverlauf [2]. Um diesen Herausforderungen zu begegnen, wurden zahlreiche Studien durchgeführt, die die Verträglichkeit von Ladevorgängen von Elektrofahrzeugen im heutigen Stromnetz analysieren. Der Fokus liegt dabei meist auf zentral organisierten Lademanagementsystemen, wie sie etwa von Netzbetreibern wie der Netze BW umgesetzt werden, siehe Kapitel 2.2.

Im Zentrum der regulatorischen Bemühungen steht der überarbeitete §14a EnWG [4], der seit 2021 Netzbetreiber dazu verpflichtet, steuerbare Verbrauchseinrichtungen über 4,2 kW, insbesondere Ladeeinrichtungen für Elektrofahrzeuge, Wärmepumpen und stationäre Batteriespeicher, in ein netzorientiertes Lastmanagement einzubinden. Netzorientiert bedeutet, dass steuerbare Lasten im Falle einer höheren Belastung des Netzstranges kurzzeitig heruntergeregelt werden können, um die Netzbetriebsmittel zu entlasten. Endkunden erhalten im Gegenzug reduzierte Netzentgelte. Die technische Umsetzung erfolgt dabei über sogenannte Steuerungseinrichtungen, die von Netzbetreibern oder beauftragten Dienstleistern gesteuert werden können. Parallel dazu treiben viele Hersteller die Entwicklung lokaler Energiemanagementsysteme mit dynamischer Steuerung weiter voran. Diese ermöglichen unter anderem ein automatisiertes PV-Überschussladen oder eine netzorientierte Lastverlagerung auf Haushaltsebene. [3], [4], [6]

2.1.1. Steuerbox im Kontext mit SMGW und dem §14a: Die Grundlage der netzorientierten Steuerung

Um dem §14a gerecht werden zu können ist unter anderem die Technologie der sogenannten Steuerbox vom Forum Netztechnik/Netzbetrieb (FNN) erforscht und in einem Leitfaden zusammengefasst worden. Diese bietet für den Netzbetreiber eine Schnittstelle vom Smart Meter Gateway ausgehend zu den Steuerbaren Lasten, welche mithilfe der Steuerbox zentral vom Netzbetreiber geschaltet werden können. [3]
Hierbei gibt es Steuerboxen, die die Schalthandlungen über Relais umsetzten und solche, die über eine digitale Schnittstelle verfügen, um die Steuerbefehle durchzusetzen. Die Nutzung der Relais ist derzeit noch gebräuchlich, allerdings nicht mehr erwünscht, da es beim Schalten mit den Relais stets nur zwei Zustände geben kann und es auch keine Rückmeldung von der geschalteten Seite gibt, welche aufzeigt, ob die Schalthandlung durchgeführt wurde, vgl. [3].

Deutlich besser umgesetzt ist dies bei der digitalen Schnittstelle. Diese ermöglicht das stufenlose Einstellen der gewünschten Parameter und bietet auch die Möglichkeit eine Rückmeldung der geschalteten Systeme zu erhalten, was bei der Fehlersuche und Fehlerbehebung helfen kann, vgl. [3].
Eine weitere Einschränkung der Steuerbox, welche mithilfe von Relais steuern soll, liegt in der maximalen Anzahl der steuerbaren Geräte. Diese wird bei der Steuerung mit Relais mit einer maximalen Anzahl von vier Geräten pro Steuerbox angegeben vgl. [3]. Zusätzlich zur Nutzung einer standardisierten digitalen Schnittstelle ist auch die Nutzung eines Energie Management Systems von großem Vorteil. So kann der Betreiber der zu steuernden Anlagen gewährleisten, dass alle Anlagen welche gemäß §14a steuerbar sein müssen gesteuert werden können, andere Geräte/Anlagen allerdings weiterhin bedarfsgerecht optimiert werden können, um zum Beispiel die gewünschten Eigenverbrauchsquoten zu erfüllen. [6]

2.1.2. Schnittstellen der Steuerbox

Die Steuerbox verfügt hierbei über mehrere Schnittstellen, welche schematisch in der Abbildung *1* dargestellt sind. Die erste ist die digitale Schnittstelle ETH1, welche als Schnittstelle für eine beliebige Anzahl an steuerbaren Einrichtungen mit digitaler Schnittstelle dienen kann. Diese ist als Ethernet Schnittstelle umgesetzt. Um diese beliebige Anzahl zu erreichen, wird allerdings ein Netzwerk oder ein Energy-Management-System benötigt, welches die Verbindungen bündelt. Des Weiteren verfügt die Steuerbox mit dem Relais W4 eine Möglichkeit zur Steuerung einer Steuerbaren Verbrauchseinrichtung über einen Einzelkontakt und kann somit gedimmt werden. Mithilfe der Relais S1-W3, kann die Steuerbox FNN eine Steuerbare Erzeugungseinrichtung in den Stufen 0 %, 30 %, 60 % und 100 % steuern. Wird das Relais W4 mit einbezogen, so kann auch ein Energie Management System (EMS) mithilfe der Relais gesteuert werden. Grundsätzlich ist die Belegung der Schnittstellen so vorgesehen, dass alle steuerbaren Einrichtungen mit digitaler Schnittstelle an die Schnittstelle ETH1 angeschlossen werden. Gibt es mehrere Steuerbare Einrichtungen mit digitaler Schnittstelle, so sind diese mithilfe einer geeigneten Netzwerkvervielfachung, wie einem Netzwerk oder einem EMS an die Schnittstelle ETH1 der Steuerbox anzuschließen. Steuerbare Verbrauchseinrichtungen wie z.B. eine Ladeeinrichtung für Elektrofahrzeuge, eine Klimaanlage, eine Wärmepumpe oder ein Speicher sind grundsätzlich an das Relais W4 der Steuerbox anzuschließen, vgl. [3]. Alle Energieerzeugungsanlagen (EZA) sind an die Relais S1 und W3 anzuschließen. Die schematische Darstellung der Steuerbox (STB) ist in Abbildung 1 dargestellt.

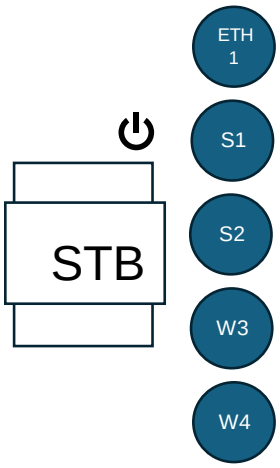

Abbildung 1: Schematische Darstellung der Steuerbox FNN, eigene Darstellung nach [3].

Sollen mehr als vier Steuerbare Einrichtungen eingebunden werden ist eine zweite Steuerbox nötig. Hierbei ist die Definition der Anzahl wie folgt:

- Jede Steuerbare Verbrauchseinrichtung (SteuVE) mit digitaler Schnittstelle = eine steuerbare Einrichtung
- Jede SeuVE mit Relais = eine steuerbare Einrichtung
- Jede Energieerzeugungsanlage (EZA) mit digitaler Schnittstelle = eine steuerbare Einrichtung
- Jede EZA mit Relais = eine steuerbare Einrichtung
- Jedes Energy Management System (EMS) mit digitaler Schnittstelle = eine steuerbare Einrichtung
- Jedes EMS mit Relais = eine steuerbare Einrichtung

2.1.3. Steuerungskonzepte der Steuerbox

Der Hinweis zu den Anforderungen an die Steuerbox und ihre Schnittstellen definiert hierbei 8 Steuerungskonzepte. Diese werden im Folgenden beschrieben.
In Abbildung 2 ist das Steuerungskonzept 1 dargestellt. In diesem verfügt der Betreiber über ein SMGW, die angeschlossene Steuerbox und eine Wärmepumpe mit einer digitalen Schnittstelle. Hierbei wird die Wärmepumpe direkt an die Ethernet-Schnittstelle der Steuerbox angeschlossen.

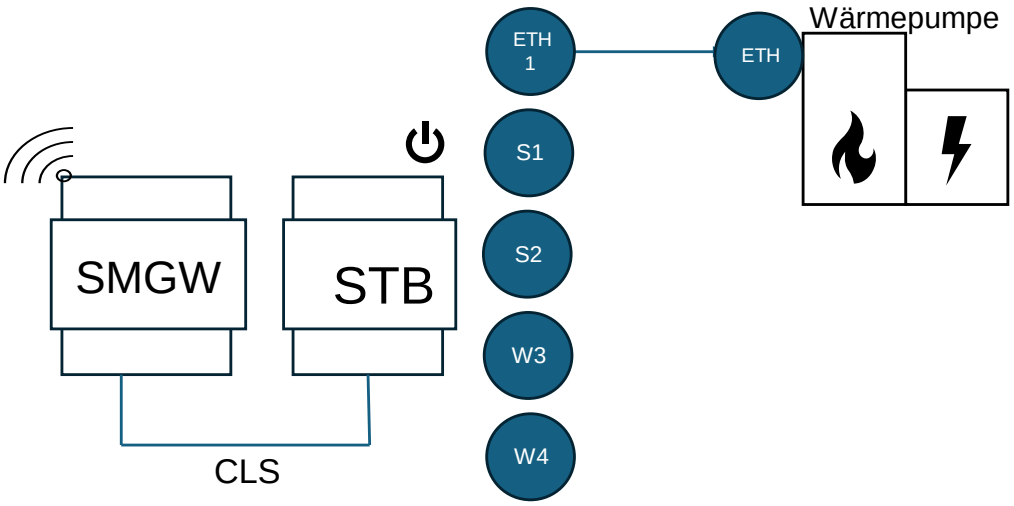

Abbildung 2: Steuerungskonzept 1: 1 SteuVE digital, eigene Darstellung nach [3].

Beim Steuerungskonzept 2, welches in Abbildung 3 dargestellt ist, verfügt der Betreiber über mehrere steuerbare Einrichtungen, welche alle in einem geeigneten Netzwerk verbunden sind. Dieses geeignete Netzwerk wird nun mit der Ethernet-Schnittstelle der Steuerbox verbunden. Dies ermöglicht eine beliebig große Anzahl von SteuVE einzubinden, vgl. [3].

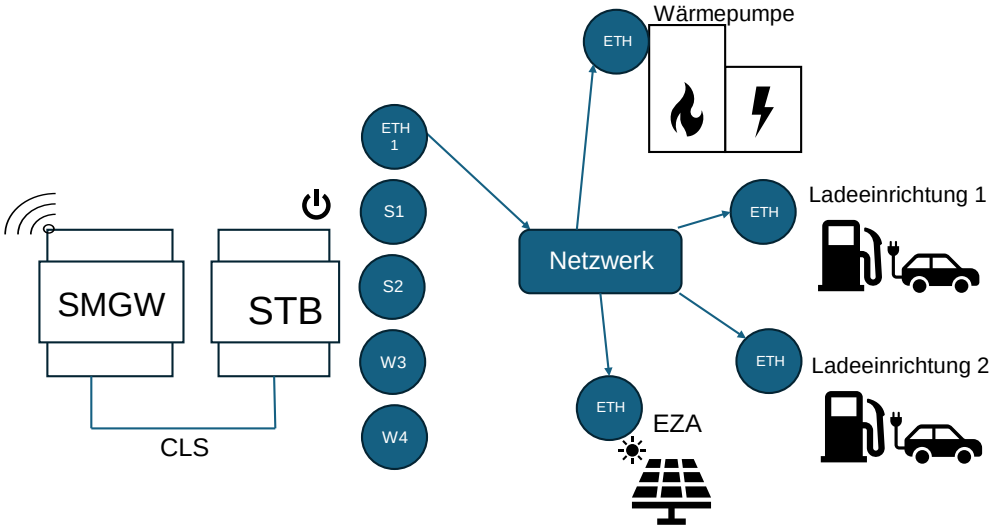

Abbildung 3: Steuerungskonzept 2: 1-4 SteuVE digital, eigene Darstellung nach [3].

Beim Steuerungskonzept 3 verfügt der Betreiber nicht nur über steuerbare Einrichtungen, sondern auch über ein Energy-Management-System (EMS), in welches alle steuerbaren Einrichtungen eingebunden sind. Dieses EMS ist wiederum mit der Ethernet-Schnittstelle der Steuerbox verbunden. Das gesamte Steuerungskonzept ist in Abbildung 4 dargestellt.

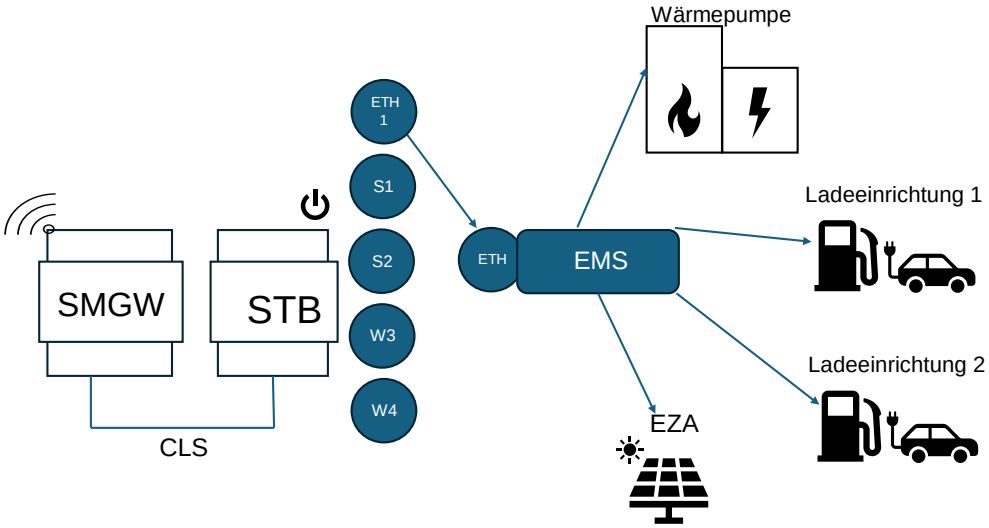

Abbildung 4: Steuerungskonzept 3: EMS Digital, eigene Darstellung nach [3].

Beim Steuerungskonzept 4 verfügt der Betreiber über mehrere Einrichtungen, eine mit einer digitalen Schnittstelle und zwei mit einer Relais-Steuerung. Hierbei wird wieder die Einrichtung mit digitaler Schnittstelle mit der Ethernet-Schnittstelle der Steuerbox verbunden und die beiden relaisgesteuerten Einrichtungen beide mit dem Relais W4 angesteuert. Dieses Steuerungskonzept ist in Abbildung 5 dargestellt.

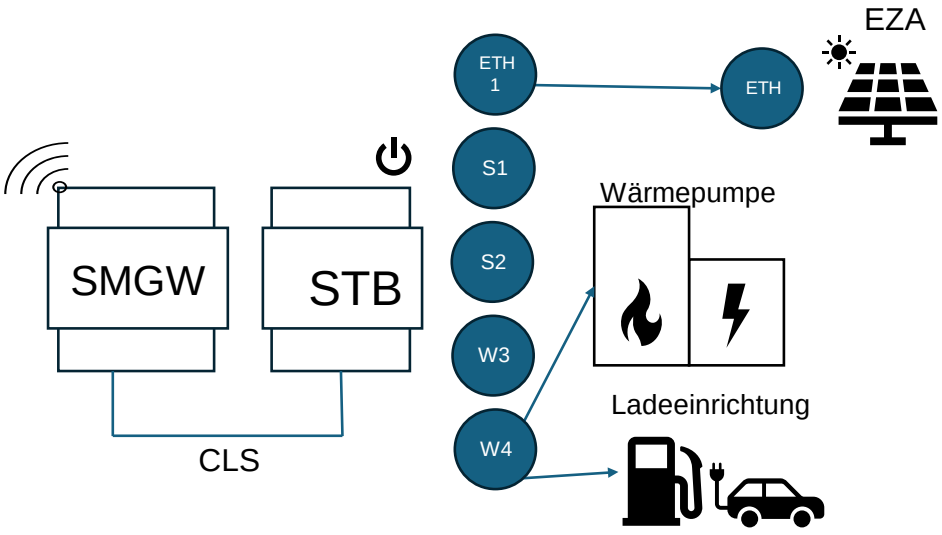

Abbildung 5: Steuerungskonzept 4: 1 EZA digital, 1-n SteuVE Relais, eigene Darstellung nach [3].

Beim Steuerungskonzept 5 verfügt der Betreiber wieder über ein Netzwerk mit beliebig vielen steuerbaren Einrichtungen mit digitaler Schnittstelle, welche in einem Netzwerk verbunden werden. Dieses Netzwerk wird wiederum mit der Ethernet-Schnittstelle der Steuerbox verbunden. Die beiden relaisgesteuerten Einrichtungen werden wieder an das

Relais W4 der Steuerbox angeschlossen und sind somit dimmbar. Das Gesamte Steuerungskonzept ist in Abbildung *5* dargestellt.

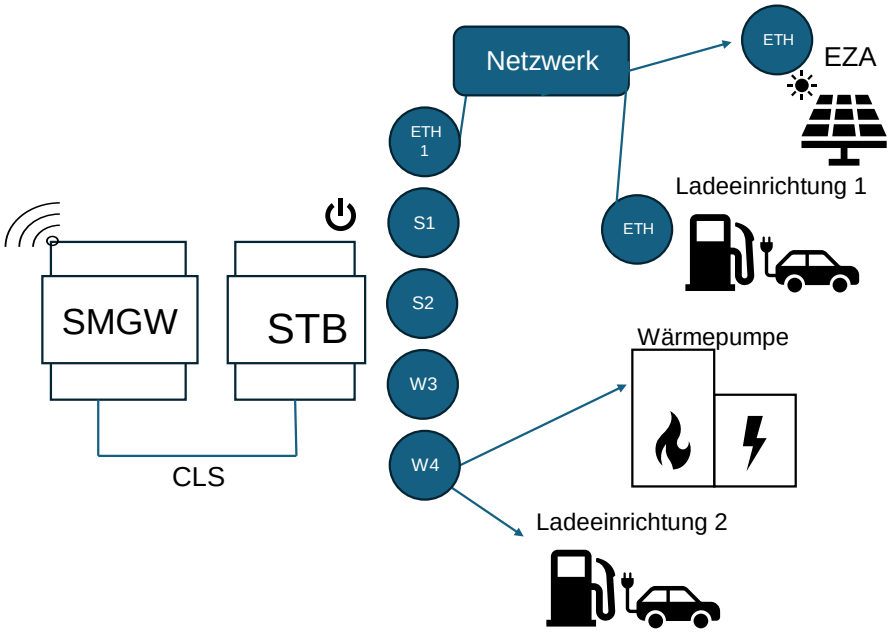

Abbildung 6: Steuerungskonzept 5: 1 EZA digital, 1 SteuVE digital, 1-n SteuVE Relais, eigene Darstellung nach [3].

Beim Steuerungskonzept 6 handelt es sich um ein ähnliches Steuerkonzept wie Steuerkonzept 3. Der Bertreiber hat hierbei eine SteuVE in Form einer Wärmepumpe, welches in ein EMS eingebunden ist. Dieses wird wiederum an die Schnittstelle ETH1 der Steuerbox angeschlossen. Das Steuerungskonzept 6 ist in Abbildung 7 dargestellt.

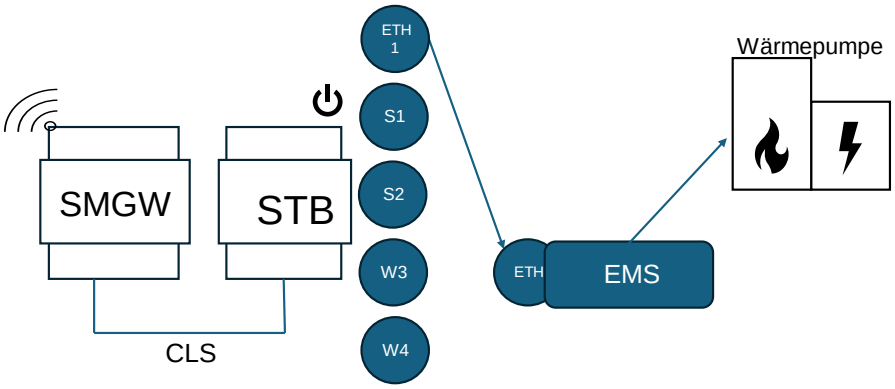

Abbildung 7: Steuerungskonzept 6: EMS digital mit 1 SteuVE, eigene Darstellung nach [3].

Beim Steuerungskonzept 7 verfügt der Betreiber über ein Netzwerk, welches direkt mit der Schnittstelle ETH1 der Steuerbox verbunden ist. Direkt in das Netzwerk ist eine Wärmepumpe mit digitaler Schnittstelle eingebunden. Eine weitere Wärmepumpe und eine

EZA sind über ein EMS auch in das Netzwerk eingebunden. Dieses Konzept ist in Abbildung 8 dargestellt.

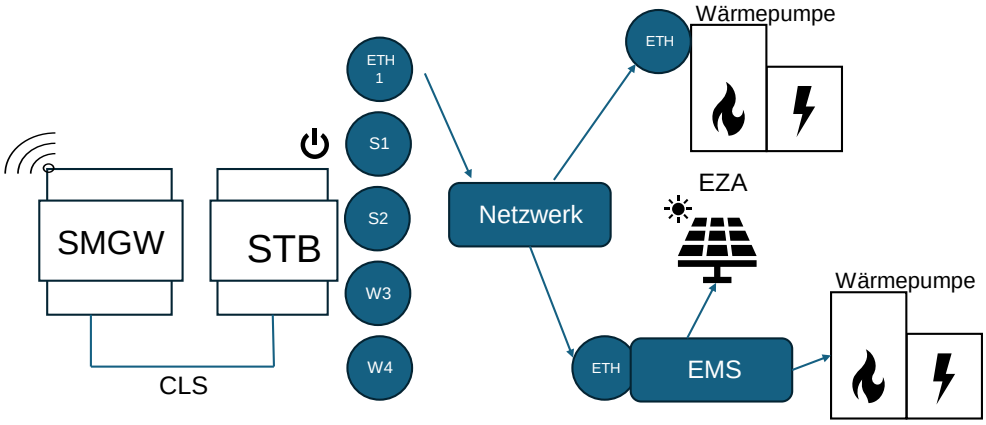

Abbildung 8: Steuerungskonzept 7: 1-4 SteuVE digital plus EMS, eigene Darstellung nach [3].

Beim Steuerungskonzept 8 verfügt der Betreiber über ein EMS, in welches eine EZA und eine Ladeeinrichtung über digitale Schnittstellen in ein EMS eingebunden, welches wiederum über die Schnittstelle ETH1 an die Steuerbox angeschlossen sind. Eine Wärmepumpe ist über das Relais W4 direkt angeschlossen, siehe Abbildung 9.

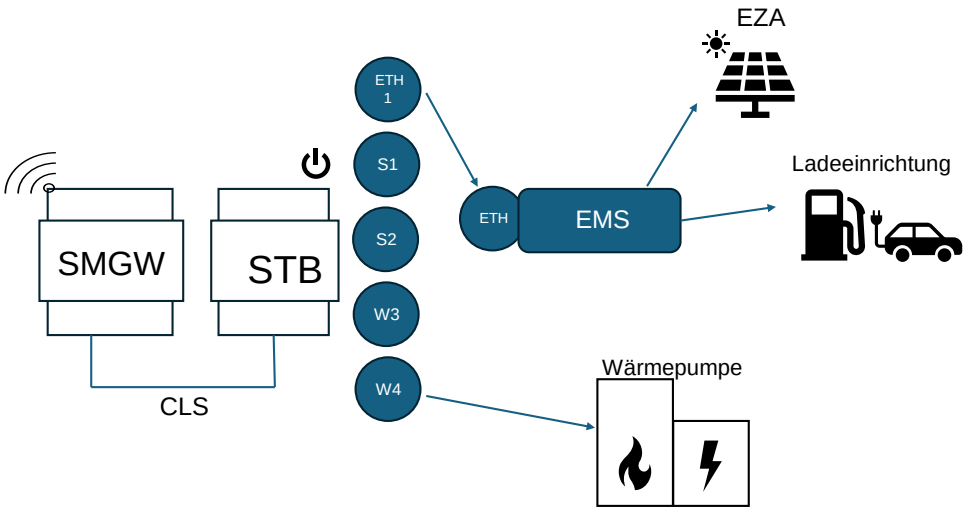

Abbildung 9: Steuerungskonzept 8: 1 digitales EMS plus 1-n SteuVE Relais, eigene Darstellung nach [3].

Ähnlich zu den Steuerungskonzepten der Steuerbox erfolgt die Ansteuerung der Geräte im MicroGrid über eine digitale Schnittstelle. Allerdings ist die digitale Schnittstelle, welche in diesem Projekt genutzt werden soll, nicht beschränkt auf einen Standard, z.B.

EEBus oder ModBus, sondern umfasst sowohl ModBusTCP, generelle TCP-Verbindungen (mit den UFE-Wechselrichtern) als auch OCPP als Schnittstellen und ist modular erweiterbar. Dadurch ist eine höhere Rückwärtskompatibilität gewährleistet und es kann eine größere Bandbreite an Steuerbaren Lasten eingebunden werden. Eine Ansteuerung der steuerbaren Lasten über Relais ist in dieser Arbeit nicht vorgesehen.

2.1.4. Anforderungen an eine digitale Schnittstelle

Seit der Version 1.3 des Lastenhefts für die Steuerbox, wandelt diese die zu sendenden Daten in Formate für EEBUS um vgl. [3], da für die Anwendung einer Steuerbox ein reines Übertragungsprotokoll wie Message Queuing Telemetry Transport (MQTT) und auch Datenbeschreibungssprachen wie JavaScript Object Notation (JSON) alleine nicht ausreichen, um die Kommunikationsschnittstelle ausreichend zu beschreiben.

Zusammengefasst umfassen die Kriterien einer digitalen Schnittstelle für die Steuerbox ,vgl. [3]:

- Basiert auf normierten Standards für Kommunikation und Datenstrukturen mit einheitlicher Anwendungsschicht
- Die Verfügbarkeit, die Zugänglichkeit und Aktualität aller Protokoll-Spezifikationen
- Die aktive Einbindung von Experten zur Ausarbeitung von Detailspezifikationen der Anwendungsfälle
- Die Umsetzbarkeit der Anwendungsfälle, siehe Kapitel 2.1.4
- Eigene Sicherheitsstandards oder Kompatibilität mit etablierten Standards, vgl. [7]
- Interoperabilität und Möglichkeit zur Konformitätsprüfung

Um eine physische Verbindung (physical Layer) der steuerbaren Lasten mit der digitalen Schnittstelle der Steuerbox zu realisieren kann entweder kabelgebunden mithilfe eines Ethernet Kabels oder die kabellose Anbindung über WLAN eine Verbindung hergestellt werden. Um letzteres zu realisieren, muss allerdings eine IP-Basierte Kommunikation mittels IPv4 oder IPv6 vorhanden sein. Dies kann mithilfe eines Dynamic Host Configuration Protocol Servers (DHCP-Servers) oder Zero Configuration Networking automatisch geschehen. Auch eine statische Vergabe von IP-Adressen in diesem Kontext ist möglich, aber nicht angestrebt, vgl. [3].

Um einen Datenaustausch zwischen der Steuerbox und der zu steuernden Einrichtung zu gewährleisten, werden verbindungsorientierte Transportprotokolle wie das Transmission Control Protocol (TCP) genutzt werden. Dies erfüllt bereits die Anforderungen an die Reihenfolgetreue und Zuverlässigkeit. Sollten modernere Kommunikationsprotokolle genutzt werden so ist die vorgeschlagene Architektur so, dass die zu steuernde Einheit als „Server“ und die steuernde Einheit, in diesem Fall die Steuerbox als „Client“ auftritt. Bei der Nutzung eines moderneren Protokolls ist ebenfalls darauf zu achten, dass eine direkte Kommunikationsverbindung aufgebaut wird und kein Message-Broker notwendig ist, da dieser eine potentielle Fehlerquelle darstellt, welche die Verfügbarkeit der steuerbaren Einrichtung verringern kann, vgl. [3].

Nachdem im Kapitel 2.1.3 die unterschiedlichen Steuerfälle der Steuerbox beschrieben wurden, werden nun die Funktionen der Steuerbox, die aus den Anwendungsfällen

abgeleitet werden und für eine digitale Schnittstelle umgesetzt werden sollen im Detail dargelegt, siehe Tabelle 1.

Tabelle 1: Funktionen der digitalen Schnittstelle der Steuerbox, nach [3].

Funktion	Beschreibung
Übertragung der Wirkleistungslimitierung	Mithilfe dieser Funktion ist es möglich, einen Leistungsgrenzwert festzulegen und während der aktiven Verbindung an die steuerbare Einrichtung zu übermitteln.
Rückmeldung des aktuellen Leistungswerts	Mithilfe dieser Funktion ist es möglich, den aktuellen Leistungsgrenzwert sowie den aktuellen Leistungswert der Anlage auszulesen.
Zyklische Verbindungsüberwachung	Mithilfe dieser Funktion wird die verlässliche Steuerbarkeit der Einrichtung sichergestellt und einen Verbindungsausfall festzustellen und darauf mit der Failsafe-Funktion zu reagieren.
Failsafe-Funktion	Bei einem Ausfall der Kommunikationsverbindung zwischen der Steuerbox und der zu steuernden Einrichtung, wird mithilfe der Failsafe-Funktion ein vorher vorbestimmten, sichereren, Betriebszustand hergestellt.
Standardisiertes Datenmodell	Um eine hohe Interoperabilität zwischen der Steuerbox und den zu steuernden Einrichtungen zu gewährleisten, ist die Verwendung eines standardisierten Kommunikationsprotokolls unumgänglich und vereinfacht auch eine Inbetriebnahme einer Anlage.
Anlageninformationen und Status	Von der zu steuernden Einrichtung sollten zu diesem Zwecke folgende Parameter zur Verfügung gestellt werden: Störung der Anlage, Identifikation der Anlage, Herstellerbezeichnung, Produktbezeichnung, Seriennummer, Hardwarerevision und Software-Revision.

Im Rahmen der VDE-AR-E 2829-6-1, vgl. [8], wurde als Mindeststandard für die Umsetzung der digitalen Schnittstelle das EEBUS-Kommunikationsprotokoll festgelegt. Die Gründe für die Festlegung dieses Standards sind im Folgenden dargestellt:

- Das EEBUS-Kommunikationsprotokoll ist ein öffentlicher und kostenloser Standard
- Das EEBUS-Kommunikationsprotokoll ist bereits als Kommunikationsprotokoll zwischen Steuereinrichtungen und steuerbaren Einrichtungen in Laboren und Praxis erprobt.
- Durch den Austausch von SKI-Werten zur Sicherung der Kommunikationsverbindung wird die Inbetriebnahme vereinfacht
- Durch die Gruppierung von Funktionen in die passenden Anwendungsfälle kann ein Abgleich der Verfügbaren Funktionen erfolgen
- Es ist eine zunehmende Verfügbarkeit von EEBUS sowohl bei steuernden als auch bei steuerbaren Einrichtungen zu verzeichnen
- Es besteht eine Standardisierungspartnerschaft zwischen dem BSI und der EEBUS Initiative e.V.
- Das EEBUS-Protokoll wird weiterhin aktiv gepflegt und erweitert besonders im Hinblick auf eine Plug and Play Charakteristik des Standards

In Zukunft soll die Steuerbox nicht mehr als einzelnes physikalisches Gerät in das System eingebunden werden, sondern in das SMGW integriert werden, vgl. [3]. Hierbei wird vor allem auf die Schnittstelle IF_CLS_CTRL, vgl. [3], gesetzt werden, welche das Kommunikationsprotokoll EEBUS nutzen wird.

Die Steuerbox kann hierbei sowohl als separates physisches Gerät oder aber auch als Bestandteil des SMGW auftreten. Der Aufbau mit der Steuerbox als separates Gerät ist im Bild dargestellt und veranschaulicht außerdem die Verantwortungsbereiche des Messstellenbetreibers (MSB) und es Betreibers der Anlage.

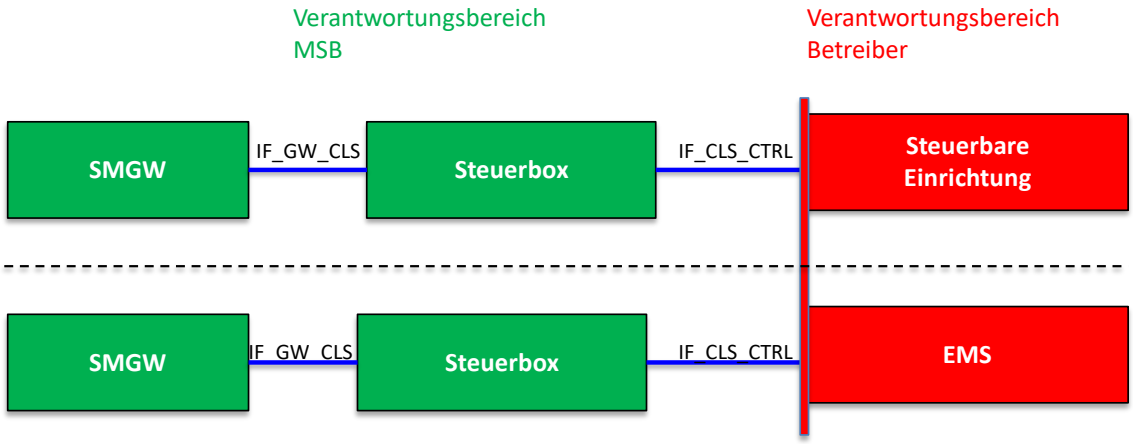

Abbildung 10: Verantwortungsbereiche Messstellenbetreiber und Anlagenbetreiber, eigene Darstellung nach [3].

Im Gesamtsystem der netzdienlichen Steuerung gemäß §14a EnWG übernimmt das Smart Meter Gateway (SMGW**)** eine zentrale Rolle. Es stellt gemäß BSI-Schutzprofil PP0073, vgl. [9], und der Technischen Richtlinie TR-03109-1, vgl. [10], den sogenannten

CLS-Kanal (Controllable Local System) bereit. Über diesen wird eine TLS-gesicherte Verbindung sowohl zum Backend-System des Netzbetreibers als auch zur Steuerbox aufgebaut, wobei sämtliche Steuerinformationen transparent durchgeleitet werden.
Die Anbindung von steuerbaren Geräten über das SMGW erfolgt über die Schnittstelle IF_GW_CLS. Diese CLS-Schnittstelle ist in den Vorgaben der TR-03109-5, vgl. [11], näher definiert und regelt den Zugang externer Steuerungseinrichtungen zu den angeschlossenen Geräten.
Die eigentliche Steuerbox arbeitet gemäß den Vorgaben des VDE-FNN-Lastenhefts sowie der TR-03109-5, vgl. [11]. Der vom SMGW initiierte CLS-Kanal besteht zwischen diesem und der Steuerbox, welche die empfangenen Steuerbefehle anschließend über die Schnittstelle IF_CLS_CTRL an die angeschlossenen Anlagen bzw. Endgeräte weiterleitet.
Die Schnittstelle IF_CLS_CTRL, vgl. [8], bildet somit die direkte digitale Verbindung zwischen Steuerbox und steuerbarer Verbrauchseinrichtung. Sie überträgt die Steuerbefehle des Netzbetreibers und stellt die technische Grundlage für die Kommunikation mit den Geräten dar.

Im Bild ist der Aufbau der Anlage mit der im SMGW integrierten Steuerbox zu sehen. Diese Konfiguration wird bereits durch die Festlegung der Bundesnetzagentur BK6-22-300, vgl. [12], betrachtet und soll zukünftig unter der Tenorziffer 2a in die BK6-22-330, [12], eingearbeitet werden.

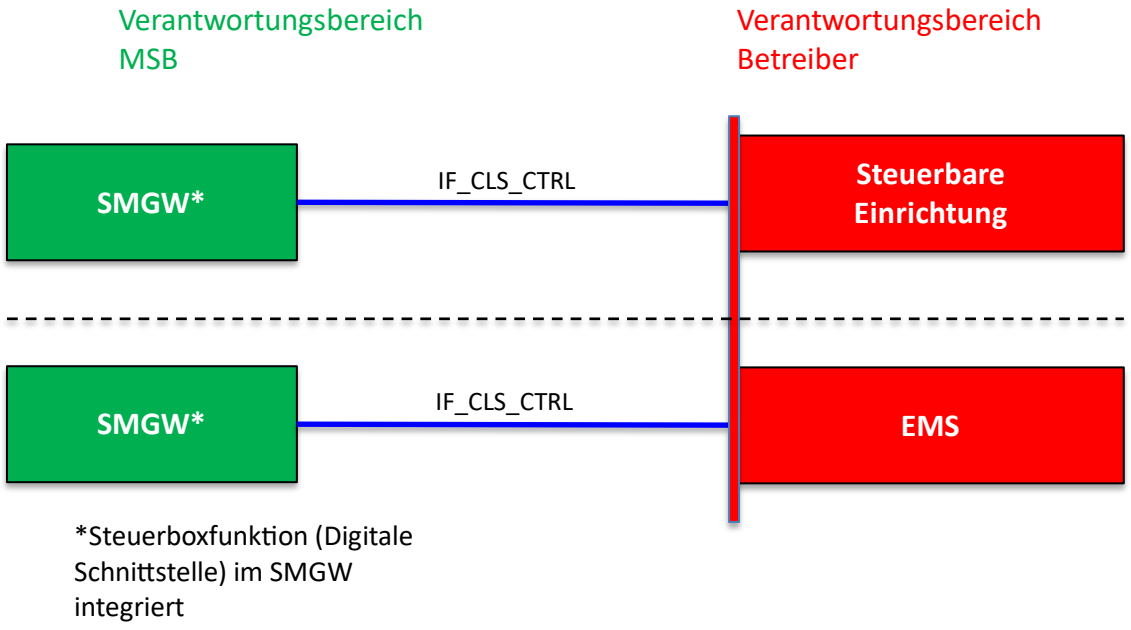

Abbildung 11: Verantwortungsbereich MSB und Betreiber der Anlage bei Integration der Steuerbox in das SMGW, eigene Darstellung nach [3].

Das SMGW wird dabei entsprechend dem BSI-Schutzprofil PP-0073, vgl. [9], sowie der Technischen Richtlinie TR-03109-1, vgl. [10] , die digitale Steuerungsschnittstelle bereitstellen. Neben der bereits existierenden CLS-Proxy-Funktionalität soll das Gateway künftig auch über eine physische Steuerungsschnittstelle, die IF_CLS_CTRL verfügen, die direkt mit steuerbaren Verbrauchseinrichtungen kommunizieren kann. Diese

Schnittstelle bildet die technische Grundlage für die Übertragung von Steuerbefehlen und erlaubt sowohl eine direkte Anbindung der Geräte als auch eine Integration von EMS, vgl. [3].

Die im VDE FNN Lastenheft definierte digitale Schnittstelle IF_CLS_CTRL dient der Kommunikation zwischen der Steuerbox und der steuerbaren Verbrauchseinrichtung, vgl. [3]. Über diese Schnittstelle können verschiedene Steuerbefehle übertragen und technische Informationen ausgetauscht werden. Die zentralen Anwendungsfälle umfassen:

1. Begrenzung der Wirkleistung im Bezug („Limitation of Active Power Consumption“, LPC)
2. Begrenzung der Wirkleistung in der Einspeisung („Limitation of Active Power Production“, LPP)
3. Überwachung des Leistungsbezugs („Monitoring of Power Consumption“, MPC)
4. Überwachung des Netzanschlusspunktes („Monitoring of Grid Connection Point“, MGCP)
5. Bereitstellung von Anlagendaten
6. Übertragung eines Heartbeats

Für die Anwendungsfälle 1 und 2 ist vorgesehen, dass die Steuerbefehle zu festen oder zyklischen Zeitpunkten übermittelt werden. Diese können im Voraus in der Steuerbox gespeichert und zum gewünschten Zeitpunkt unabhängig von einer aktiven Kommunikationsverbindung mit dem Verteilnetzbetreiber ausgeführt werden. Dieses Prinzip ermöglicht eine hohe Robustheit gegenüber Kommunikationsausfällen und eine höhere Skalierbarkeit durch reduzierte Anforderungen an die Kommunikationsinfrastruktur. Die Anwendungsfallbezeichnungen sind die Bezeichnungen, welche aus der VDE-AR-E 2829-6-1, vgl. [3], hervorgehen und im BK6-22-300, vgl. [12] aufgenommen werden.
Neben der Wirkleistungssteuerung definiert das Lastenheft der Steuerbox weitere technische Funktionen, insbesondere:

- Verbindungsüberwachung: Die Steuerbox prüft kontinuierlich, ob eine gültige Verbindung zur steuerbaren Einrichtung besteht. Im Fehlerfall wird ein Failsafe-Mechanismus aktiviert.
- Failsafe-Verhalten: Bei Kommunikationsstörungen wird automatisch ein zuvor übermittelter Grenzwert für eine definierte Zeit eingesetzt, um den Netzschutz zu gewährleisten.

Alle technischen Funktionen des Gespanns Steuerbox/SMGW laut der VDE-AR-E 2829-6-1, vgl. [3], sind Abbildung 12 dargestellt.

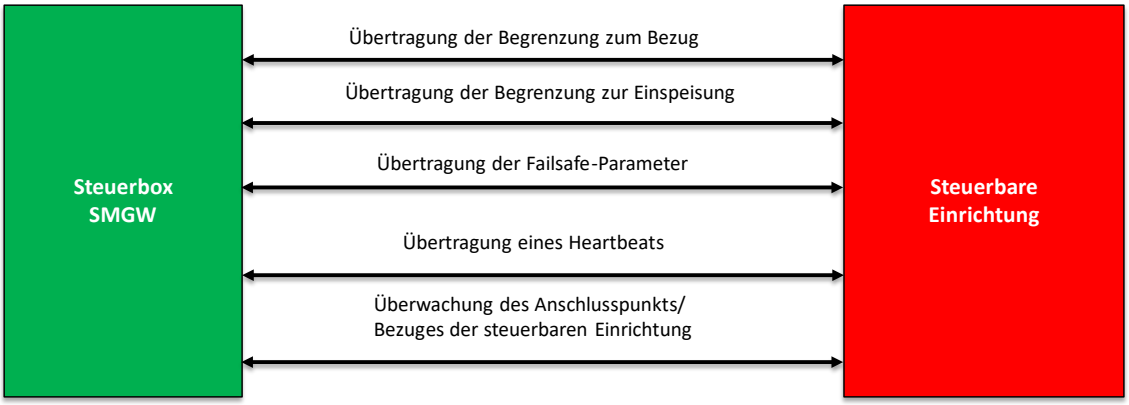

Abbildung 12: Zentrale Funktionen der Steuerbox, eigene Darstellung nach [3]

Zur Umsetzung der beschriebenen Steuer- und Überwachungsfunktionen nutzt die Steuerbox die EEBus-Kommunikationsarchitektur, vgl. [13], als verbindlichen Mindeststandard. Die Kommunikation erfolgt über die folgenden Schichten:

- Physikalisch: Ethernet (ETH)
- Netzwerk: IPv4/IPv6
- Transport: TCP bzw. TLS
- Applikation: Smart Home IP (SHIP, vgl. [14]) & Smart Premises Interoperable Neutral-Message Exchange(SPINE, vgl. [15]), über WebSockets und Domain Name System Sevice Discovery (DNS-SD)/ multicast Domain Name System (mDNS) zur Adressierung

Dargestellt sind die verschiedenen Layers der Kommunikation in der Abbildung 13, welche den Protokollstapel des EEBUS Protokolls darstellt.

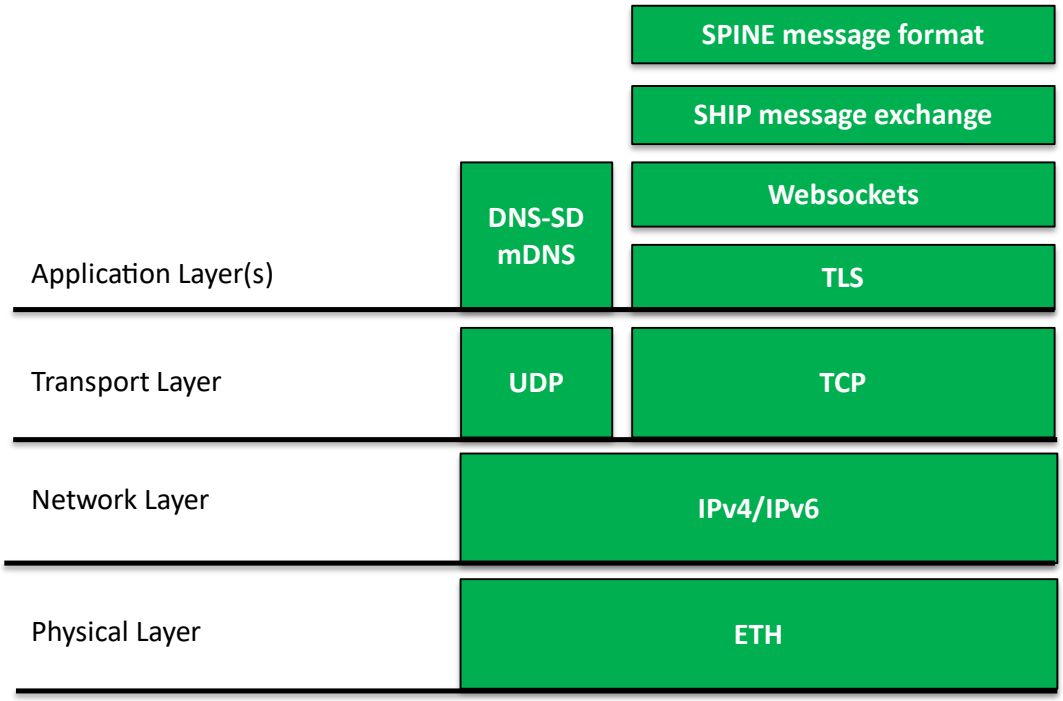

Abbildung 13: Application Layer des EEBUS Protokolls, eigene Darstellung nach [3]

Im Network Layer erfolgt die Anbindung der steuerbaren Einrichtung mithilfe von IPv4 und IPv6, somit ist die Anbindung sowohl statischer als auch dynamischer IP-Adressen gewährleistet. Um das gegenseitige Auffinden der EEBUS-Komponenten zu ermöglichen, kommt multicast Domain Name System (mDNS RFC 6762, vgl. [16]) und Domain Name System Sevice Discovery (DNS-SD RFC6763, vgl. [16]) zum Einsatz. Hierbei ist es so, dass jede EEBUS-Komponente eine Beschreibung des sogenannten SHIP-Dienstes mittels DNS-SD veröffentlicht.
Der eigentliche Datenaustausch erfolgt über das Transportprotokoll TCP in Verbindung mit einer TLS-Verschlüsselung, wie in VDE-AR-E 2829-6 Teil 4 spezifiziert. Als Anwendungsschicht kommt das WebSockets-Protokoll (RFC 6455 vgl. [16]) zum Einsatz. Die Nachrichten zwischen den Geräten werden dabei im JSON-Format übertragen. Teil 4 der genannten VDE-Norm enthält zudem genaue Anforderungen an die eingesetzten X.509-Zertifikate Quelle sowie an deren automatische Aktualisierung im laufenden Betrieb, um eine durchgängig gesicherte Kommunikation zu gewährleisten. Jede Anwendungsfunktion wird in EEBUS über sogenannte Anwendungsfälle umgesetzt. Für die Grenzwertübertragung werden z. B. die Anwendungsfälle LPC und LPP genutzt, während MPC und MGCP-Messdaten zur Überwachung übertragen. Zusätzlich enthalten die Nachrichtenstrukturen Mechanismen zur Überprüfung der Zustellung (Heartbeat) und zur Auslösung von Failsafe-Szenarien im Fehlerfall, vgl. [3].

Die übermittelten Inhalte basieren auf dem sogenannten SPINE-Datenmodell (Smart Premises Interoperable Neutral-Message Exchange), wie es in VDE-AR-E 2829-6 Teil 3, vgl. [3], spezifiziert ist. Dieses Modell strukturiert die Datenübertragung zwischen Steuerungseinrichtung und Gerät in sogenannte Use Cases, die jeweils bestimmten Steuer- oder Überwachungsfunktionen zugeordnet sind.

2.1.5. Forschungsprojekte zur SteuerBox nach FNN

Ein Beitrag zur praktischen Erprobung der Steuerbox gemäß §14a EnWG erfolgte im Rahmen des Forschungsprojekts KEMAL (Kundenorientiertes Energiemanagement mit autonomer Lastregelung). Ziel des Projekts ist es, ein innovatives Energiemanagementsystem (EMS) zu entwickeln, welches einerseits netzdienliche Steuerungsmaßnahmen nach §14a EnWG, vgl. [4], ermöglichen und andererseits Kundenanforderungen berücksichtigen.
Der Fokus liegt hierbei auf der Umsetzung des Mehrwertprodukts „Priorisiertes Laden“, mit dem Elektrofahrzeugnutzer sich in ereignisgesteuerte Tarife (z. B. TAF-5) einbuchen können, um bei netzseitigen Eingriffen bevorzugt behandelt zu werden. Hierbei liegt der Fokus auf dem Zusammenspiel von Netzbetreibern und den Nutzern welche über das SMGW als externe Marktteilnehmer fungieren, innerhalb der sogenannten gelben Ampelphase nach BDEW, vgl. [6], [17]. Die Kommunikation zwischen Netzbetreiber, EMS und externer Marktteilnehmer erfolgt hierbei über das SMGW als zentrale Kommunikationseinheit.

Im Teilvorhaben der Hochschule Biberach wird ein entsprechender Demonstrator entwickelt und im E-Mobility-Reallabor unter Realbedingungen getestet. Dabei wird unter anderem die Funktionalität zur Umsetzung des Energiewirtschaftlichen Anwendungsfalls 1 (EAF-1) die Steuerung nach §14a EnWG, vgl [4], des Energiewirtschaftlichen Anwendungsfalls 14 (EAF-14), die Bereitstellung von Daten für Energiemonitoring und Mehrwertdienste Elektrizität, und Energiewirtschaftlicher Anwendungsfall 16 (EAF-16) die separate Erfassung von Ladeverbräuchen integriert, vgl. [6]. Die technische Grundlage bildet die Steuerbox in Kombination mit dem im Projekt entwickelten SMGW der Firma EMH metering GmbH & Co KG und einem darauf abgestimmten EMS welches am elenia Institut für Hochspannungstechnik und Energiesysteme der Technischen Universität Braunschweig entwickelt wurde, vgl. [18]. Die Kommunikation erfolgt über die WAN-Anbindung mittels 450-MHz-LTE-Technologie, um eine hohe Ausfallsicherheit zu gewährleisten, vgl. [18].
Das Projekt KEMAL zeigt exemplarisch, wie sich standardisierte Steuerungstechnologie z. B. die Schnittstelle IF_CLS_CTRL mit kundenorientierten Mehrwertdiensten kombinieren lässt. Das Projekt liefert damit Erkenntnisse für die praktische Umsetzbarkeit der zentralen Steuerung über die Steuerbox unter Berücksichtigung regulatorischer Vorgaben und Nutzerakzeptanz. Der abschließende Bericht steht allerdings noch aus.

2.1.6. Relevanz der Steuerbox für den Aufbaus des Demonstrators

Im Rahmen des Demonstratorsystems wird bewusst auf den Einsatz einer klassischen Steuerbox nach FNN, vgl. [3], verzichtet. Stattdessen kommt ein dezentraler Ansatz zum Einsatz, bei dem ein Raspberry Pi als Ersatz für die Steuerbox fungiert. Dieser agiert, unabhängig von der vom Netzbetreiber zentral geführten Infrastruktur, wie sie im §14a EnWG [4] vorgesehen ist, in einem Schwarmnetz mit anderen Raspberry Pis. Dieser dezentrale Ansatz orientiert sich jedoch am Konzept der Steuerbox wobei jedoch ein von der Infrastruktur der Steuerbox und SMGW unabhängiges System entwickelt werden soll. Die Anwendungsfälle sind aber vergleichbar.

Die Entscheidung für diese Vorgehensweise wurde aus mehreren Gründen getroffen:

1. Unabhängigkeit vom Smart-Meter-Gateway: Die Steuerbox ist auf die Anbindung an ein zertifiziertes Smart-Meter-Gateway angewiesen, das insbesondere in privaten Haushalten noch nicht flächendeckend verfügbar ist. Der Raspberry Pi benötigt diese Anbindung nicht und kann damit auch jetzt schon in Bestandshaushalte eingebaut werden. Hinzu kommen die hohen Kosten für SMGW.
2. Dezentraler Optimierungsansatz: Während die Steuerung der Geräte bei der Steuerbox zentral vom Netzbetreiber aus erfolgt, verfolgt der Demonstrator in dieser Arbeit eine dezentrale Regelung. Mehrere Raspberry Pis bilden einen Schwarm, in welchem die lokalen Messwerte, Spannung und Strom, der Knoten austauscht werden und basierend auf einem gemeinsamen Optimierungsalgorithmus Steuerentscheidungen getroffen werden. Hierbei stellt jeder Raspberry Pi einen Haushalt dar. Ziel ist es, netzorientiert auf Lastspitzen zu reagieren und gegebenenfalls Ladevorgänge zu reduzieren während trotzdem die Kundenwünsche eingehalten werden. So können Lastspitzen gezielt vermieden werden, ohne dass der Netzbetreiber eingreifen muss.
3. Kompatibilität mit älteren Geräten: Die Steuerbox setzt mit der digitalen Schnittstelle auf den Kommunikationsstandard EEBUS, der zukunftsorientiert ist, jedoch Bestandsgeräte welche ältere Kommunikationsstandards verwenden nicht miteinschließt. Für den Demonstrator werden etablierte Schnittstellen wie Modbus TCP und OCPP genutzt, wodurch auch Geräte mit diesen Standards eingebunden werden können.
4. Minimalanforderung für Systemeinbindung: Für einen funktionalen Knoten im Demonstratornetzwerk reicht ein einziger Raspberry Pi in Kombination mit einer steuerbaren Wallbox oder einem Wechselrichter, solange von diesen Messwerte abgefragt werden können, aus. Zusätzlich wurde eine Messplatine entwickelt, welche an den Raspberry Pi angeschlossen werden kann, um die Strom- und Spannungswerte am Hausanschluss aufzunehmen.

Mit diesem geplanten Aufbau, wird exemplarisch ein System umgesetzt, was die Steuerung der steuerbaren Lasten als Reaktion auf eine steigende Netzauslastung ermöglicht.

Für diese Arbeit wird ein Raspberry Pi pro Knoten verwendet, wobei auf die bereits im MicroGrid der TH Köln verbauten Messgeräte und Wechselrichter als Informationsquellen und ansteuerbare Geräte zurückgegriffen wird. Die Wechselrichter sollen hierbei Ladevorgänge von Elektrofahrzeugen imitieren und bei Bedarf gedrosselt werden.

2.2. Experimente und Versuche der Netzbetreiber

Auch die Netzbetreiber haben ein Interesse daran, die Lastschwankungen im Niederspannungsnetz so gering wie möglich zu halten und die Lastspitzen vorhersehbar beziehungsweise geglättet. Ein Ansatz wie das erreicht werden kann, ist zum Beispiel ein zentrale gesteuerter Ladepark oder eine indirekte Steuerung durch Preisanreize, um die Lastspitzen, welche durch Elektromobilität entstehen zu bündeln und steuern zu können [19], [20], [21]. Hierzu wurde von den Netzen BW Projekte und Versuchsaufbauten durchgeführt, um zentrale Lademanagementkonzepte zu testen. Die Projekte werden im Folgenden mit ihren jeweiligen Ergebnissen kurz dargestellt.

2.2.1. Gesteuerte Laden von Elektrofahrzeugen über Preisanzeige

Ein Forschungsprojekt, was sich mit der Frage beschäftigt, wie Elektromobilität in das Niederspannungsnetz integriert werden kann ist das „Gesteuerte Laden von Elektrofahrzeugen über Preisanzeige“ vom TÜV Rheinland, dem Fraunhofer ISI und dem VDI/VDE aus dem Jahr 2020. Diese Kurzstudie untersuchte besonders die Rollen der Ampelphasen nach dem BDEW, vgl. [17], [22], die Nutzung von SMGW und die Nutzung von OCPP als Kommunikationsstandard mit den Ladepunkten, vgl. [21].

Betrachtet wurden hierbei besonders das Projekt „Laden am Arbeitsplatz“ (LamA), welches das OCPP 1.6 nutze um die Kommunikation zwischen dem zentralen Backend, für eine ganzheitliche Optimierung, und den Ladestationen umzusetzen, vgl. [21]. Die Nutzung des OCPP stellte sich hierbei als zentraler und zuverlässiger Bestandteil der Umsetzung heraus, vgl. [21].

Auch im Forschungsprojekt „ELBE“ wurde das OCPP erfolgreich genutzt, um Soll- und Ist-Leistungswerte zwischen dem zentralen Backend und den Ladestationen auszutauschen, um so eine netzdienliche Steuerung umzusetzen.

Die Studie stellt somit das OCPP als Schlüsseltechnologie für die Umsetzung von Lademanagementkonzepten heraus, vgl. [21]. Daher wird auch in dieser Arbeit auf das OCPP zur Ansteuerung der Wallboxen zurückgegriffen werden.

2.2.2. Netzlabor E-Mobility Allee

Im Forschungsprojekt E-Mobility-Allee der Netze BW untersuchte in einem Wohnquartier mit hoher Durchdringung an Ladepunkten die Auswirkungen der Elektromobilität auf das Niederspannungsnetz, sowie verschiedene Maßnahmen zur Stabilisierung der

Spannung und des Stroms. Im Fokus standen sowohl zentral gesteuerte Ladevorgänge als auch verhaltensbasierte Steuerung der Ladevorgänge.

In diesem Projekt wurden vier verschiedene Arten des Lademanagements bzw. der Laststeuerung getestet. Der erste Ansatz war in diesem Fall das freie Laden, bei welchem die Testhaushalte ohne Einschränkung mit bis zu 22 kW laden konnten, was bei den 10 teilnehmenden Haushalten zu einer theoretischen Auslastung von 174 % führen würde. Real gemessen war der höchste Wert jedoch bei 56 % erreicht, vgl. [20].
Der nächste Ansatz, welcher in diesem Projekt untersucht wurde, war ein präventives Lademanagement, welches in zwei Unteransätze aufgeteilt wurde. Der erste Ansatz waren sogenannte Freigabengruppen, bei denen die 10 Ladepunkte in zwei Gruppen aufgeteilt wurden und im 15 Minuten Takt zwischen voller Leistung, also 22 kW und gedrosselter Leistung, also 5,5 kW wechselten. Dies führte zu einer Verlängerung der Ladedauer von ca. 60 Minuten, wobei sich sowohl die theoretische Netzbelastung von 174 % auf 126 % als auch die reale Netzbelastung von 56 % auf 31 % senkten, vgl. [20]. Der zweite Ansatz war eine Freigabequote, bei dem zu jedem Zeitpunkt eine vorab festgelegte Gesamtleistung gleichmäßig auf alle Ladevorgänge aufgeteilt wurde. Dieser Ansatz erzielte ähnliche Ergebnisse wie der Ansatz der Freigabegruppen, vgl. [20].

Als nächstes wurden zwei verschiedene Arten des reaktiven Lademanagements getestet, ein stromgeführtes Lademanagement und ein Spannungsgeführtes Lademanagement. Beim ersten Lademanagement wurde der Strom in der versorgenden Ortsnetzstation herangezogen, um zu bestimmen, ob der Netzstrang überlastet ist. Wird ein gewisser Grenzwert überschritten, werden alle Ladevorgänge um 30 % gedrosselt, bis der Grenzwert wieder erreicht wird. Hierbei stellte sich eine Ladezeitverlängerung von ca. 40 Minuten ein, wobei mit einer Worst-Case Belastung des Stromnetzes von 73 % gerechnet wurde, welche zu einer realen Netzbelastung von 36 % führte, vgl. [20]. Dieses Konzept führte zu einer optimalen Auslastung des Stromnetzes, brachte allerdings auch eine hohe Komplexität bei der Umsetzung der Überwachung mit sich. Besonders die Kommunikation der geeigneten Netzzustandsdaten stellte hier eine Herausforderung dar, vgl. [20].

Beim Spannungsgeführten Lademanagement überwachten die Ladestationen die Spannung des Stromnetzes am jeweiligen Knoten und regelten den Ladevorgang herunter, hierbei lag die maximale Ladeleistung wieder bei 220 kW und einer Netzauslastung von 174 %. Real stellte sich jedoch eine Netzbelastung von 37% ein. Die Netzspannung ist jedoch kein geeignetes Mittel um die Ladevorgänge zu steuern, da sie nicht immer die Auslastung des Netzes abbildet, vgl. [20].
Zusätzlich zu den Lademanagementkonzepten wurden auch dezentrale und zentrale Batteriespeicher im Netz installiert, um das Netz zu stabilisieren, vgl. [20].
Die Studie zeigt jedoch, dass sich das stromgeführte Lademanagement als besonders geeignet herausstellte, um das Netz optimal auszulasten, was sich auch im Forschungsprojekt GridMaximizer widerspiegelt, siehe Kapitel 2.4.

2.2.3. Netzlabor E-Mobility Carré

Im Netzlabor E-Mobility Carré wurden den Studienteilnehmenden 45 E-Fahrzeuge übergeben, welche für den Testzeitraum genutzt werden sollten, vgl. [19]. Zunächst konnten die Nutzer ungeregelt Laden, um das Ladeverhalten zu erfassen. Hierbei stellte sich heraus, dass bei 58 verfügbaren Ladepunkten nie mehr als 13 Fahrzeuge gleichzeitig geladen wurden. Der Mittelwert der gleichzeitig ladenden Fahrzeuge lag hierbei bei 5, was jedoch nur zu 7,6 % der Zeit erreicht wurde. Rund 42 % der Zeit wurde kein Fahrzeug geladen, 31,5 % der Zeit lud nur ein Fahrzeug und 12,4 % der Zeit luden zwei Fahrzeuge parallel, vgl. [19]. Auch in diesem Projekt wurden Lademanagementkonzepte erprobt, welche zu einem ähnlichen Ergebnis führten wie bei Projekt der E-Mobility-Allee, wobei in diesem Abschlussbericht keine genauen Werte angegeben wurden wie bei der E-Mobility-Allee, vgl. [19], [20].
Interessant stellt sich jedoch der Gleichzeitigkeitsfaktor, vgl. Kapitel 2.5, der Ladevorgänge dar, welcher sich trotz der höheren Anzahl an Elektrofahrzeugen als relativ gering herausstellte.

2.3. Zellulare Netze

Das Forschungsprojekt C/sells war Teil des Förderprogramms „Schaufenster intelligente Energie – Digitale Agenda für die Energiewende“ (SINTEG) des Bundesministeriums für Wirtschaft und Energie. Das Ziel des Projekts war es, skalierbare Lösungen für eine nachhaltige Energieversorgung zu entwickeln und zu demonstrieren. Im Fokus stand dabei die Umsetzung eines zellulären Energiesystems mit hohen Anteilen erneuerbarer Energien, insbesondere Photovoltaik, in den Bundesländern Baden-Württemberg, Bayern und Hessen [23].
Eine zentrale Komponente von C/sells war die Entwicklung und Erprobung einer Abstimmungskaskade zwischen Verbrauchern und Netzbetreibern. Diese sollte eine effiziente und sichere Koordination der verschiedenen Teilnehmer im Energiesystem ermöglichen. Dabei wurden intelligente Messsysteme (Smart Meter) und Controllable Local Systems (CLS) eingesetzt, um eine bidirektionale Kommunikation zu realisieren [23].
Ein weiterer Schwerpunkt des Projekts lag auf dem Basisinstrument des C/sells, dem regionalisierten Handel mit Energie und Flexibilitäten. Durch die Integration von Prosumern, also Verbrauchern, welche auch Energie erzeugen und ins Netz einspeisen, sollte ein dezentraler Energiemarkt geschaffen werden. Dies ermöglichte es, lokale Flexibilitätspotenziale zu nutzen und die Netzstabilität zu erhöhen [23].
Die Ergebnisse des Forschungsprojektes C/sells zeigen, dass ein zelluläres Energiesystem mit einer effektiven Abstimmungskaskade und einem regionalisierten Energiemarkt einen Beitrag zur erfolgreichen Umsetzung der Energiewende leisten kann. Die im Projekt entwickelten Konzepte und Technologien bieten Erkenntnisse für zukünftige Anwendungen in intelligenten Energiesystemen.

Das in dieser Arbeit entwickelte Steuerungskonzept steht dem zellularen Ansatz des Projekts C/sells nahe. Beide Konzepte beruhen auf einer Dezentralisierung der

Netzsteuerung und einem hohen Maß an Autonomie auf Seiten der Netzteilnehmer. Während C/sells den Fokus auf die Markintegration der Prosumer und Teilnehmer-Kommunikation legt, geht das in dieser Arbeit angepeilte Swarm-Grid-Modell noch einen Schritt weiter. Die Intelligenz der Netzführung liegt vollständig bei den einzelnen Geräten, die anhand lokaler Messdaten und Kommunikation mit den anderen Geräten aus dem Netzstrang mithilfe einer Optimierung selbstständig über ihr Verhalten entscheiden. Dadurch entsteht eine „Schwarmintelligenz“, welche flexibel auf die lokale Netzsituation reagieren kann, ohne dass eine zentrale Koordination durch den Netzbetreiber erforderlich ist.

2.4. GridMaximizer

Im Rahmen des Projektes GridMaximizer der TH Köln wird eine Technologie entwickelt, welches das Problem der immer weiter steigenden Elektrifizierung des Energiebedarfs angehen soll. Der Lösungsansatz verspricht mithilfe dezentral gesammelter Messwerte z.B. durch SMGW oder Wallboxen den Netzzustand nicht nur abzuschätzen, sondern konkret zu bestimmen, um die steuerbaren Lasten im Netz netzdienlich zu steuern, vgl. [24]. Durch die Kenntnisse über den genauen Zustand des Netzes, besonders der bezogenen Ströme an jedem Netzknoten, kann dieses deutlich besser ausgelastet werden und es können mehr elektrische Lasten, wie zum Beispiel Ladepunkte für Elektrofahrzeuge oder Wärmepumpen in das Netz integriert werden [20]. Hierzu muss die Kommunikation der Geräte untereinander und die Kommunikation der Geräte mit den Steuerbaren Lasten im MicroGrid realisiert werden. Diese Arbeit stellt mit dem Aufbau eines Demonstrators zur dezentralen Regelung von steuerbaren Lasten im Netz einen Teil dieses Forschungsprojektes dar.

2.5. Gleichzeitigkeitsfaktor

Der Gleichzeitigkeitsfaktor g gibt an, in welchem Maße einzelne Lasten gleichzeitig Spitzenbelastungen erzeugen, und spielt eine zentrale Rolle bei der Auslegung von Niederspannungsnetzen. Definiert ist er als das Verhältnis der mittleren Spitzenleistung (P_{Avg}) gegenüber der Spitzenleistung pro Haushalt (P_{max}), vgl. [25]:

$$g = \frac{P_{Avg}}{P_{max}}$$

Ein kleinerer Gleichzeitigkeitsfaktor bedeutet, dass die Einzelspitzen seltener zusammentreffen und die Netzauslastung in der Realität geringer ausfällt, als es eine bei einer reinen Addition der möglichen Lasten der Fall wäre. Hierbei ist es so, dass der Gleichzeitigkeitsfaktor in einem Netz mit nur einem Knoten natürlich 1 entspricht und bei Netzen mit deutlich mehr Knoten immer weiter zurück geht. Bei einer Netzgröße von 150 Anschlusspunkten liegt er beispielsweise nur noch bei 0,17, vgl. [26].

Im Kontext moderner, volatiler Lasten, insbesondere der Elektromobilität, kann es jedoch sein, dass sich dieser Faktor verändert bzw. erhöht. Da Ladevorgänge bei Elektrofahrzeugen häufig in den Abendstunden stattfinden, kann der Gleichzeitigkeitsfaktor steigen. In

der Praxis hat sich jedoch gezeigt, dass dies nicht notwendigerweise immer der Fall ist, vgl. Kapitel 2.2.2. Trotzdem ist der Gleichzeitigkeitsfaktor besonders für kleine und schlecht ausgebaute Netze ein wichtiger Faktor. [19]

2.6. Bedeutung für diese Arbeit

Die meisten der vorab genannten Arbeiten und Forschungen beziehen sich auf einen zentralisierten Ansatz, welcher vom Netzbetreiber gesteuert werden muss. Bisher gibt es nur wenige Forschungsprojekte zur dezentralen Regelung von steuerbaren Lasten in Niederspannungsnetzen und auch keine Alternativen zur Nutzung der Steuerbox in Verbindung mit einem SMGW. Daher untersucht diese Arbeit im Forschungsprojekt GridMaximizer einen Ansatz, der sich durch die Dezentralität des Systems und die Möglichkeit auszeichnet, Bestandsladeplätze ohne SMGW zu integrieren und zu steuern. Hierbei liegt der Fokus besonders auf den älteren bzw. gängigeren Schnittstellen ModBusTCP und OCPP und der Verwendung von einfacher und kostengünstiger Hardware in Kombination mit einer Optimierung des Netzstranges und perspektivisch auch einer Netzzustandsschätzung zum besseren Abfangen von Lastspitzen. Besonders die Netzzustandsschätzung, vgl. [27] wird wichtig, wenn die Anzahl der Knoten an denen die Messwerte wie Strom und Spannung bekannt sind abnimmt und sich das Netzmodell nicht mehr einfach optimieren lässt.

3. Hintergrund

Ein alternatives Konzept gegenüber der zentralen Steuerung über Steuerboxen stellt das sogenannte Swarm Grid dar, das im Rahmen des Forschungsprojekts GridMaximizer weiterentwickelt wurde, vgl. [28]. Das Swarm-Grid-Prinzip basiert auf der Idee einer dezentralen Intelligenz im Stromnetz: Jedes steuerbare Gerät, wie etwa Wallboxen, Wärmepumpen oder Wechselrichter für Batteriespeicher, agiert als eigener „Grid Agent“, der auf Basis lokaler Messwerte und Kommunikationsdaten mit benachbarten Netzkomponenten Entscheidungen trifft. Das Ziel ist es, ohne zentrale Instanz ein stabiles und effizientes Netzverhalten zu ermöglichen. Besonders wichtig ist dies in kritischen Situationen wie der sogenannten „roten Ampelphase“ im Ampelkonzept des BDEW [22].
Die Geräte im Swarm Grid erfassen kontinuierlich relevanten elektrischen Größen wie Spannung, Stromstärke und ggf. Phasenwinkel am Netzknoten. Durch den Austausch dieser Daten über ein dezentrales Kommunikationsnetzwerk kann eine lokale Netzzustandsschätzung (State Estimation) durchgeführt werden. Auf dieser Basis optimieren die Geräte ihr Verhalten, z.B. die bezogene Ladeleistung selbstständig, vgl. [27]. Ergänzend lassen sich auch Netztopologien rekonstruieren, was insbesondere bei dynamischen Änderungen im Netz wie durch das Zu- oder Abschalten von Wallboxen relevant ist, vgl. [29].
Für die Umsetzung dieses Konzepts muss eine Kommunikation zwischen den einzelnen Netzknoten gewährleistet werden

3.1. Technologische Grundlagen der Systemarchitektur

Um die Zielsetzung und das Systemdesign des Demonstrators einordnen zu können, werden in diesem Kapitel die wesentlichen Technologien und Rahmenbedingungen vorgestellt, die der Arbeit zugrunde liegen.

3.1.1. Steuerbare Lasten und §14a EnWG

Wie bereits in Kapitel 2.1.1 erwähnt werden Wallboxen, Wärmepumpen und Wechselrichter für Batteriespeicher ab einer Leistung von 4,2 kW als steuerbare Verbrauchseinrichtungen (SteuVE) angesehen [4]. Der §14a des Energiewirtschaftsgesetzes, [4], verpflichtet Netzbetreiber dazu, diese Geräte in ein netzorientiertes Lastmanagement einzubinden. Voraussetzung hierfür ist eine technische Möglichkeit zur Fernsteuerung und zur Rückmeldung des Gerätezustands [3]. Diese Fähigkeit wird heute entweder durch SMGW in Kombination mit Steuerboxen mit Relais, oder digitaler Schnittstelle gewährleistet, vgl. Kapitel 2.1.1. Allerdings ist der Ausbau bzw. die Durchdringung des deutschen Niederspannungsnetzes durch SMGW noch nicht weit fortgeschritten, um dies in den nächsten Jahren flächendeckend zu gewährleisten. Die Bundesnetzagentur spricht 2023 noch von etwa einer halben Millionen Smart Metern. [30]

3.1.2. Kommunikationsprotokolle zur Gerätesteuerung

Neben dem in Kapitel 2.1.2 erwähnten Protokoll EEBUS, gibt es noch weitere, bereits etablierte Protokolle zur Einbindung und Ansteuerung von steuerbaren Lasten. Diese sind das Protokoll Modbus TCP und das in Kapitel 2.2 erwähnte Protokoll OCPP. Beide Protokolle sind geeignet, um steuerbare Geräte in ein System zur Steuerung einzubinden. Modbus TCP eignet sich hierbei für eine Vielzahl verschiedener Anlagen, welche von Wechselrichtern über Ladeboxen bis zu Wärmepumpen reichen können. OCPP eignet sich zur Einbindung modernerer Ladeboxen, vgl. [21].

Modbus TCP ist ein weit verbreitetes, etabliertes Kommunikationsprotokoll. Es basiert auf einem einfachen Server-Client-Prinzip. Hierbei kann der Client, in diesem Fall der Raspberry Pi, Steuerbefehle an untergeordnete Server, in diesem Fall die Wechselrichter und Messgeräte, senden [31]. Diese Geräte, verfügen über definierte Register, in denen Soll- oder Ist-Werte gesetzt und abgerufen werden können. Beispielsweise können Ladeleistungen als Sollwerte gesetzt werden, oder auch aktuelle Zustände, Spannungen oder Ströme als Messwerte abgefragt werden.

Die Kommunikation erfolgt über das TCP/IP-Protokoll auf Port 502, was die Integration in bestehende Netzwerke erleichtert. Ein Vorteil von Modbus TCP liegt darin, dass viele bestehende Anlagen es nativ unterstützen. Damit eignet sich Modbus TCP besonders gut zur Anbindung älterer oder kostengünstiger Geräte. Hierbei stellen sich die Protokoll Layer wie in Abbildung 14 dar.

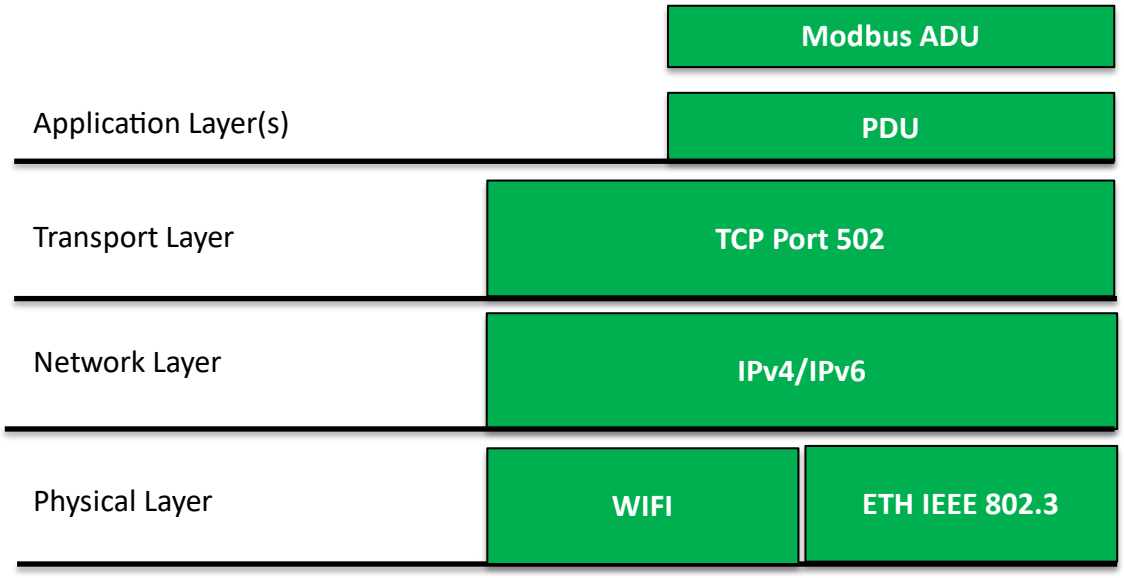

Abbildung 14: Protocol layers von ModbusTCP nach [31]

Modbus definiert in der Anwendungsschicht einen sogenannten Header, welcher die Grundinformationen der Nachricht umfasst, wie zum Beispiel Transaktions ID, Protokoll ID und Länge. Dies geschieht in der Application Data Unit (ADU). In der Protocol Data Unit (PDU) hingegen werden dann der Funktionscode und die Daten der Nachricht übertragen. Auf Netzwerkebene Arbeitet ModbusTCP mit IPv4 und IPv6 [31]. Als physical

layer unterstützt ModbusTCP nativ Ethernet IEEE 802.3, vgl. [31], kann aber auch über WIFI betrieben werden, vorausgesetzt die zu steuernden Geräte verfügen über ein entsprechendes Modul, vgl. [31].

Open Charge Point Protocol (OCPP) hingegen ist ein speziell für die Elektromobilität entwickelter Kommunikationsstandard, der die Interaktion zwischen Ladepunkten und einem Backend regelt. In dieser Arbeit wird die Version OCPP 1.6 verwendet, welche sich als stabil und gut dokumentiert etabliert hat. OCPP erlaubt die Steuerung und Überwachung von Ladevorgängen. Die zentralen Funktionen des Protokolls umfassen das Setzen und Verändern von Ladeleistungen, das Starten oder Stoppen von Ladesessions, das Abfragen von Statusmeldungen sowie das Auslesen von Messwerten wie Strom, Spannung oder Energieverbrauch. Die für diese Arbeit relevanten Funktionen sind in Abbildung 15 dargestellt und werden im OCPP-Server in Python umgesetzt, siehe Kapitel 4.5.3.

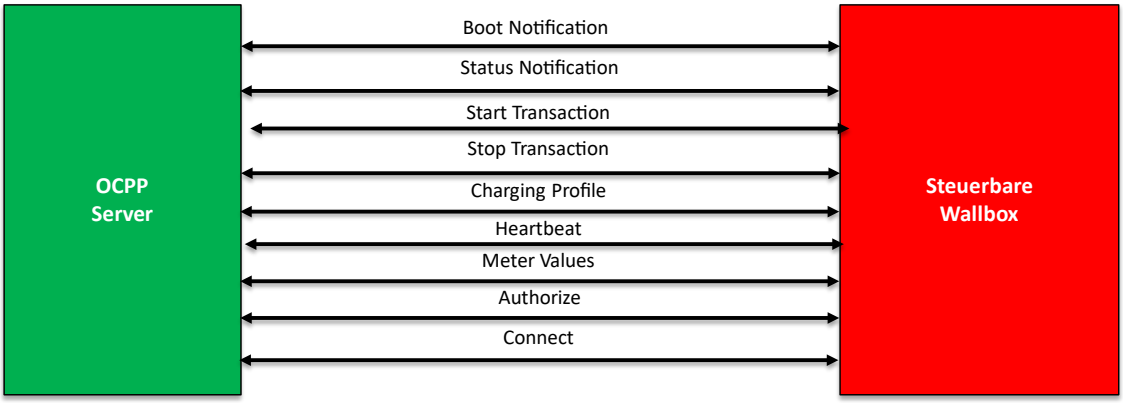

Abbildung 15: Funktionen des OCPP, eigene Darstellung nach [32]

Die Kommunikation mit einer OCPP fähigen Wallbox erfolgt über eine WebSocket-Verbindung, die es erlaubt, bidirektionale Nachrichtenverbindungen in Echtzeit aufzubauen. Die Nachrichteninhalte werden in JavaScript Object Notation (JSON) übertragen. Innerhalb des Demonstrators fungiert ein lokal auf dem Raspberry Pi laufender OCPP-Server, welcher als Python Modul in die Python-Bibliothek *devicecom* implementiert wurde, als Backend. Damit wird eine Anbindung an einen zentralen OCPP-Server überflüssig, und die Wallboxen lassen sich direkt in die dezentrale Regelungsarchitektur integrieren. Die Protocol-Layers des OCPP sind in der Abbildung *16* dargestellt.

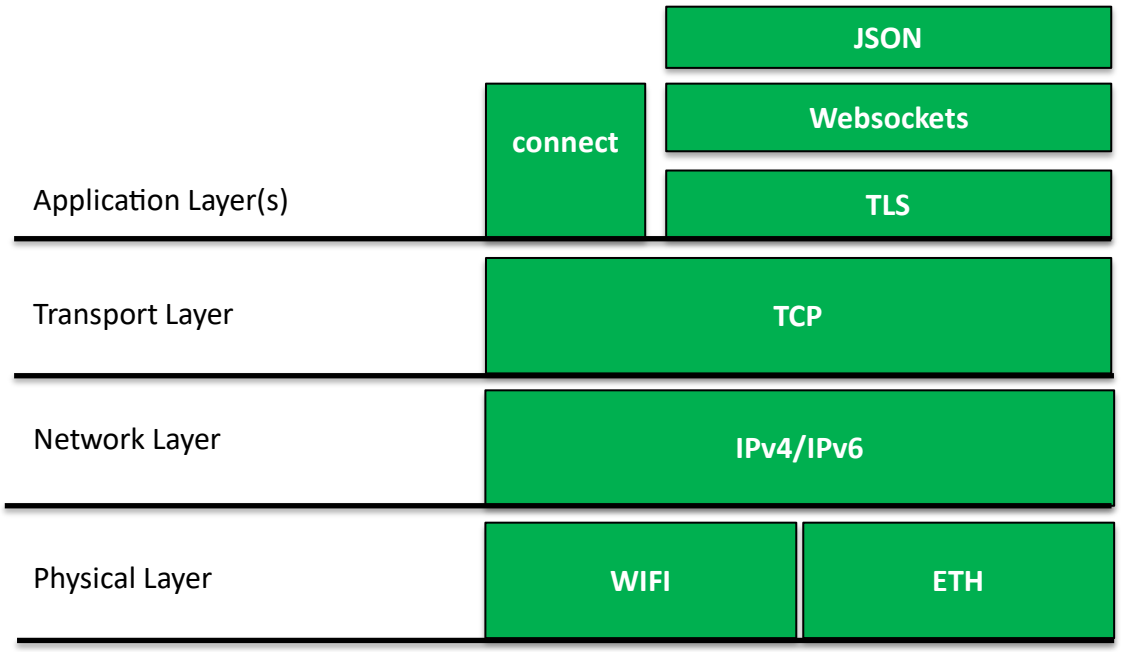

Abbildung 16: Protocol Layers des OCPP, eigene Darstellung nach [32]

Für die Umsetzung des Projektes bzw. des Demonstrators, sind beide Protokolle von zentraler Bedeutung, da durch die Kombination eine umfassende Abdeckung von steuerbaren Lasten erreicht werden kann. So ermöglicht die Nutzung von Modbus-TCP die Einbindung von Wechselrichtern und perspektivisch auch anderen steuerbaren Lasten wie beispielsweise Wärmepumpen und auch älterer Wallboxen, wohingegen OCPP sich auf die Einbindung modernerer Ladepunkte beschränkt.

3.1.3. Kommunikationsinfrastruktur und Datenaustausch

Die Kommunikation zwischen den Netzknoten erfolgt über das User Datagram Protocol (UDP), [33], welches durch eine schnelle und ressourcenschonende Broadcast-Kommunikation ermöglicht. UDP ist eine Alternative zum TCP-Protokoll, welches in diesem Projekt zur Ansteuerung der Geräte genutzt wird. Der Unterschied liegt darin, dass UDP keine Verbindung zwischen dem Versender und dem Empfänger der Daten aufbaut, sondern die Nachrichten einfach gesendet werden. Man spricht von einem Broadcast. UDP sichert somit nicht ab, ob die Pakete wirklich angekommen sind, vgl. [33]. Dies ist jedoch tolerierbar ist, wenn die Sendefrequenz hoch genug ist und immer dasselbe Nachrichtenpaket gesendet wird. Auch ermöglicht dieses Protokoll das Versenden von Daten an mehrere Empfänger gleichzeitig.

Zur Zwischenspeicherung der Messdaten wird ein Redis(Remote Dictionary Server) - Store als temporärer Datenspeicher eingesetzt. Redis ist eine Datenbank, die Werte unter einem gewissen Schlüssel speichert, unter welchem sie danach wieder abgerufen werden können. Aufgrund der geringen Latenzzeiten eignet sich Redis besonders zur Zwischenspeicherung von häufig aktualisierten Daten, wie beispielsweise Messdaten, wie sie in diesem Anwendungsfall anfallen werden. Darüber hinaus unterstützt Redis zeitgesteuerte

Löschfunktionen über Time to Live (TTL), was für die geplanten Hardwareausstattungen essenziell ist, damit die Speicher der Raspberry Pis nicht überfüllt werden. [34]

Die REST-API (Representational State Transfer – Application Programming Interface) [35], stellt eine Schnittstelle zur Verfügung, über die externe Anwendungen, wie etwa Benutzeroberflächen oder andere Python Module, mit dem System interagieren können. Die API basiert auf dem HTTP-Protokoll und definiert sogenannte Endpunkte, über die Befehle gesendet oder Daten abgefragt werden. Die API ermöglicht es, Steuerbefehle wie das Setzen von Ladeleistungen direkt an die angebundenen Geräte weiterzuleiten oder aktuelle Zustandsdaten gezielt aus dem Redis-Store auszulesen, ohne dass eine direkte Verbindung zum Gerät besteht.

Diese Trennung zwischen Datenhaltung (Redis) und externer Schnittstelle (REST-API) stellt sicher, dass die Software weiterhin modular weiterentwickelt werden kann und die Steuer- und die Kommunikationslogik unabhängig voneinander agieren. Gleichzeitig ermöglicht sie eine einfache Integration von weiteren Schnittstellen für Softwaremodule, Grafischen Oberflächen oder Testfunktionen ohne tiefgreifende Änderungen an der Systemarchitektur.

3.1.4. Kommunikationslogik und Geräteintegration

Im Aufbau des Demonstrators übernimmt die selbstentwickelte Bibliothek *devicecom* die Anbindung der steuerbaren Geräte über die Schnittstellen Modbus TCP und OCPP. Die Steuerbefehle werden intern über Python Funktionen ausgeführt, welche wiederum auf die Modbus Register oder OCPP-Module übersetzt werden. Neue Geräte können durch Ergänzung der Konfigurationsdateien integriert werden, ohne die Systemlogik anzupassen.
Diese modulare Herangehensweise erlaubt eine flexible Erweiterung und gleichzeitig eine Umsetzung der im Swarm Grid-Konzept beschriebenen dezentralen Regelung. Jedes Gerät stellt in diesem Szenario seine lokalen Messdaten bereit, überträgt diese per UDP-Broadcast an alle anderen Knoten. Hiermit kann nun die Optimierung jedem Knoten geeignete Sollwerte zuweisen, welche durch die Bibliothek *devicecom* wieder umgesetzt werden können.
Das in [5], beschriebene Konzept des Swarm Grid beschreibt ein dezentrales Steuerungssystem, in dem jeder Netzknoten Messdaten erhebt, über eine dezentrale Kommunikation Informationen austauscht und auf selbem Weg auch Steuerwerte erhält. Ziel ist eine Schwarmintelligenz, bei der alle Netzteilnehmer zur Stabilität und Effizienz des Systems beitragen, ohne dass eine zentrale Steuerung vom MSB erforderlich ist. Dieses Konzept bildet die theoretische Grundlage des im Rahmen dieser Arbeit entwickelten Demonstrators.

3.1.5. Docker Container im Kontext der Softwareentwicklung

Docker ist eine Open-Source-Plattform, vgl. [36], zur Containerisierung von Anwendungen, die in der Softwareentwicklung eine zentrale Rolle einnimmt. Ein Docker-Container ist eine isolierte Umgebung, die eine Anwendung samt aller Abhängigkeiten wie z. B. Bibliotheken und Konfigurationen enthält. Im Gegensatz zu herkömmlichen virtuellen Maschinen teilt sich der Container den Kernel mit dem Host-Betriebssystems, was sie ressourcenschonender und schneller in der Ausführung macht. Dieses Konzept ermöglicht es, Software auf unterschiedlichen Systemen konsistent und reproduzierbar auszuführen, unabhängig davon, um was für eine Art von System es sich handelt, vgl. [36].
In der Softwareentwicklung bietet Docker erhebliche Vorteile. Anwendungen können samt definierter Umgebungen in sogenannten Docker Images bündeln und wobei über sogenannte Dockerfiles beschrieben wird, wie diese Images aufgebaut werden. Dadurch wird verhindert, dass die Software nur auf dem System des Entwicklers läuft, da jeder Container auf jedem System mit Docker identisch läuft. Dies ist besonders wichtig, um die Inbetriebnahme der Software zur Ansteuerung des MicroGrids auf den Raspberry Pis zu vereinfachen. Auch wird eine Zusammenarbeit im Team vereinfacht, da alle Softwarebausteine als Services im Docker Container laufen und somit von allen Beteiligten einfach verwendet werden können.
Besonders komplexere Software-Stacks wie in diesem Fall, lassen sich mithilfe des Docker-Compose Tool gut orchestrieren, vgl. [36]. Hierbei läuft jeder Dienst in einem eigenen Container mit einem eigenen Logging, wodurch die Fehlersucher erheblich vereinfacht wird. Dadurch bietet sich Docker-Compose besonders an, wenn, wie in diesem Fall, mehrere voneinander abhängige Dienste genutzt werden sollen. Dabei werden alle notwendigen Dienste in einer Konfigurationsdatei (docker-compose.yml) beschrieben und werden gemeinsam mit einem einzigen Befehl (docker-compose up) gestartet. [36]
Zur Containerisierung eines Python-basierten Projekts wird in der Regel eine sogenannte „Dockerfile“ erstellt, welche als „Bauanleitung“ für das entsprechende Docker-Image dient. Diese enthält unter anderem die Definition eines geeigneten Python-Images, die Installation notwendiger Abhängigkeiten aus einer sogenannten requirements.txt sowie den Startbefehl für die Anwendung. Anschließend wird aus dieser Konfiguration mithilfe des Befehls „docker build“ ein Image erzeugt, das die komplette Ausführungsumgebung abkapselt. In dem Dockerfile werden auch diejenigen Ports des Systems definiert, welche nach außen verfügbar sein müssen. [36]
Im Projektkontext wurde Docker unter anderem eingesetzt, um eine modulare Trennung zwischen der Steuerlogik, der REST-API, dem Datenbackend (Redis) sowie der Geräteschnittstellen (Modbus und OCPP) zu realisieren. Dies erleichtert die Wartung, die Weiterentwicklung einzelner Module, ohne bereits funktionierende Module zu beeinflussen.
Die Integration von Docker in die Entwicklungsumgebung PyCharm wurde ebenfalls umgesetzt, um eine nahtlose Steuerung und das Debugging der Container direkt aus der integrierten Entwicklungsumgebung (IDE) heraus, zu ermöglichen, vgl. [37]. Über die integrierten Docker-Plugins lassen sich sowohl Images als auch laufende Container verwalten, Logs analysieren und Entwicklungszyklen verkürzen. Sowohl das „Dockerfile“ als auch die „docker-compose.yml“ sind im digitalen Anhang dieser Arbeit verfügbar.

4. Inhalt

4.1. Neuigkeit der Ergebnisse

Obwohl es innerhalb des Forschungsprojektes GridMaximizer mehrere Forschungsarbeiten zur Ansteuerung der Geräte im Microgrid gab, wurde bisher kein Demonstrator zur Veranschaulichung der Steuerung steuerbaren Lasten im MicroGrid aufgebaut. Das Ziel dieser Arbeit ist es, einen solchen Demonstrator aufzubauen und in Betrieb zu nehmen.

4.2. Eigener Beitrag

Der Beitrag dieser Arbeit ist zum Teil die Python Bibliothek *devicecom*, besonders die Callback Funktionen und die Speicherung des letzten Messwerts zur Abfrage für die Kommunikation in einem Redis Store, die Inbetriebnahme, die Dokumentation dieser und das Troubleshooting, so, dass der aufgebaute Demonstrator tatsächlich für die Vorführung des Prinzipes von GridMaximizer genutzt werden kann. Um dies umzusetzen, wird auf die schon umgesetzte Ansteuerung der bereits vorhandenen Geräte wie Wechselrichter, UMD und PQIDA zurückgegriffen, wobei in dieser Arbeit die Ansteuerung von OCPP-Geräten umgesetzt wird. Auch wird in dieser Arbeit das Broadcasting Modul der SYP-Gruppe genutzt, vgl. [38], welches jedoch so angepasst werden muss, dass es den Anforderungen des Demonstrators entspricht. Hierbei liegt der Fokus besonders auf der Kopplung des Redis Stores, welcher die Daten enthält, welche gesendet werden sollen. Zuletzt wird noch eine grundlegende Optimierung eingesetzt, wobei hier sichergestellt werden muss, dass diese Optimierung die vollständige Datensätze aus der Kommunikation erhält, diese verarbeiten kann und die Ergebnisse der Optimierung an die steuerbaren Geräte gesendet werden.

4.3. Systemkonzept und Anforderungen

4.3.1. Übersicht über die benötigte Software und die Anforderungen an diese zur Umsetzung des Demonstrators

Um eine dezentrale Optimierung und Steuerung der im Microgrid der TH-Köln vorhandenen Geräte gewährleisten zu können, muss zunächst die Verbindung der Geräte sicher und stabil umgesetzt werden. Hierzu wurde die Python-Bibliothek *devicecom* entwickelt, welche die Registrierung und die Verbindung der Geräte verwalteten. Auch beinhaltet diese Bibliothek alle nötigen Konfigurationsdateien und Module, welche für die Ansteuerung der Geräte nötig sind. Wichtig sind hierbei besonders die ModBusRegister und das Python Modul, welches die Ansteuerung der UFE-Wechselrichter mit Hexcodes umsetzt. Um die in der Bibliothek *devicecom* umgesetzten Funktionen nutzen zu können und um als Schnittstelle zu den restlichen Systembausteinen zu dienen, wird eine Fast-API REST Schnittstelle genutzt, welche die Möglichkeit bietet, die verschiedenen Funktionen über sogenannte Endpunkte aufzurufen und diese somit für andere Module des Systems verfügbar macht. Die Anforderungen an die Fast-API sind die, dass diese von allen anderen Modulen genutzt werden kann um Anfragen, Befehle und Messwerte zwischen dem Redis-Store und den anderen Modulen auszutauschen. Hierbei stehen insbesondere klar definierte Endpunkte und Funktionen im Vordergrund, welche eine saubere und einfache

Kommunikation zwischen den Modulen und Geräten gewährleisten. Auch soll die Fast-API modular erweiterbar sein

Um die mithilfe der Fast-API und *devicecom* ausgelesenen Messwerte verarbeiten zu können und um einen Informationsverlust zu verhindern, wird ein sogenannter Redis-Store genutzt, welche genutzt werden kann, um Daten zwischenzuspeichern. Die Anforderungen an diesen Speicher sind, dass die Daten sicher und schnell zwischengespeichert werden, ohne hohe Hardwareanforderungen an das System zu stellen. Durch die in Kapitel 3.1.3 beschriebenen Eigenschaften von Redis, eignet sich dieses Tool besonders für diesen Anwendungszweck. Um die gewünschte Dezentralität des Demonstrators zu erreichen, muss es möglich sein, die in Redis gespeicherten Daten zu versenden und zu empfangen. Die Anforderungen an die Nachrichtenübertragung sind hierbei vorrangig, dass alle im System teilnehmenden Knoten die Informationen aller anderen Knoten empfangen können, beziehungsweise, dass jeder Knoten seine eigenen Informationen allen anderen Knoten zur Verfügung stellt. Hierbei ist es wichtig, dass so viele eingehende Nachrichten wie möglich erfasst werden, da größere Lücken zu einem Informationsausfall und damit zum Ausfall der Ansteuerung der ausgefallenen Geräte führt.

Für eine effektive Umsetzung dieses Konzepts sind spezifische Anforderungen an die Kommunikation definiert worden. Die Betriebsdaten müssen in Echtzeit übertragen werden und beinhalten Messgrößen wie die Phasenspannungen und -ströme, ggfs. Leistungsfaktoren und Spannungsphasoren. Die Gesamtgröße einer Nachricht beträgt 32 Byte, welche sich wie in Tabelle 2 dargestellt auf die verschiedenen Größen aufteilen.

Tabelle 2: Nachrichteninhalt und Länge, vgl. [5].

Item	Bytes
Phasenspannung	6
Phasenströme	6
Leistungsfaktor	6
Spannungsphasor	6
Benötigte Energiemenge	2
Gewünschter Endzeitpunkt	2
Station ID	2
Status	1
Prüfsumme	1
Summe	32

Zusätzlich zu den gemessenen Werten an jedem Knoten müssen noch zusätzliche Informationen übertragen werden, um die Optimierung optimal mit Daten zu versorgen. Diese sind im Folgenden beschrieben.

- Benötigte Energiemenge: Diese gibt an, wie viel Energie ein Gerät bis zu einem definierten Endzeitpunkt benötigt. Dies kann beispielsweise der Fall sein, wenn ein Elektrofahrzeug bis zu einem gewünschten Zeitpunkt einen gewissen Ladezustand erreichen soll oder mithilfe einer Wärmepumpe eine bestimmte Temperatur in einem Gebäude erreicht werden soll. Diese Information wird gemeinsam mit dem zugehörigen Endzeitpunkt übertragen, sodass die Daten in der Optimierung berücksichtigt werden können. Um eine ausreichend hohe Auflösung sicherzustellen, sind hierfür zwei Byte vorgesehen.
- Gewünschter Endzeitpunkt: Dieser gibt an, bis wann ein Gerät die gewünschte Energiemenge benötigt. Um eine Auflösung von einer Minute zu gewährleisten, also 1.440 Zeitwerte pro Tag, sind ebenfalls zwei Byte erforderlich.
- Station ID: Zur eindeutigen Identifikation wird jedem Knoten eine ID zugewiesen. Obwohl ein Byte theoretisch reichen würde, um 256 Geräte in einem Netzbereich abzubilden, werden zwei Byte reserviert, um eine eventuelle Erweiterung auf benachbarte Netzsegmente zu ermöglichen.
- Status: Das Statusbyte dient zur Übermittlung des Betriebszustandes, von Warnungen oder etwaigen Fehlern am Knoten. Ein Byte kann hierbei theoretisch 256 unterschiedliche Zustände darstellen, was in diesem Fall völlig ausreichend ist.
- Prüfsumme: Da die Datenpakete per UDP-Broadcast versendet werden und in diesem Protokoll keine Empfangsbestätigung vorgesehen ist, wird zur Sicherstellung der Datenintegrität eine Prüfsumme über ein zusätzliches Byte integriert.
- Gesamt: Damit ergibt sich eine Größe von 32 Byte pro Nachricht, die zwischen den Geräten ausgetauscht wird.

Bei maximal 256 Netzteilnehmern ergibt sich ein theoretischer Datenbedarf von 8.192 Byte/s, was bei einer Broadcast-Übertragung realisiert werden kann, vgl. [5].
Zur Übertragung der oben beschriebenen Daten wurden im Projekt verschiedene Technologien untersucht und getestet:

- Powerline Communication (PLC / Narrowband G3-PLC): Diese nutzt die bestehende Stromverkabelung für die Datenübertragung. Getestet wurden devolo G3-PLC-Modems mit einer Übertragungsrate von bis zu 65 kbit/s. Die Technologie erwies sich grundsätzlich als geeignet, wies jedoch eine Paketverlustrate von bis zu 4 % auf, insbesondere wenn die Kommunikation über unterschiedliche Phasen oder längere Leitungen erfolgte, vgl. [39]
- Broadband Powerline (BPL): Diese Technik, getestet mit Devolo Magic 2 Geräten auf Basis des G.hn-Standards, zeigte in Kombination mit den Wechselrichtern im MicroGrid erhebliche Störeinflüsse. Bereits bei moderater Leistungsaufnahme kam es zu kompletten Kommunikationsausfällen, was BPL für reale Netzbedingungen untauglich, vgl. [40]
- LoRa: In Tests mit RAK3276-Modulen und Raspberry Pi 4 zeigte sich, dass die Reichweite stark von der Antennenpositionierung abhängt. In Gebäuden führten schon Innenwände zu stark reduzierten Empfangsraten. Die Verbindung war nur mit direkter Sichtlinie und optimaler Positionierung stabil möglich. Daher eignet sich LoRa nur bedingt für kritische Anwendungen mit hohen Anforderungen an die Zuverlässigkeit, vgl. [39]
- 5G / DECT-2020 NR+: Diese Funktechnologie wurde mithilfe von nRF9151-DK Boards der Firma Nordic Semiconductor getestet. Im Freifeld konnten bei direkter Sichtverbindung Reichweiten von über 760 m mit weniger als 0,3% Paketverlust erzielt werden. Auch unter nicht idealen Bedingungen, z. B. aus einem Gebäudekeller heraus, war eine Kommunikation bei entsprechender Antennenplatzierung möglich. Die hohe Zuverlässigkeit, Mesh-Fähigkeit die Verfügbarkeit eines lizenzfreien Frequenzbands machen NR+ zur derzeit vielversprechendsten Technologie für Swarm-Grids, vgl. [41]

Insgesamt zeigen die Untersuchungen, dass eine robuste, echtzeitfähige Kommunikation zwischen den Netzteilnehmern möglich ist. Voraussetzung hierfür ist, dass die eingesetzte Technologie anwendungs- und standortspezifisch eingesetzt wird. Das Swarm-Grid-Konzept stellt somit einen zukunftsweisenden Ansatz für eine flexible, resiliente Netzführung dar, insbesondere für die Integration erneuerbarer Energien in dezentralen Strukturen, vgl. [5]. Zur Veranschaulichung des Prinzips der dezentralen Regelung am Demonstrator an der TH Köln wird zunächst noch einmal auf ein lokales Ethernet Netzwerk zurückgegriffen werden, jedoch sehen die weiteren Entwicklungen die Nutzung der oben genannten physical layer vor.

Da für die Umsetzung des Demonstrators nicht nur Messwerte, sondern auch Kundenwünsche berücksichtigt werden sollen, wurde eine grafische Nutzeroberfläche (GUI) entwickelt, welche es ermöglicht, eben diese oben beschriebenen Kundenwünsche mit in das

System einzubringen. Die Anforderungen an die GUI sind es, dass ein Ladevorgang gestartet werden kann, bei dem die Rahmenbedingungen des Ladevorgangs mit eingegeben werden können. Auch diese eingegebenen Daten werden in der Folge im Redis Store gespeichert, aus welchem sie später dann zusammen mit den Messwerten für die zu sendende Nachricht und die Optimierung extrahiert werden.

4.3.2. Hardwarevoraussetzungen zur Umsetzung des Demonstrators

Um eine dezentrale Optimierung und Steuerung der im Microgrid der TH-Köln vorhandenen Geräte gewährleisten zu können ist anders als bei dem Konzept der Steuerbox FNN kein SMGW notwendig, sondern es reicht ein Raspberry Pi mit einem angebundenen physical layer wie zum Beispiel Ethernet über Powerline, LoRa oder 5G aus. Dies gilt für den Fall, dass Geräte angesprochen werden sollen, welche selbst Messdaten zur Verfügung stellen können. Im Fall des MicroGrids ist dies sowohl für die Wechselrichter, welche zum Laden der Batterien und damit zur Simulation von Lasten im Netz verwendet werden, als auch für die Wallboxen, welche mit dem OCPP angesprochen werden sollen, der Fall ist.
Um ein detailliertes Bild über den Zustand des Netzknotens zu erhalten, wird im MicroGrid auf an die an jedem Knoten installierten Messgeräte, welche eine Vielzahl an Messgrößen messen können, zurückgegriffen. Eine solche Ausstattung ist für den Betrieb der dezentralen Optimierung und Steuerung jedoch nicht zwingend notwendig, da wie bereits erwähnt teilweise die Messwerte der zu steuernden Geräte genutzt werden können und im Zusammenhang mit dem Forschungsprojekt eine Messplatine entwickelt wurde, welche mindestens Spannung und Ströme an einem Knoten messen kann. Diese kann auch direkt am Hausanschluss genutzt werden, um die Werte für den ganzen Netzknoten zu erheben, wie das auch die deutlich teureren Messgeräte der Firma A-Eberle und UMD-98 tun können.

Zusammenfassend beschränkt sich die hinreichende Hardware pro Netzknoten also auf einen Raspberry Pi, einen physical layer und gegebenenfalls die im Rahmen des Projektes Progressus entwickelte Messplatine, welche als Ersatz für ein Smart Meter dienen soll, vgl. [42]. Aufgrund des Aufbaus des MicroGrids wird jedoch auf die vorhandenen Messgeräte zurückgegriffen.
Für die Umsetzung des Demonstrators wird ein Netzwerk aus 3 Raspberry Pis aufgebaut werden, wobei 2 Netzknoten mit Wechselrichtern versehen sind, um Ladevorgänge zu simulieren und ein Netzknoten mit einem Lastwiderstand versehen ist, um nicht steuerbare Lasten zu simulieren, welche sich auf die Ladevorgänge auswirken sollen.

4.4. Systemarchitektur und Modulkategorien

Auf Softwareseite wurde für das MicroGrid und die Ansteuerung der verschiedenen Geräte eine eigene Bibliothek mit dem Namen *devicecom*, vgl. [43], für Device Communication erstellt. Grundsätzlich umfasst diese Bibliothek zum jetzigen Stand einen OCPP-Server um als Endpunkt für die OCPP fähigen Wallboxen im Netz zu dienen und eine umfassende Code Struktur, welche die Initialisierung, Verbindung und Steuerung der ModBusTCP fähigen Geräte im Netz übernimmt.
Diese ist so aufgebaut, dass Sie modular erweitert und verbessert werden kann. So ist in der Theorie für ein neues Gerät mit der Schnittstelle ModBusTCP lediglich ein Eintrag in die entsprechende Konfigurationsdatei mit der jeweiligen IP-Adresse des Gerätes und in die Konfigurationsdatei für die jeweiligen ModBusRegisters zu tätigen. Ist dies geschehen, greifen alle weiteren Scripte auf diese definierten Schnittstellen zurück und binden das Gerät automatisch ein und ermöglichen die Steuerung des Gerätes mithilfe der integrierten Logik.

Die entwickelten Module können hierbei in vier Kategorien aufgeteilt werden. Zunächst gibt es die Kategorie *device*, welche alle Module umfasst, welche für die Konnektivität und wie oben beschrieben, die direkte Kommunikation und Steuerung der genutzten ModbusTCP Geräte verantwortlich ist.
Des Weiteren gibt es die Kategorie *adapter*, welche alle Module umfasst, welche nötig sind, um von einem Endgerät auf die durch die *devices* verfügbar gemachten Informationen zuzugreifen und von einem Endgerät Steuerbefehle an die Geräte zu senden. In der Kategorie *adapters* befindet sich unter anderem auch der OCPP-Server, welcher lokal gestartet wird, um einen Endpunkt für die angeschlossene Wallbox zu bieten. Weiterhin umfasst die Kategorie *adapter* die oben erwähnte REST-Schnittstelle, welche die Endpunkte zum Aufruf der verschiedenen Funktionen enthält. Diese REST Schnittstelle ist vor allem wichtig, um die von den Geräten gesammelten Messwerte an einen wie oben beschriebenen Redis-Store zu senden, um einen uneingeschränkten Zugriff auf die Messdaten zu ermöglichen. Die REST API-Schnittstelle wurde definiert, um Befehle auch manuell über einen Web Endpoint auslösen zu können, um Funktionen zu testen. In der REST API sind verschiedene Endpoints, sogenannte Routen erstellt worden, welche die verschiedenen Funktionen ermöglichen.
Die dritte Kategorie ist die des Graphical User Interfaces (GUI). Diese umfasst die Web GUI, welche es ermöglicht, Kundenwünsche einzutragen und somit einen Ladevorgang zu starten, die Messwerte der einzelnen Geräte abzufragen und diese grafisch darzustellen.
Die letzte Kategorie ist die der *central_node*. In dieser Kategorie laufen alle oben erwähnten anderen Kategorien zusammen. Die *central_node* wird genutzt, um die zu sendenden Nachrichten zu formulieren und die eingehenden Nachrichten auszuwerten. Auch verarbeitet die *central_node* die Messwerte und startet die Optimierung. Sie stellt somit den zentralen Punkt an jedem Netzknoten dar.
Die Softwarearchitektur wurde gezielt so entwickelt, dass ein einzelner Raspberry Pi mit minimaler Hardwareanforderung in der Lage ist, ein vollständiges Steuerknotenmodul zu

bilden. Die Architektur folgt einem modularen Aufbau, der zwischen Geräteschnittstellen, Kommunikationsadaptern und der Benutzerinteraktion unterscheidet.

4.5. Detaillierte Beschreibung der Software

Um die Funktionsweise der Software darzustellen, wurde ein Ablaufdiagramm erstellt, welches die Software mit ihren unterschiedlichen Modulen darstellt und die Anfragewege zwischen den einzelnen Modulen abbildet, siehe Abbildung *17*. Dabei sind nicht alle Module, Klassen und Funktionen einzeln dargestellt, jedoch gibt die Abbildung eine detaillierte Übersicht über die Informationsflüsse und die involvierten Teile der Software, um eine ganze Laufzeit der Informationen darzustellen. Im Folgenden Kapitel wird die genaue Umsetzung der verschiedenen Funktionen und auch die zugehörigen REST API-Routen genauer beschrieben und dargelegt.

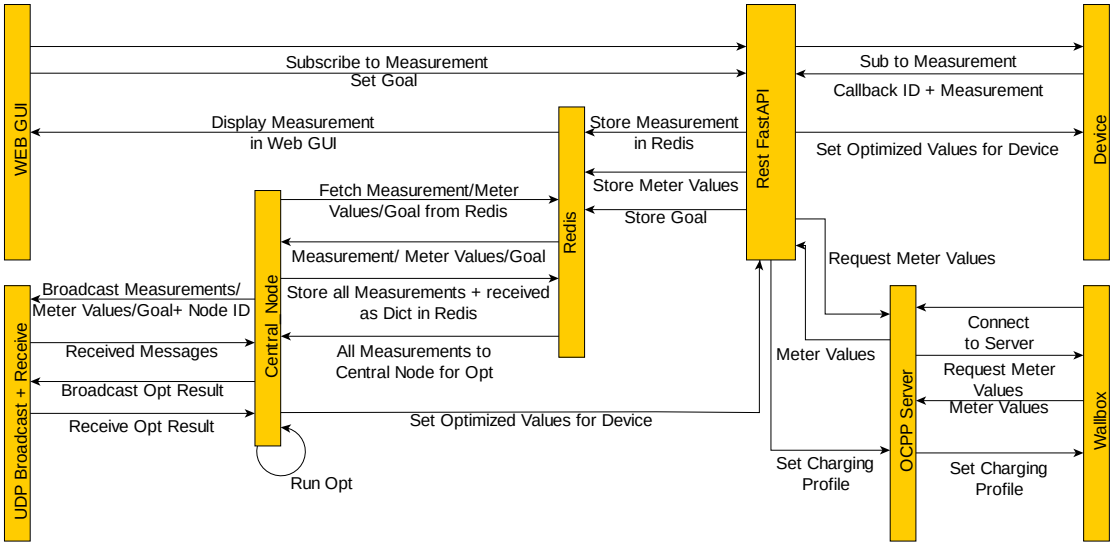

Abbildung 17: Ablaufdiagramm der entwickelten Software, eigene Darstellung

4.5.1. Python Bibliothek „devicecom“

Die Python Bibliothek *devicecom* [43] wird in diesem Projekt genutzt und weiterentwickelt, um die Ansteuerung der im MicroGrid verbauten Geräte zu gewährleisten. Zu dieser Bibliothek gehören alle jenen Python Module, welche direkt für die Verbindung und die Steuerung der angeschlossenen Geräte notwendig sind. Einige Module waren zum Zeitpunkt der Arbeit bereits fertig gestellt und gehen aus verschiedenen Arbeiten hervor, welche bereits am MicroGrid durchgeführt wurden. Hierzu zählt insbesondere die Ansteuerung der Messgeräte „Universal Measuring Device 98“ (UMD98), der PQIDA Smart Messgeräte der Firma A. Eberle und der Wechselrichter der Firma UFE. Auf der obersten Ebene dieser Bibliothek stehen allen Geräten die Konfigurationsdatei mit den Geräten und den zugehörigen IP-Adressen zur Verfügung. Des Weiteren liegen dort auch

die Gerätespezifischen Modbus Register gespeichert, welche für die Ansteuerung der Geräte genutzt werden.

4.5.1.1. *Device_register*

Um zuverlässig eine Verbindung zu den Geräten aufbauen zu können, steht das Python Modul *device_register* zur Verfügung. Dieses registriert alle angeschlossenen Geräte anhand ihrer IP-Adressen aus einer Konfigurationsdatei, der *device_register.yaml* und speichert alle so registrierten Geräte in eine weitere Konfigurationsdatei, aus welcher die angeschlossenen Geräte später aufgerufen werden können. Hiermit kann also gesteuert werden, welche Geräte tatsächlich an einem Knoten in die Software eingebunden werden. Für die Umsetzung dieses Projektes wird die Konfigurationsdatei so angepasst, dass jeweils die physisch an einem Knoten angebundenen Geräte alle einem Raspberry Pi zugeordnet werden. Über diese Konfigurationsdatei und die Angabe einer neuen IP-Adresse und dem Namen des Geräts ein neues Gerät hinzugefügt werden.

4.5.1.2. Selectors

Im Modul *selectors* werden die sogenannten Selektoren zusammengebaut, welche, wenn sie später aufgerufen werden auf bestimmte Funktionen bzw. Modbus Register der verschiedenen Geräte verweisen und somit die Zuordnung der Funktionen des Geräts zu den jeweiligen Registern übernehmen. Hierbei werden sowohl *MeasurementSelectors* als aus *ControllableSelectors* erstellt, um eine klare Trennung zwischen Abfragen und Steuerbefehlen zu gewährleisten.

4.5.1.3. *Controllable* und *Measurement*

Die in *selectors* erstellten Selektoren werden nun in den Modulen *Controllable* und *Measurement* genutzt in welchen die Selektoren zunächst validiert werden. Dies geschieht, damit keine falschen Anfragen und Steuerbefehle an die Geräte gesendet werden, welche zu Störungen der Software und der Geräte führen können. Im Fall der *Controllables* wird hierbei auch zwischen einer *OptionControl*, also einem Zustandswechsel und der *ContinuousControl* also einem Spektrum unterschieden. Die *OptionControl* wird zum Beispiel verwendet, um die Schalter im MicroGrid zu steuern. Wichtiger für diese Arbeit sind die *ContinuousControls.* Diese werden genutzt, um die UFE-Wechselrichter steuern zu können. In den *ContinuousControls* werden daher auch die Ober- und Untergrenzen der Werte eingestellt, auf die die Geräte maximal gesetzt werden können.
Im Modul *Measurement* ist hierbei die Abfrage der Messwerte implementiert. Insbesondere ist hier die sogenannte Subscription auf einen *MeasurementSelector* umgesetzt, welche in dieser Arbeit genutzt wird, um die Messwerte der Geräte abzufragen. Gleichzeitig wird in diesem Modul auch die *callback_id* zurückgegeben, welche später auch in der REST-API genutzt wird, um die Messwerte verwalten zu können. Diese *callback_id* wurde im Zuge dieser Arbeit hinzugefügt, um die Aufnahme der Messwerte zu vereinfachen. Es ist sowohl eine Funktion für das Abonnieren einzelner Messwerte oder mehrerer Messwerte gleichzeitig umgesetzt. Analog dazu gibt es auch zwei Funktionen, welche das Deabonnieren der Messwerte ermöglichen. Diese basieren dann jedoch nicht mehr auf den Selektoren, sondern lediglich auf der *callback_id* was das Deabonnieren der Messwerte vereinfacht.
Sind die Messwerte abonniert, so werden sie mit der Funktion *poll_measurement* zyklisch abgefragt.

4.5.1.4. *Device*

Die vorherigen Module werden alle zusammengefasst in dem Modul *device*. Dieses dient als Vorlage für die Gerätespezifischen *device* Module, welche für jedes Gerät, welches genutzt werden soll, erstellt werden müssen. Dies erleichtert die Einbindung von neuen Geräten in die bestehende Software. Ist das Modul für das jeweilige Gerät angepasst, umfasst es alle Funktionen, welche mit dem Gerät umgesetzt werden können. Im Falle der Messgeräte ist besonders die Funktion *get_measurements* aus dem Modul für die UMD98 von Bedeutung, welche das Modul *selectors* in Verbindung mit dem Modul *Measurement* dafür nutzen die Messwerte des Geräts abzufragen.

Analog nutzt die Funktion *set_Value* aus dem Modul VSI_Rack die Selektoren und das Modul *Controllable,* um den Ladestrom für die Wechselrichter zu setzen. Das VSI_Rack bündelt hierbei die Steuerung aller drei Wechselrichter an einem Netzknoten in ein Modul, um die Ansteuerung zu erleichtern. Hier sind auch die Gerätespezifischen Minimal- und Maximalwerte für die Steuerung der Geräte angegeben. Im Falle der UFE-Wechselrichter sind dies für den Batteriestrom -70 A bis +70 A, oder der Netzstrom von -10 A bis +10 A.

4.5.2. REST-API-Schnittstelle

Die rest.py-Datei stellt eine zentrale Schnittstelle zur Interaktion mit den im MicroGrid eingesetzten Geräten dar. Zentral bedeutet in diesem Fall zentral für jeden Raspberry Pi, nicht notwendigerweise zentral für das ganze MicroGrid. Sie ermöglicht sowohl das Abrufen von Messdaten als auch das Setzen von Steuerbefehlen. Auch die Kommunikation mit OCPP-fähigen Wallboxen wird über die REST-API und WebSockets umgesetzt. Die Schnittstelle ist so konzipiert, dass sie über definierte Endpunkte von der Web-GUI oder anderen Modulen angesprochen werden kann. Eine Übersicht der einzelnen Endpunkte und ihre jeweiligen Ziele, können der Abbildung 18 entnommen werden.

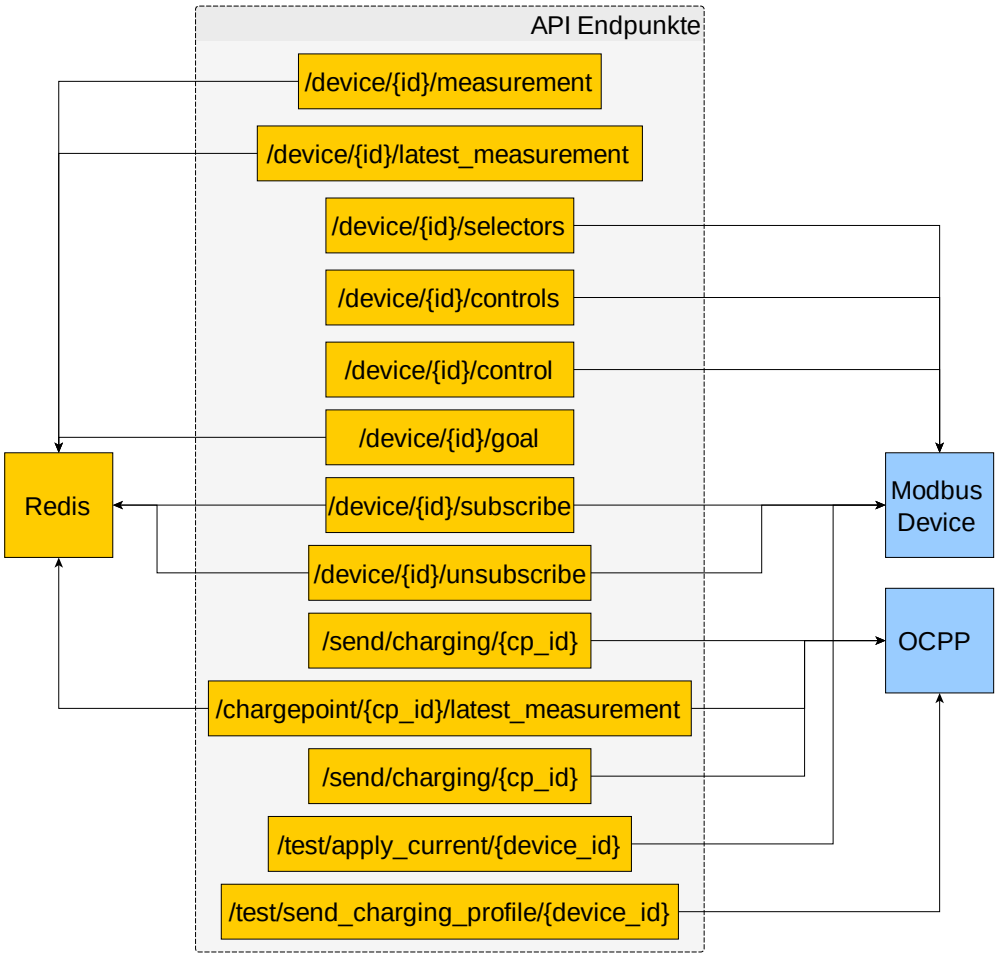

Abbildung 18: Darstellung der REST-API Endpunkte und deren Gegenspieler, eigene Darstellung.

Für den Betrieb der entwickelten Softwarearchitektur im Rahmen des MicroGrid-Systems spielen einige ausgewählte API-Endpunkte eine zentrale Rolle. Diese Endpunkte stellen die essenzielle Schnittstelle zwischen den in Kapitel 4.5.1 beschriebenen Funktionen der einzelnen Geräte, der Central_Node und der grafischen Benutzeroberfläche dar.

Insbesondere für den laufenden Betrieb, also die kontinuierliche Überwachung von Messgeräten und der Steuerung der Wechselrichter sind folgende Endpunkte von Bedeutung:

- /device/{device_id} /latest_measurement
- /device/{device_id} /control
- /device/{device_id} /goal
- /device/{device_id} /subscribe
- /device/{device_id} /unsubscribe
- /send-charging/{cp_id}
- /meter_values/{cp_id}

Ergänzend zu diesen produktiven Endpunkten existieren zwei Test-Endpunkte (/test/apply_current/{device_id}), (/test/send_charging/{device_id}), mit denen es möglich ist das Setzen eines optimierten Stromwertes und eines optimierten Charging Profils zu simulieren. Diese Endpunkte werden vor allem in der Entwicklungs- und Testphase genutzt, um das Verhalten der Steuerbaren Lasten bei Aufruf der Steuerfunktionen zu prüfen, ohne dass eine vollständige Optimierungsberechnung im Vorfeld notwendig ist.

4.5.2.1. Endpunkt /{device_id}/subscribe

Die Grundlage für jede Optimierung und für die spätere Steuerentscheidung bilden die aktuellen Messwerte der angeschlossenen Geräte. Um diese Messwerte regelmäßig und automatisiert bereitzustellen, greift die Software nicht auf den einfachen Endpunkt /device/{device_id}/measurement zurück, der für einmalige Abfragen einzelner Messwerte konzipiert ist. Stattdessen wird der Endpunkt /device/{device_id}/subscribe verwendet, um sogenannte „Messwert-Subscriptions“ einzurichten. Innerhalb dieses Endpunktes wird die Hilfsfunktion *validate_measurement_selector* aufgerufen, welche dazu dient zu validieren, ob die angefragten Selektoren auch tatsächlich der richtigen Syntax entsprechen und auf einen der in der Konfiguration festgelegten Modbus Register zeigen.

Ein Subscribe-Aufruf erlaubt es, für ein bestimmtes Gerät eine oder mehrere Messgrößen zu abonnieren. In diesem Fall Strom und Spannung. Hierzu wird dem Endpunkt eine Liste sogenannter Measurement-Selektoren übergeben, die Parameter wie Phase, Intervall und Messgröße spezifizieren. Diese Selektoren sind werden durch die entsprechende Funktion in *devicecom* gebaut und sprechen somit die richtigen Modbus Register pro Gerät an.

Mit jedem Abonnement wird im Hintergrund eine eindeutige *callback_id* generiert. Diese ID dient als Referenz für den gesamten Subscription-Vorgang und wird benötigt, um später gezielt einzelne Abonnements beenden zu können, ohne wissen zu müssen, welche Messgrößen genau abonniert sind. Diese Trennung ist besonders relevant, wenn mehrere Abonnements parallel existieren, z.B. wenn in zwei Geräte auf die Messwerte desselben Geräts zugreifen wollen. Der mit dem Subscription-Prozess verknüpfte Callback sorgt dafür, dass alle ermittelten Messdaten automatisiert im Redis-Store gespeichert werden, wobei die Schlüssel jeweils die device_id enthalten, um eine eindeutige Zuordnung der Daten zu gewährleisten.

4.5.2.2. Endpunkt /{device_id}/latest_measurement

Die im Redis-Store gespeicherten Messwerte stehen anschließend dem Endpunkt /device/{device_id}/latest_measurement zur Verfügung. Dieser wird regelmäßig von der grafischen Benutzeroberfläche (GUI) aufgerufen, um die aktuellen Werte darzustellen. Die zurückgelieferten Daten werden dabei so aufbereitet, dass beispielsweise Strom- und Spannungswerte unter standardisierten Feldnamen wie "current" und "voltage" abrufbar sind, unabhängig von der konkreten Formatierung im Ursprungsgerät. Dies ist nötig, da sich die Form der Daten zwischen ModBusTCP Geräten und OCPP-Geräten deutlich unterscheidet und so eine fehlerhafte Nachricht zustande kommen würde, welche infolgedessen die Optimierung verhindern kann.

4.5.2.3. Endpunkt /{device_id}/goal

Ein weiterer essenzieller Baustein für den Betrieb des MicroGrid-Systems ist die Übergabe von Ladezielen an die Optimierung. Dies geschieht über den Endpunkt /device/{device_id}/goal. Über diesen können aus der Benutzeroberfläche, in diesem Fall über die WEB GUI, welche später beschrieben wird Ladeziele an ein spezifisches Gerät übermittelt werden, um dessen zukünftiges Ladeverhalten zu steuern.

Die zu übermittelnden Parameter sind in diesem Fall der gewünschte Ladeendstand des Fahrzeugs in Prozent (target_soc), die geplante Abfahrtszeit (departure_time), der aktuelle Ladezustand (current_soc) sowie die absolute Batteriekapazität des Fahrzeugs in Kilowattstunden (battery_capacity). Diese Angaben werden in der zentralen Redis-Datenbank gespeichert und sind somit sowohl für die spätere Optimierung als auch für andere beteiligte Module jederzeit verfügbar.

Durch die Trennung zwischen aktuellen Messwerten, latest_measurements, und benutzerdefinierten Zielwerten, goal, wird eine klare Trennung zwischen regelmäßig zu sendenden Werten und im Bedarfsfall zu sendenden Werten erreicht. Die Zielwerte werden hierbei nur im Bedarfsfall gesendet und dienen ausschließlich als zusätzlicher Input für die Optimierung zur Einhaltung von Kundenwünschen. Dadurch wird auch eine Reduzierung des Traffics auf dem physical layer erreicht, was auch zur Sicherstellung der Übertragung der Informationen beiträgt.

4.5.2.4. Endpunkt /{device_id}/control

Sobald durch die Optimierung oder einen externen Eingriff (z. B. manuelles Testing) ein konkreter Steuerwert eingegeben wird muss dieser an das entsprechende Gerät übermittelt werden. Der Endpunkt, welcher für die Ansteuerung der ModBusTCP fähigen Geräte vorgesehen ist /device/{device_id}/control.

Dieser Endpunkt benötigt neben dem Wert des zu setzenden Stroms in Ampere, sogenannte „Controllable Selectors“ entgegen. Diese verweisen in ihrer Gesamtheit auf ein bestimmtes ModBus Register aus der Konfiguration, mit welchem die Steuerung dann umgesetzt wird.

Das mapping der Selektoren auf diese festgelegten ModBusRegister sorgt für eine eindeutige Ansteuerung der Geräte und stellt sicher, dass der Befehl nur dann ausgeführt wird, wenn dieser auch korrekt formuliert wurde und ein entsprechendes ModBusRegister für ein Gerät existiert. Eine vorherige Validierung sorgt dafür, dass nur gültige Kombinationen akzeptiert werden – etwa, wenn das Gerät tatsächlich auf Phase 1 steuerbar ist. Die Übertragung erfolgt synchron und enthält im Erfolgsfall eine Bestätigung.

Für die Anbindung von OCPP-kompatiblen Wallboxen an das MicroGrid-System stehen zwei spezielle Endpunkte zur Verfügung: /send-charging/{cp_id} und /meter_values/{cp_id}. Sie dienen als Brücke zwischen der lokalen Steuerlogik und dem Kommunikationsprotokoll OCPP 1.6.

4.5.2.5. Endpunkt „send_charging“

Der Endpunkt /send-charging/{cp_id} wird verwendet, um einen optimierten Ladestrom in Form eines Charging Profiles an die jeweilige Wallbox zu übertragen. Hierbei ruft die

Software intern die Methode *send_charging_profile* des verbundenen Charge Points auf, vgl. 4.5.3.7. Diese Funktion wird ausschließlich verwendet, wenn eine Optimierung bereits durchgeführt wurde und deren Ergebnis aktiv an die Wallbox weitergegeben werden soll.

4.5.2.6. Endpunkt /meter_values/{cp_id}

Ergänzend dazu ermöglicht der Endpunkt /meter_values/{cp_id} das Abfragen von sogenannten MeterValues-Nachrichten. Diese Nachrichten sind Bestandteil des OCPP-Protokolls und enthalten aktuelle Messdaten wie Strom, Spannung oder Energieverbrauch. Dieser Endpunkt wird automatisch genutzt, wenn die Gesamtsoftware feststellt, dass eine OCPP-fähige Wallbox verbunden ist, um von dieser Messwerte zu erhalten. Besonders für die Kategorie der OCPP-Wallboxen ist, das diese gleichzeitig Messgerät und steuerbare Last sein können.

Beide Endpunkte interagieren nur mit sogenannten *connected_chargepoints*, welche beim Start des OCPP-Servers verwaltet werden. Dies stellt sicher, dass nur tatsächlich verbundene Wallboxen angesprochen werden können, und reduziert gleichzeitig die Komplexität bei der Geräteadressierung.

4.5.2.7. Endpunkt /test/apply_current/{device_id}

Um auch ohne eine vollständig implementierte Optimierung das Verhalten der angeschlossenen Geräte zu testen, wurde ein dedizierter Test-Endpunkt implementiert: /test/apply_current/{device_id}. Dieser erlaubt das direkte Setzen eines Stromwerts auf ein ModbusTCP-fähiges Gerät, ohne dass zuvor eine Optimierung oder ein Zielwert gesetzt werden muss. Der Endpunkt dient in der Praxis zur Verifikation der Ansteuerlogik, zum Debugging einzelner Komponenten und für einfache Funktionstests bei der Inbetriebnahme. Intern wird dieselbe Methode (*apply_optimized_current*) genutzt wie im normalen Ablauf, sodass sichergestellt ist, dass der Kommunikations- und Steuerungsweg vollständig funktioniert.

4.5.2.8. Endpunkt /test/send_charging_profile/{device_id}

Analog zum Endpunkt /test/apply_current/{device_id} ist mit dem Endpunkt /test/send_charging_profile/{device_id} ein Endpunkt umgesetzt, welcher es erlaubt, das Setzen eines Ladeprofils für die Wallboxen per OCPP zu testen, ohne das seine vorherige Optimierung ablaufen musste. Intern wird hierbei die *send_charging* Methode aus Python genutzt, die das Charging Profil erstellt und mit der Funktion *send_charging_profile* ein Ladeprofil an die Wallbox sendet.

4.5.3. Integrierter OCPP-Server

Der in dieser Arbeit implementierte OCPP-Server bildet einen zentralen Bestandteil der Kommunikationsarchitektur für den Demonstrator und stellt die Verbindung zu OCPP-fähigen Ladestationen (Wallboxen) her. Technisch basiert die Implementierung auf der Python-Bibliothek „ocpp“ in der Version 1.6, welche die Protokolllogik des Open Charge Point Protocols abbildet.
Eine schematische Übersicht über die Funktionsweise des OCPP-Servers in Verbindung mit einer verbundenen Wallbox und der Central Node liefert die Abbildung 19. Hierbei laufen die Funktionen in ihrer Reihenfolge von oben nach unten ab.

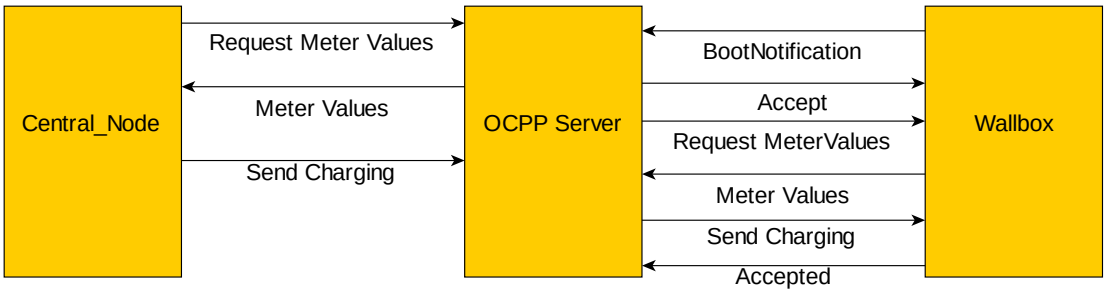

Abbildung 19: OCPP-Server mit Funktionen, eigene Darstellung

4.5.3.1. on_connect

Der OCPP-Server wird beim Start des Systems über eine asynchrone WebSocket-Verbindung initialisiert und wartet auf eingehende Verbindungen von Ladestationen mithilfe der Funktion „on_connect“. Sobald sich eine Wallbox verbinden möchte, wird mithilfe dieser Funktion die Websocket Verbindung initialisiert. Außerdem wird der neuen Wallbox eine „ChargePoint-ID“ mitgegeben, welche allerdings nur vorübergehend genutzt wird, um die Verbindung zu halten.

4.5.3.2. *Boot_notification*

Nachdem mithilfe der „on_connect“ Funktion eine Websocket Verbindung aufgebaut wurde, wird eine sogenannte „BootNotification“ gesendet. Diese enthält den sogenannten „charge_point_vendor“, also den Hersteller der Wallbox und das sogenannte „charge_point_model“, welches das Modell der Ladebox beschreibt. Die Informationen über Hersteller und Modell werden verarbeitet, und eine eindeutige „charge_point_id“ wird vergeben. Diese ID dient zur Identifikation der Wallbox im Gesamtsystem und wird als Referenz in Redis und anderen Modulen genutzt.
Beim Aufbau des Demonstrators stellte sich ein Problem mit der Wallbox der Firma Mennekes heraus. Diese sendet als BootNotification einen String mit zu vielen Zeichen, welcher vom Server nicht validiert werden kann. Dieses Problem ist bereits bekannt und wird mit einen Workaround von Seiten des MobilityHouses behoben, vgl. [44]. Die

Validierung wird in diesem Fall einfach ausgesetzt. Die Umsetzung im Quellcode des Servers ist in der Abbildung 20 dargestellt.

```
@on(Action.boot_notification, skip_schema_validation=True)  slorre1
async def on_boot_notification(self, charge_point_vendor, charge_point_model, **kwargs):
```

Abbildung 20: Aussetzung der Validierung, eigene Darstellung

4.5.3.3. *on_Heartbeat*

Um sicherzustellen, dass die Wallbox weiterhin mit dem Server verbunden ist, werden sogenannte *Heartbeats* hin und her geschickt. Dies passiert in einem festgelegten Zeitintervall. Dabei werden jedoch kaum Informationen ausgetauscht, sondern lediglich der Zeitpunkt des *Heartbeats* festgestellt.

4.5.3.4. *on_status_notification*

Die Funktion *on_status_notification* wird im Betrieb der Wallbox genutzt, um beim Aufruf den Status der Wallbox wieder zu spiegeln, hierbei wird der Status pro „Charge Point ID“ und pro „connector-id“ gemeldet. Die „connector-id“ bezeichnet den tatsächlichen Anschluss der Wallbox für den der Status gemeldet werden soll. Die „connector-id“ wird hierbei von der Wallbox übermittelt und ist wichtig um besonders das „Charging Profile“ auf den richtigen Vorgang zu setzen.

4.5.3.5. *On_authorize*

Diese Funktion reagiert auf die Nachricht der Wallbox, wenn diese, wie bei der Mennekes Wallbox, mithilfe eines RFID Tags freigeschaltet und der Ladevorgang damit gestartet wird und autorisiert die sogenannte *id_tag_info*.

4.5.3.6. *trigger_meter_values* und *on_meter_values*

Diese Funktion stellt die Antwort des Servers darauf dar, wenn die Funktion „trigger_meter_values“ aufgerufen wird. Die Funktion „trigger_meter_values“ fragt von der Wallbox Messwerte an, die Wallbox sendet daraufhin eine Antwort. Diese Antwort wird dann mithilfe der Funktion „on_meter_values“ verarbeitet. Im Rahmen dieser Arbeit bedeutet das, dass die Nachricht der Wallbox mithilfe der Hilfsfunktion *parse_meter_values* aufbereitet und dann in den Redis-Store geschrieben wird. Dabei werden die Messwerte unter dem Schlüssel "device:<charge_point_id>:measurements" gespeichert. Dadurch stehen die Messdaten den anderen Modulen, insbesondere der *Central_Node*, zur Verfügung. Die Funktion *trigger_meter_values* wird hierbei periodisch ausgelöst, um Messdaten aktiv von der Wallbox anzufordern. Diese zyklische Abfrage erfolgt in einem separaten Polling-Loop, welcher beim Start des OCPP-Servers als Hintergrundaufgabe initiiert wird. Ziel ist es, auch ohne ein externes Ereignis stets aktuelle Messdaten für den Optimierungslauf zur Verfügung zu stellen.

4.5.3.7. *send_charging_profile*

Mithilfe des Servers können sogenannte „ChargingProfiles“ an die Wallbox gesendet werden, um eine feste Stromvorgabe zu setzen. Dies geschieht in Reaktion auf eine empfangene „optimized_message“, welche im durch die Central_Node generiert wurde. Die Methode „send_charging_profile“ erstellt ein valides OCPP ChargingProfile mit Angabe des maximalen Stroms in Ampere und übergibt dieses mittels eines „SetChargingProfile“-Requests an die Wallbox. Ein valides „ChargingProfile“ ist in Abbildung 21 dargestellt.

```
async def send_charging_profile(self, current_limit):
    profile = {
        "chargingProfileId": 1,
        "stackLevel": 1,
        "chargingProfilePurpose": ChargingProfilePurposeType.tx_profile,
        "chargingProfileKind": ChargingProfileKindType.absolute,
        "chargingSchedule": {
            "chargingRateUnit": "A",
            "chargingSchedulePeriod": [
                {"startPeriod": 0, "limit": current_limit}
            ]
        }
    }
    req = call.SetChargingProfile(connector_id=1, cs_charging_profiles=profile)
    res = await self.call(req)
    logger.info(f"⚙ Sent charging profile to {self.id}, response: {res}")
    return res
```

Abbildung 21: Darstellung eines OCPP konformen Charging Profiles, eigene Darstellung

4.5.3.8. *on_start_transaction* und *on_stop_transaction*

Wird ein Ladevorgang gestartet, so löst dies die Funktion *on_start_transaction* aus, welche auf die Rückantwort der Wallbox an den Server reagiert. Diese setzt den Autorisierungsstatus der Wallbox auf den Wert *accepted* und startet das *meter*, um die geladene Energiemenge zu erfassen.
Analog hierzu ist die Funktion *on_stop_transaction* die Antwort des Servers auf die Beendigung des Ladevorgangs darstellt, auch hier wird der Wert *accepted* gesetzt und das *meter* gestoppt.
Die Kommunikation erfolgt dabei für alle Funktionen über WebSockets. Die verwendeten Nachrichten und Antworten orientieren sich streng am OCPP 1.6 Standard, vgl. [32], [45] und ermöglichen eine Verbindung und Steuerung von realen kommerziellen Wallboxen. Die Zuordnung von Endgeräten erfolgt automatisch anhand ihrer gemeldeten Hersteller-/Modellkombination, welche mithilfe der Funktion *on_boot_notification* ermittelt werden.

4.5.4. Central Node

Das wichtigste Python Modul, welches in dieser Arbeit entwickelt wurde, ist die sogenannte *Central_Node*. In diesem Laufen alle anderen Funktionen und Module der Software zusammen. Die *Central_Node* übernimmt das Starten der Optimierung und verwaltet zu diesem Zweck auch die eingehenden Nachrichten und die enthaltenen Messdaten sowie die eigenen Messdaten des Knotens. Auch der Versand der Messwerte und der Optimierungsergebnisse wird in der *Central_Node* gesteuert. Im Folgenden werden alle dazu nötigen Funktionen beschrieben. Der Ablauf dieses Moduls teilt sich hierbei in drei Teilbereiche auf, welche der Abbildung 22 entnommen werden können.

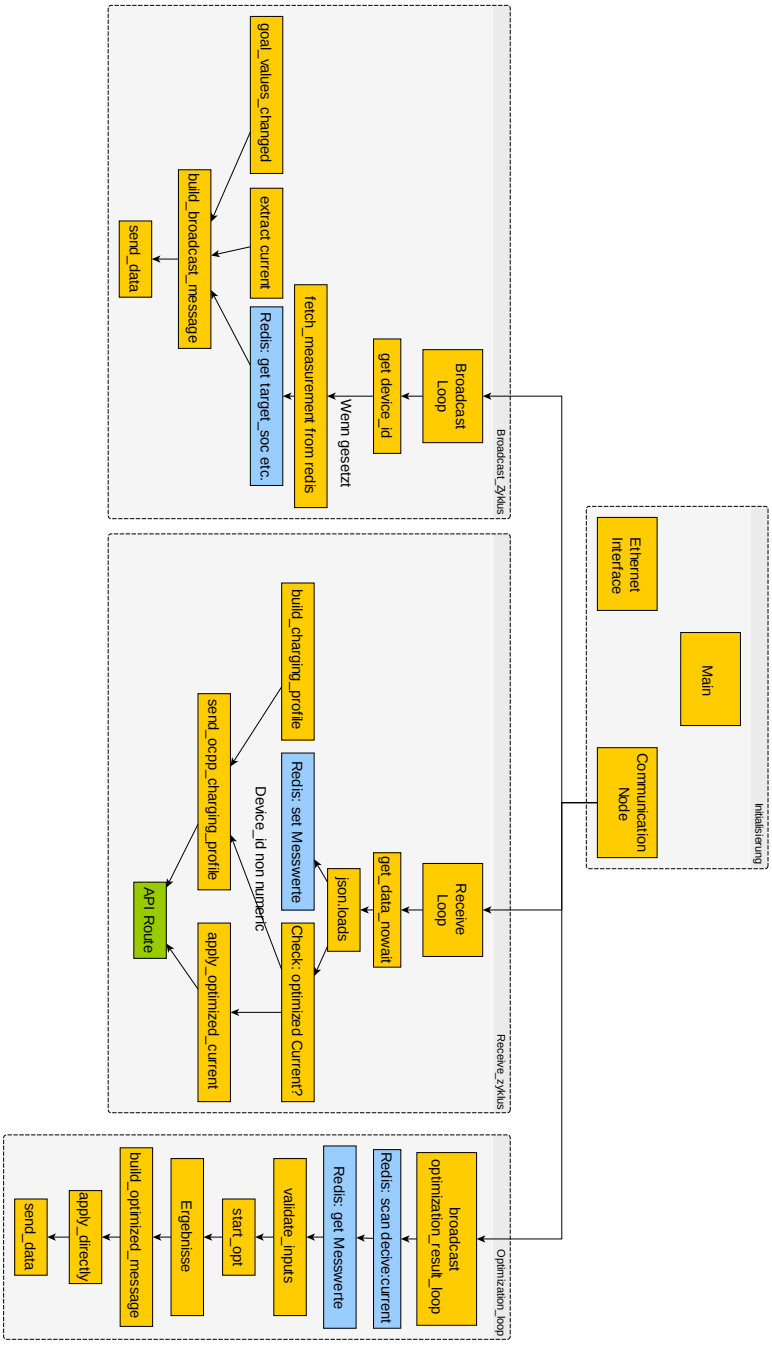

Abbildung 22: Ablaufdiagramm Central_Node, eigene Darstellung.

Die *Central_Node* startet zunächst die Module *EthernetInterface* und *CommunicationNode*, welche für die UDP-Kommunikation im lokalen Netzwerk erforderlich sind. Das Modul *EthernetInterface* wird dabei verwendet, da der Demonstrator im MicroGrid vorerst über das vorhandene Ethernet-Netzwerk betrieben wird.

4.5.4.1. Broadcast-Zyklus

Nach dem Start der *CommunicationNode* verzweigt sich die Funktionalität in drei parallele Hauptprozesse. Der im Diagramm links dargestellte Strang bildet den Ablauf des Broadcast-Zyklus ab, welcher in einem fortlaufenden Loop mit einem Intervall von einer Sekunde ausgeführt wird.
Zunächst wird in diesem Loop die *device_id* ermittelt und als *active_device_id* im Redis-Store gesetzt. Damit ist eindeutig, von welchem Knotenpunkt die kommenden Messwerte stammen. Anschließend werden die zugehörigen Messwerte sowie, falls vorhanden, die Zielwerte aus dem Redis-Store geladen. Hierzu werden die Funktionen *fetch_measurements_from_redis* sowie die Redis-Zugriffe auf Zielgrößen wie *target_soc* oder *departure_time* genutzt.
Daraufhin wird mit der Hilfsfunktion *extract_current* der aktuell gemessene Stromwert des Geräts extrahiert. Dieser wird anschließend im Redis-Store gespeichert und steht später als Eingangsgröße für die Optimierung zur Verfügung.
Eine weitere Hilfsfunktion, *goal_values_changed*, prüft, ob sich die vom Nutzer in der Web-GUI gesetzten Zielwerte seit der letzten Übertragung verändert haben. Ist dies nicht der Fall, werden diese Werte in der nächsten Broadcast-Nachricht nicht erneut mitgesendet, um unnötigen Datenverkehr zu vermeiden.
Abschließend erzeugt die Funktion *build_broadcast_message* die eigentliche UDP-Nachricht, welche alle relevanten Mess- und Zielwerte gemäß den in Kapitel 3 definierten Parametern enthält. Diese Nachricht wird schließlich über die Funktion *send_data* per UDP an die anderen Knoten im Netzwerk gesendet.

4.5.4.2. Receive Zyklus

Der in der Abbildung *22* mittig dargestellte Strang stellt den Empfang der Daten und die darauffolgende Reaktion der Central Node dar. Beim Start der Communication Node wird die Funktion *_start_receiving* gestartet, welche kontinuierlich nach Nachrichten im Netzwerk Ausschau hält, diese analysiert und feststellt, ob die in Kapitel 3 erläuterte „Checksum“ erfüllt ist.
Ist dies der Fall, so wird die empfangene Nachricht in eine sogenannte *Message queue* gestellt. Diese Vorgehensweise stellt sicher, dass keine Nachrichten, welche im Netzwerk gesendet wurden, unbeachtet bleiben. Auf diese *Message queue* greift auch die *Central_Node* mit ihrem *receive_loop* zu. Dieser *receive_loop* ruft die *get_data_nowait* Funktion auf, welche überprüft, ob eine Nachricht vorliegt, ohne die Abfrage zu blockieren. Ist dies der Fall, so wird die Nachricht geladen.
Daraufhin wird untersucht, welche Art von Nachricht empfangen wurde. Ist in der Nachricht das Schlüsselwort *optimized_current* nicht enthalten, so wird der Inhalt der Nachricht als Dictionary im Redis Store gespeichert um, wie oben erklärt in der nächsten

Optimierung mit berücksichtigt zu werden. Ist das Schlüsselwort *optimized_current* enthalten, so wird vom *receive_loop* der Wert für den Strom für die eigene ID ausgelesen.
Aufgrund dieser ID wird auch unterschieden, ob es sich um eine Wallbox mit OCPP-Anbindung oder um ein Gerät mit ModBusTCP Steuerung handelt. Hierzu wird die device_id daraufhin untersucht, ob sie rein numerisch ist, was für ein ModBusTCP Gerät spricht, oder ob sie Buchstaben enthält, was in diesem Fall für eine Wallbox spricht. Hat der *receive_loop* festgestellt, dass am eigenen Knoten eine OCPP-fähige Wallbox angeschlossen ist und angesteuert werden soll, so wird ein Zwischenschritt nötig, um den optimierten Strom für diesen Knoten zu setzen.
OCPP nutzt sogenannte „Charging Profiles“, welche eine festgelegte Payload haben, vgl. Kapitel 4.5.3.7, um von einer Wallbox akzeptiert zu werden. Der optimierte Wert für den Strom wird also in diese Payload integriert, um dann auf die Wallbox gesetzt werden zu können. Ist dies geschehen, wird vom *receive_loop* der API-Endpunkt zum Setzen eines OCCP-Charging Profiles aufgerufen, welcher dann über den integrierten OCPP-Server an die Wallbox weitergegeben wird. Wird das Charging-Profile von der Wallbox akzeptiert, quittiert diese mit der entsprechenden Nachricht.
Stellt der *receive_loop* fest, dass der *optimized_current* für ein ModBusTCP-fähiges Gerät gesetzt werden soll, ist ein weiterer Zwischenschritt nötig. Der *receive_loop* ruft direkt die Funktion *apply_optimized_current* auf, welche derzeit ein Mapping enthält, welches jedem Messgerät das entsprechende steuerbare Gerät am Netzknoten zuweist. Dadurch ist die *device_id* des steuerbaren Gerätes gesetzt. Auch in der Funktion definiert sind die nötigen *selectors*, um den Wert für den Strom auf allen 3 Phasen bzw. auf jedem Wechselrichter an diesem Netzknoten zu setzen. Auch hierbei wird aus den *selectors*, vgl. Kapitel 4.5.1.2, und dem Wert des *optimized_current* einen Payload erstellt. Dieser Payload wird dann mithilfe des API-Endpunkts an das Gerät gesendet. Unterliegend wird hierbei die Funktion *set_value* aufgerufen, welche bei dieser Architektur validiert, ob die genutzten *selectors* die richtige Form haben. Ist dies der Fall, so wird der Strom auf den 3 Phasen gesetzt, vgl. Kapitel 4.5.1.

4.5.4.3. Optimierungszyklus

Der in der Abbildung *22* rechts dargestellte Strang stellt den Ablauf des Optimierungs-Loops dar. Hierbei beginnt dieser Loop damit, dass die *broadcast_optimized_results* Funktion aufgerufen wird, welche zunächst einmal den Redis-Store nach gespeicherten Werten durchsucht. Hierbei sucht die Funktion nach den Dictionaries, die durch den Receive-Loop und den Broadcast-Loop dort gespeichert wurden, um sie in diesem Ablauf der Optimierung zur Verfügung zu stellen.
Die gespeicherten Werte werden vom *broadcast_optimized_results* erst einmal in die *device_inputs* gespeichert. Hierbei ist es so, dass die Optimierung zukünftig als Mindesteingangsgrößen den aktuellen Zeitpunkt, genannt *timestamp* und den *current* also den Strom an dem Knoten benötigt. Daher sind diese beiden Größen in der Kategorie *required* verordnet und müssen bei jedem Aufruf mindestens übergeben werden.
Knoten die nur diese Größen übergeben werden in diesem Fall als nicht steuerbare Lasten interpretiert und sorgen nur dafür, dass ein möglichst vollständiges Gesamtbild des Netzstranges erstellt werden kann. In der *optional* Kategorie sind als Größen noch zusätzlich diejenigen Größen verordnet, welche über das GUI eingetragen wurden. In diesem Fall

sind das der gewünschte Akkustand, *target_soc*, die gewünschte Abfahrtszeit, *departure_time*, der aktuelle Ladestand, *current_soc* und die absolute Kapazität der im Elektrofahrzeug verbauten Batterie.
Diese Größen zielen explizit auf die Optimierung und Steuerung von Ladevorgängen im Netz ab, welche dann in der Optimierung auch als solche behandelt werden. Gibt es keine verfügbaren Werte, so wird die Optimierung mit einem entsprechenden Hinweis übersprungen.
Werden allerdings valide Inputs für die Optimierung festgestellt, wird die *start_opt* Funktion ausgeführt, welche die tatsächliche Optimierung startet. Die Inputs für ein Netzwerk aus zwei Knoten ist der Abbildung *22* zu entnehmen und zeigt die Dateistruktur und die zulässigen Werte für den konkreten Start der Optimierung.
Wenn die Optimierung abgeschlossen ist, so wird ein Dictionary zurückgegeben, welches wieder einen *timestamp* enthält und für alle beteiligten *devices* einen *optimized_current*. Diese Nachricht wird dann mit derselben *send_data* Funktion, welche bereits im *broadcast_loop* beschrieben wurde, versendet. Diese *optimized_message* wird dann wiederum von allen Raspberry Pis im Broadcasting Netzwerk empfangen und entsprechend dem im Absatz zum *receive_loop* beschriebenen Vorgehen verarbeitet und genutzt.

4.5.5. Grafische Oberfläche GUI

Für den Aufbau des Demonstrators im MicroGrid der TH Köln wurde eine einfache und klar strukturierte Grafische Benutzeroberfläche (Graphical User Interface, GUI) erstellt, welche es ermöglicht, die bereits beschriebenen Kundenwünsche einzutragen, um diese somit der Optimierung zugänglich zu machen. Des Weiteren ist es möglich die im MicroGrid integrierten Messgeräte zu abonnieren, um die Messwerte dieser zu erhalten. Das Design der GUI kann der Abbildung 23 entnommen werden.

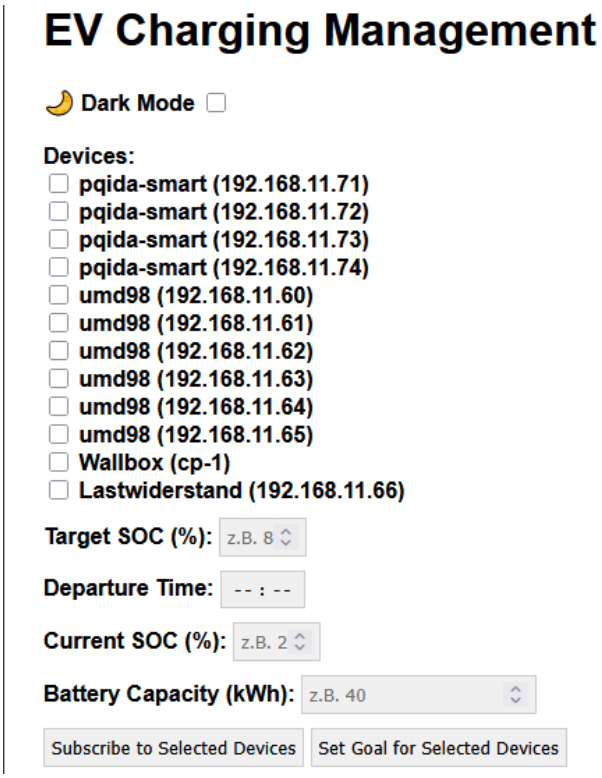

Abbildung 23: Design der GUI, eigene Darstellung.

Wie in der Abbildung 23 ersichtlich, steht im oberen Bereich der GUI ein Auswahl-Menü zur Auswahl des gewünschten Geräts bereit. Unterstützt werden sowohl ModbusTCP-basierte Geräte wie die PQIDA-Messgeräte und UMD98, als auch OCPP-fähige Ladepunkte wie z. B. eine Wallbox mit der ID cp-1. Wird ein Gerät ausgewählt und der Knopf „Subscribe to Selected Device“ gedrückt, so spricht dies den in Kapitel 4.5.2.1 beschriebenen Endpunkt der API an, welche die Ziele in den Redis Store überträgt. Dabei unterscheidet die GUI automatisch, ob es sich um ein PQIDA-Gerät oder ein anderes Gerät handelt, und wählt je nach Gerätetyp die richtigen Selektoren. Für PQIDA-Geräte müssen zusätzlich auch harmonische Oberschwingungen abgefragt werden, um der Struktur der Selektoren gerecht zu werden.

Direkt unterhalb der Gerätauswahl befinden sich Eingabefelder für die Definition eines Ladeziels. Hier können der gewünschte Ziel-Ladezustand (State of Charge, SoC), die geplante Abfahrtszeit, der aktuelle Ladezustand sowie die Batteriekapazität angegeben

werden. Diese Parameter bilden die Grundlage für eine spätere Optimierung der Ladeleistung. Wird der Knopf mit der Beschriftung „Set Goal“ betätigt, so werden die eingegebenen Werte mithilfe des in Kapitel 4.5.2.3 beschriebenen Endpunktes an den Redis Store übertragen.

Im unteren Teil der GUI befindet sich eine Live-Anzeige der Messwerte: Spannung und Strom. Diese Werte werden in regelmäßigen Abständen vom Redis Store abgefragt und aktualisiert dargestellt. Zusätzlich wird ein Diagramm erzeugt, welches eine Leistungsanzeige bietet, siehe Abbildung 24.

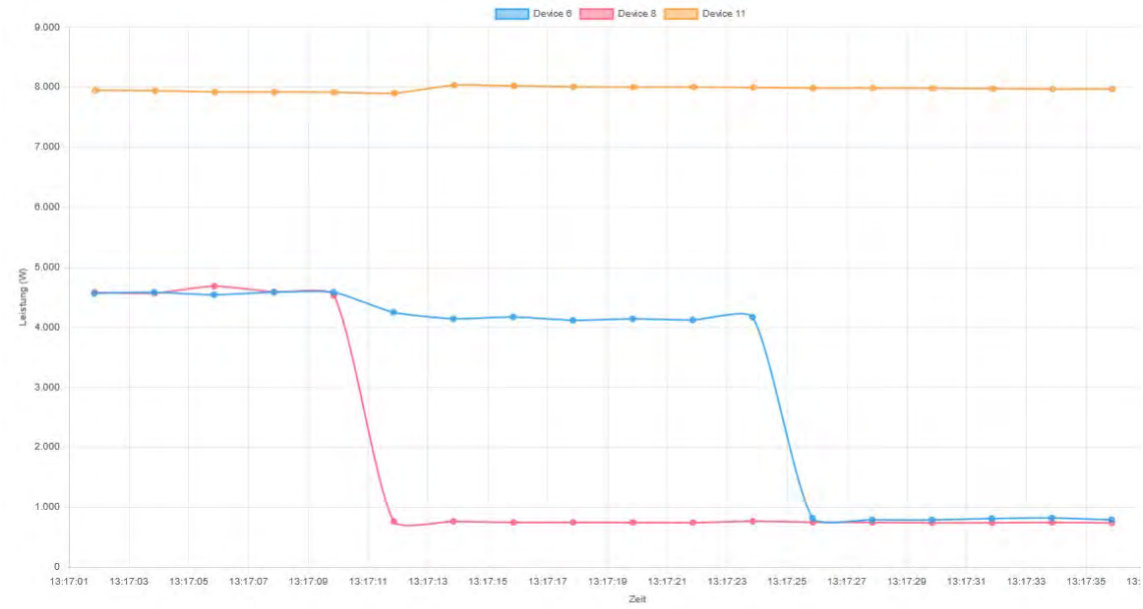

Abbildung 24: Live Leistungsbezugsanzeige der GUI, eigene Darstellung.

Dieses wird kontinuierlich über Chart.js aktualisiert. Hierbei wird die zeitliche Entwicklung der Leistung dargestellt, wodurch sich die Änderungen der bezogenen Leistungen auf einen Blick erkennen lassen. Das Diagramm ist auf 20 Werte begrenzt, um die Darstellung übersichtlich zu halten. Auf der y-Achse des Diagramms ist hierbei die Leistung in [W] angegeben, auf der x-Achse die Uhrzeit. Oberhalb des Graphens sind die Messwerte für Strom und Spannung der jeweiligen Geräte in Zahlenwerten dargestellt, siehe Abbildung *25*.

Voltage (V)	Current (A)
225.08	1.18
222.01	1.12
220.05	12.10

Abbildung 25: Strom und Spannung an den Knoten, eigene Darstellung.

Die gesamte GUI arbeitet hier mit API-Requests, welche auf die in Kapitel 4.5.2 beschriebenen Endpunkte zugreifen. Diese sind als asynchrone Aufrufe umgesetzt. Hierdurch reagiert die Oberfläche schnell auf die User Inputs.

Zusammenfassend stellt die GUI eine leicht bedienbare Oberfläche dar, die sowohl die Eingabe von Ladezielen als auch die Überwachung der Energieflüsse in Echtzeit erlaubt. Sie verbindet klassische Messgeräte über Modbus mit modernen OCPP-Ladepunkten und bietet damit eine universelle Plattform zur Steuerung und Analyse im MicroGrid-Kontext.

4.6. Darstellung der zentralen Funktionen der Software und Vergleich mit dem Konzept der Steuerbox

Abschließend werden nun analog zu den Kapiteln 2.1.2, 2.1.3, 3.1.2 das Steuerungskonzept des Demonstrators, sowie die zentralen Funktionen und die aus der Kombination von ModbusTCP und OCPP resultierenden Protocol Layers dargestellt.

Zunächst ist hierbei das Steuerungskonzept des Demonstrators zu nennen, welches dem Steuerungskonzept 1 der Steuerbox ähnelt. Diese Ähnlichkeit rührt daher, dass an den Raspberry Pi als Ersatz zur Steuerbox, steuerbare Geräte per Ethernet Schnittstelle angebunden werden. Anders als bei der Steuerbox, ist jedoch kein SMGW angeschlossen, sondern es wird mit deinem Messgerät gearbeitet. Dieses liefert zwar Messwerte, jedoch werden die Steuerwerte und Steuerbefehle innerhalb des Raspberry Pis erzeugt und treffen nicht über die Schnittstelle CLS ein. Das Steuerkonzept des Demonstrators ist exemplarisch dargestellt, wobei die zu steuernden Geräte unter dem Punkt Ladeeinrichtung/Wechselrichter zusammengefasst sind, siehe Abbildung 26.

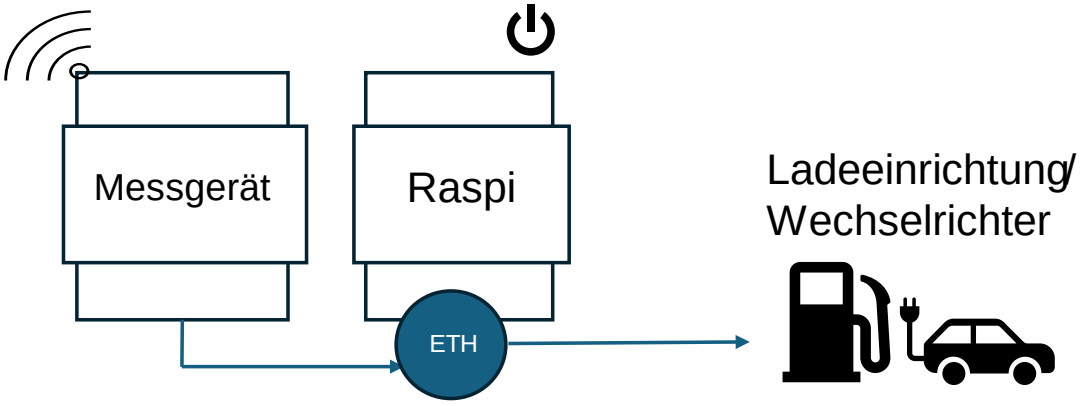

Abbildung 26: Steuerungskonzept Demonstrator, eigene Darstellung

Da bei der Umsetzung des Demonstrators nicht nur auf ein Protokoll zur Ansteuerung der Geräte gesetzt wird, ergibt sich aus der Zusammenfassung der beiden Protokolle Modbus TCP und OCPP ein kombiniertes Protokoll. Aus der Kombination der beiden Protokolle Modbus TCP und OCPP, leitet sich auch eine Erweiterung der Protokoll Layer ab, welche in der Abbildung 27 dargestellt ist.

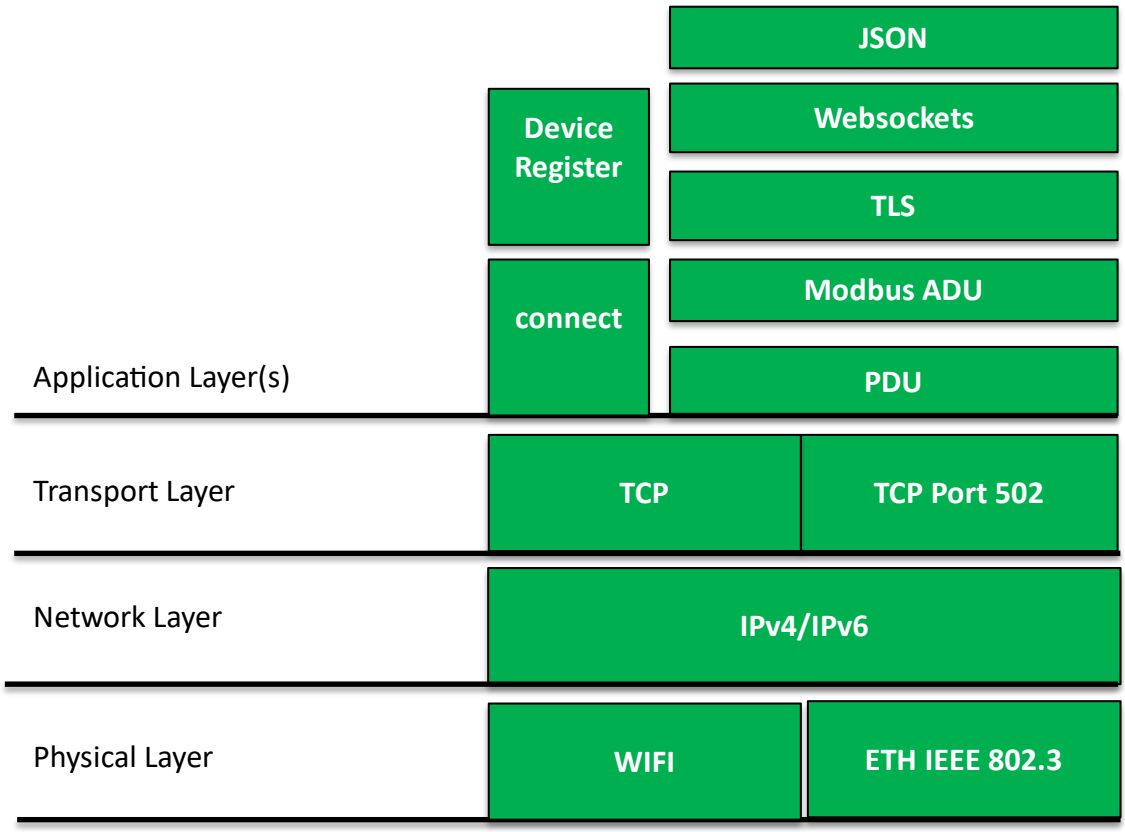

Abbildung 27: Resultierende Protocol Layer aus der Kombination von Modbus TCP und OCPP, eigene Darstellung

Im Network Layer erfolgt die Anbindung des steuerbaren Geräts analog zum Konzept der Steuerbox mithilfe von IPv4 und IPv6, was die Anbindung sowohl mit statischer als auch dynamischer IP-Adressen gewährleistet. Um die Anlagen aufzufinden, bzw. einzubinden, wird für Modbus Geräte die in der Bibliothek *devicecom* entwickelte Funktion *Device_register* verwendet, für die Einbindung von OCPP-Anlagen wird die Funktion *on_connect* verwendet. Die konkreten Einstellungen, um eine Wallbox mit dem OCPP-Server zu verbinden sind in Kapitel 5 dargelegt.
Der eigentliche Datenaustausch erfolgt über das Transportprotokoll TCP für OCPP-Anlagen und mit TCP-Port 502 für Modbus-Anlagen. In der Anwendungsschicht kommen sowohl das WebSocket-Protokoll als auch das Modbus ADU zum Einsatz. Die Nachrichten zwischen dem Raspberry Pi und den OCPP-Anlagen werden dabei im JSON-Format übertragen. Die Nachrichten für die Modbus TCP Anlagen werden hingegen mithilfe des PDU übertragen.

Der in dieser Arbeit entwickelte Demonstrator verfolgt wie bereits erwähnt ähnliche Ziele wie die Steuerbox nach FNN, was sich auch in den zentralen Funktionen der Software widerspiegelt. Wie in den vorherigen Kapiteln beschrieben umfasst die entwickelte Software eine Vielzahl an Funktionen zum Betrieb des Demonstrators, die zentralen Funktionen zur Ansteuerung der im Netz verbundenen Geräte lassen sich jedoch auf vier Funktionen herunterbrechen. Diese sind in Abbildung *28* dargestellt.

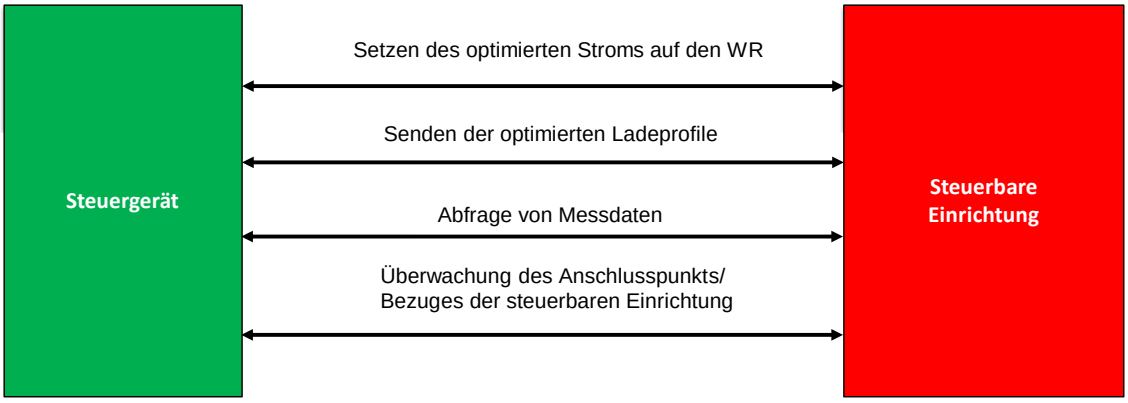

Abbildung 28: Zentrale Funktionen der entwickelten Software, eigene Darstellung.

Die zentralen Funktionen der entwickelten Software beschränken sich zum aktuellen Stand auf das Setzen der optimierten, das Senden von Ladeprofilen an eine OCCP-fähige Wallbox, die Abfrage von Messwerten der jeweiligen Geräte bzw. Messgeräte und somit allgemein die Überwachung des Anschlusspunktes. Diese Funktionen umfassen nur die direkte Kommunikation zwischen den Steuergeräten, in diesem Fall Raspberry Pis und den steuerbaren Einrichtungen, in diesem Fall den Wechselrichtern und der Wallbox. Perspektivisch soll die Ansteuerung der Geräte auch auf EZA ausgeweitet werden.

Diese Funktionen erfüllen im Demonstrator denselben Zweck, wie die Funktionen der Steuerbox, vgl. Abbildung 12, wobei im Falle der hier entwickelten Software noch die Funktionen zur Kommunikation der Steuergeräte untereinander hinzukommen, vgl. Kapitel 4.5.4.

5. Aufbau und Funktionsweise des Demonstrators

Im MicroGrid der TH Köln sind wie bereits erwähnt unterschiedliche Typen von Geräten und Verbindungsarten verbaut. So sind die Messgeräte bereits zweigeteilt in Messgeräte mit der Bezeichnung UMD98, was für Universal Measuring Device steht, und in PQIDA Smart Geräte der Firma A-Eberle. Diese beiden Geräte sind über ModbusTCP ansteuerbar und liefern eine große Bandbreite an Messgrößen.
Für den Betrieb des Demonstrators ist allerdings nur das Auslesen der Spannung und des Stroms von Interesse. Beide Geräte werden grundsätzlich über ihre im Netzwerk des MicroGrids vergebene IP-Adresse verbunden. Hierzu wird die Funktion *device_register* aus der Bibliothek *devicecom* genutzt, welche aus der *device_register.yaml* die Konfiguration, in diesem Fall die IP-Adresse und den konkreten Port zur Verbindung extrahiert und sich mit den Geräten verbindet.
Alle weiteren Befehle werden nun über die Modbus-Register des Gerätes gesendet, dass bedeutet, dass über die TCP-Verbindung eine bestimmte Nachricht gesendet wird, welche eine bestimmte Antwort vom Gerät auslöst. Die Device-Register und die jeweiligen Funktionen können der beigefügten „Modbus-Register“ Datei entnommen werden.
Die nächste Kategorie von Geräten, welche zur Demonstration genutzt und in diesem Fall auch gesteuert werden soll, sind die Wechselrichter der Firma UFE. Diese werden zwar auch über eine TCP-Verbindung eingebunden, jedoch werden sie nicht über Modbus-Register, sondern über sogenannte Hexbytes gesteuert.
Die letzte Kategorie von Geräten, die eingebunden werden soll, ist die der OCPP fähigen Wallboxen. Diese sind einfach einzubinden, da die Verbindung mit dem auf den im Netz integrierten Raspberry Pi Geräten mithilfe eines OCPP-Servers realisiert wird. Um hier eine Verbindung aufzubauen, ist es lediglich nötig, in der Benutzeroberfläche der Wallbox, in diesem Fall der Wallbox der Firma Mennekes, die IP-Adresse und den Port des zu Nutzenden OCPP-Servers einzutragen.
Bei der Wallbox der Firma Mennekes befindet sich die Einstellung für diese Verbindung in der Kategorie Einstellungen -> Netzwerk -> backend. Hier kann die IP-Adresse des Servers mit dem zugehörigen Port in der Form „ws://IP-Adresse:Port“ eingetragen werden.
Befinden sich die beiden Geräte nun in einem Netzwerk wird sich die Wallbox automatisch mit dem OCPP-Server verbinden. Verbindet sich die Wallbox mit dem OCPP-Server, so ist dies auch im Logging der Software zu sehen, vgl. Abbildung 29.

```
2025-06-26 15:54:50 - websockets.server - INFO - connection open
2025-06-26 15:54:50 - OCPP - INFO - 🔌 Neue WebSocket-Verbindung von ('172.18.0.1', 49102)
2025-06-26 15:54:50 - ocpp - INFO - unknown: receive message [2,"c0d41518-1eaf-4450-87d7-639b527a9be0","BootNotification"
{"chargePointModel":"1","chargePointVendor":"cp"}]
2025-06-26 15:54:50 - OCPP - INFO - 🔌 BootNotification von cp-1: Vendor=cp, Model=1
2025-06-26 15:54:50 - ocpp - INFO - cp-1: send [3,"c0d41518-1eaf-4450-87d7-639b527a9be0",{"currentTime":"2025-06-
26T15:54:50.143170","interval":10,"status":"Accepted"}]
```

Abbildung 29: Erfolgreiche Verbindung zwischen Wallbox und OCPP-Server, eigene Darstellung

Für die Präsentation wurde ein Demonstrator aus 3 Raspberry Pis aufgebaut, wobei zwei der Raspberry Pis mit den Wechselrichtergruppen verbunden sind und einer der

Raspberry Pis mit dem Messgerät, das den Strom am Lastwiderstand misst. Das hieraus resultierende Netzwerk ist in Abbildung 30 dargestellt.

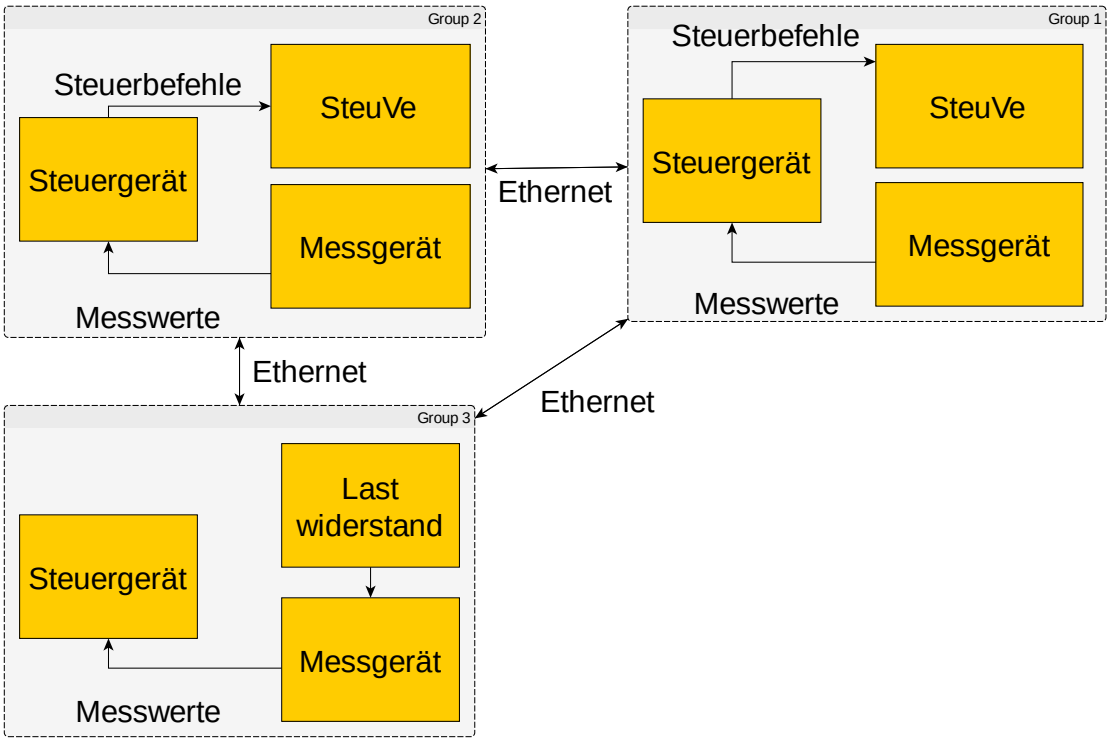

Abbildung 30: Demonstratornetzwerk, eigene Darstellung.

Hierbei wurden gezielt die in Kapitel 4.5 genannten Konfigurationsdateien angepasst, um zu garantieren, dass jeder Raspberry Pi sich nur mit den Geräten an seinem Knoten verbindet. Dies ist wichtig, da nur so eine echte dezentrale Regelung umgesetzt werden kann. Nun werden über die GUI auf dem Raspberry Pi die jeweiligen Messgeräte abonniert, um die Messwerte zu erhalten. Die Software quittiert dies im Logging über einen Code „200 OK“, siehe Abbildung 31.

```
INFO: 172.18.0.1:39924 - "POST /api/device/6/subscribe HTTP/1.1" 200 OK
```

Abbildung 31: Bestätigung des Abonnements der Messwerte, eigene Darstellung.

Analog zum Bezug der Messwerte von den Messgeräten wird wie in Kapitel 4.5.3 beschrieben, können auch die Messwerte von einer Wallbox über die Funktion „Meter Values“ abgefragt werden. Trotzdem wurde die Funktion getestet und verifiziert. Der Ablauf und die ermittelten Daten für die Funktion sind in Abbildung 32 dargestellt. Auch bei dem Bezug der Messwerte direkt von der Wallbox werden diese in Redis gespeichert und stehen somit später zur Verfügung.

```
cp-1: send [2,"e557e23f-39a9-43b0-a501-91bc20ff1302","TriggerMessage",{"requestedMessage":"MeterValues"}]
cp-1: receive message [3,"e557e23f-39a9-43b0-a501-91bc20ff1302",{"status":"Accepted"}]
TriggerMeterValues für cp-1, Antwort: TriggerMessage(status='Accepted')
```

Abbildung 32: Logging zur Funktion "meter_values", eigene Darstellung.

Sobald die Messwerte zur Verfügung stehen, wird der *broadcast_loop* ausgelöst und versendet diese, um sie somit den anderen Knoten zur Verfügung zu stellen. Nachdem die ersten Nachrichten erfolgreich verschickt sind, werden diese auch an den anderen Knoten empfangen, wobei die eingehenden Messwerte zur besseren Verarbeitung in Redis gespeichert werden. Dieser Vorgang ist in Abbildung 33 über das Logging der Software abgebildet.

Abbildung 33: Versand und Empfang der Daten, eigene Darstellung.

Wie bereits im Kapitel 4.5.5 dargestellt können nun auch die Kundenwünsche zum Ladevorgang eingetragen werden. Hierzu werden mit dem bereits beschriebenen Endpunkt /goals/ die Ziel- und Ausgangswerte in den Redis Store gesetzt und darüber auch in die versendeten Nachrichten integriert. Diese Informationen gehen in die Optimierung ein und starten somit einen simulierten Ladevorgang an dem jeweiligen Knoten. Werden die Ziele für den Ladevorgang eingegeben, so quittiert die API dies mit dem in Abbildung 34 dargestellten Code 200.

Abbildung 34: Quittierung der goal Funktion, eigene Darstellung.

Im Falle des Demonstrators wird ein Knoten gewählt, welche die Optimierung für die anderen Knoten ausführen soll. Später soll die Wahl des optimierenden Knotens dynamisch erfolgen. Die Auswahl wird hierbei darüber getroffen, welcher Knoten am besten von allen anderen zu erreichen ist. Erhält nun der Knoten, welcher in der Demonstration dafür ausgewählt wurde, die Optimierung durchzuführen, die Daten für einen oder mehrere Knoten, so wird die Optimierung ausgeführt und die optimierten Ströme für die einzelnen Knoten berechnet.
Ein beispielhaftes Dictionary für ein Gerät mit allen Parametern, welche per UDP-Broadcast übertragen werden und in die Optimierung mit eingehen können ist der Abbildung 35 dargestellt.

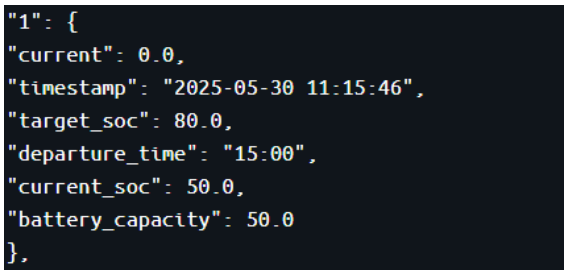

Abbildung 35: Beispielhafte Geräteparameter für die Nachricht und die Optimierung, eigene Darstellung.

In der Demonstration ist hierfür ein einfacher Algorithmus gewählt worden, welcher die benötigte Energiemenge der Ladevorgänge berechnet und darüber den nötigen Strom bestimmt, welcher dann für die jeweiligen Wechselrichtergruppen gesetzt wird. Diese optimierten Ströme werden dann vom optimierenden Knoten in eine Nachricht verpackt, welche jedem Knoten einen optimierten Strom zuweist. Der optimierende Knoten ignoriert seine selbst gesendeten Nachrichten jedoch, weshalb der Strom am optimierenden Knoten direkt beim Versand aus der Nachricht extrahiert wird und über den API-Endpunkt „control“, vgl. 4.5.2.4, gesetzt wird. Der beschriebene Vorgang ist in Abbildung 36 im Logging der Software dargestellt.

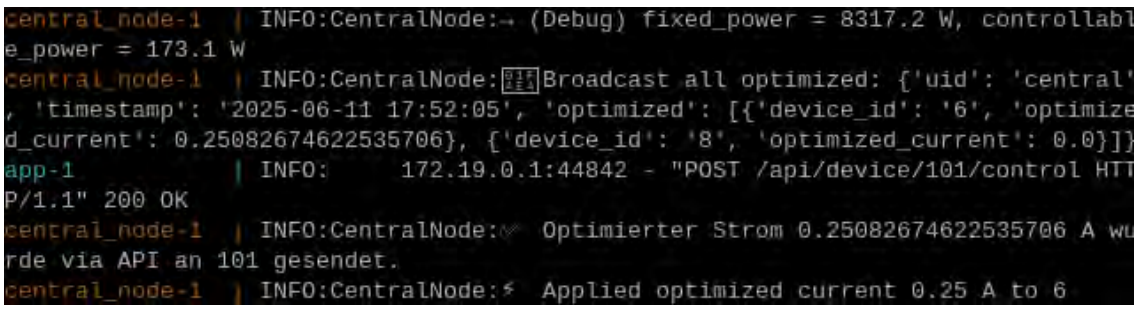

Abbildung 36: Versand der optimierten Ströme und setzen desselben, eigene Darstellung.

An den nicht optimierenden Knoten wird die optimierte Nachricht mit dem *receive_loop* empfangen und der optimierte Strom für den jeweiligen Knoten aus der empfangenen Nachricht extrahiert. Im Anschluss wird geprüft, welche Art von steuerbaren Geräten an dem Knoten angeschlossen sind, vgl. 4.5.4. Wird festgestellt, dass ein ModbusTCP steuerbares Gerät angeschlossen ist, so wird der Strom direkt über den Api Endpunkt „control“, vgl. 4.5.2.4, an das Gerät gesendet.

Wird im *receive_loop* festgestellt, dass am Knoten ein steuerbares Gerät angeschlossen ist, welches anstatt über ModbusTCP über das OCPP gesteuert werden kann, so wird wie in Kapitel 4.5.4 beschrieben zunächst ein OCPP-Charging Profile erstellt, welches dann über den Endpunkt *send-charging*, vgl. 4.5.2.5, an die Wallbox gesendet wird. Ist dies

erfolgreich geschehen, wird die Wallbox eine Bestätigung zurücksenden. Diese Bestätigung ist in der Abbildung 37 dargestellt.

```
ocpp - INFO - cp-1: receive message [3,"9cbb3322-2884-42dd-a451-4f6cbc3dfc11",{"status":"Accepted"}]
OCPP - INFO - SetChargingProfile-Antwort: SetChargingProfile(status='Accepted')
- "POST /api/send-charging/cp-1 HTTP/1.1" 200 OK
```

Abbildung 37: Bestätigung der Annahme des gesendeten Charging Profiles, eigene Darstellung.

Auch die in Kapitel 4.5.2.7 dargestellte Testfunktion konnte mit dem Aufbau des Demonstrators getestet werden. Hierbei wird über die Web-Oberfläche der Fast API eine Device ID eingegeben und der gewünschte Strom gesetzt. Dies ist im Logging der Software ersichtlich siehe Abbildung 38.

```
INFO: 172.18.0.1:42364 - "POST /api/test/apply_current/6 HTTP/1.1" 200 OK
INFO: 127.0.0.1:33670 - "POST /api/device/101/control HTTP/1.1" 200 OK
```

Abbildung 38: Erfolgreiche Ausführung der Testfunktion apply_current, eigene Darstellung.

Diese Funktion wird beim Betrieb des Demonstrators genutzt, um einem der Wechselrichter im MicroGrid einen negativen Ladestrom zuzuweisen, was ein überladen der Batterien im MicroGrid verhindert. Dies ist nicht nur wichtig, um die Batterien zu schonen, sondern auch um die Werte für die Ladevorgänge richtig setzen zu können, da bei einer Überladung der Batterien der Batteriekontroller dafür sorgt, dass die gesetzten Ströme nicht gehalten werden können.

6. Darstellung und Diskussion der Ergebnisse

6.1. Darstellung der Ergebnisse

Nachdem in Kapitel 5 die Funktionsweise der Software im Betrieb beschrieben wurde, muss festgestellt werden, ob die Software tatsächlich auch das Ergebnis erzielt, welches angestrebt wird. Hierzu werden für die Demonstration drei Raspberry Pis genutzt, welche in ein Netzwerk verschaltet werden. Zwei dieser Raspberry Pis werden genutzt, um die Messwerte an den Wechselrichtergruppen aufzunehmen und somit die Ladevorgänge zu überwachen. Ein Raspberry Pi wird genutzt, um die Messwerte am Lastwiderstand zu überwachen. Wie in Kapitel 5 beschrieben, tauschen diese drei Knoten nun ihre Messwerte aus. Das Versenden und Empfangen der Nachrichten kann Abbildung 33 entnommen werden. Werden Daten empfangen, so wird die Optimierung durchgeführt. Dann werden, wie in Kapitel 5 beschrieben zwei Ladevorgänge gestartet. Hierzu werden ein gewünschter Abfahrtszeitpunkt ein Endladestand, der aktuelle Ladestand und die imaginäre Kapazität des Akkus angegeben. Diese Werte fließen nun in die Optimierung ein, welche die Sollstromwerte errechnet. Dies ist nötig um den Ladevorgang wie gewünscht abzuschließen. Die Stromwerte werden für die Wechselrichter gesetzt und die Batterien werden aufgeladen.
Nun wird der Lastwiderstand hinzugeschaltet und die ermittelten Messwerte von diesem Knoten fließen in die Optimierung mit ein und beeinflussen die laufenden Ladevorgänge. Die Ladevorgänge werden infolge der gestiegenen Gesamtlast im System herunter geregelt. Dieses Herunterregeln ist sowohl an den Displays der Messgeräte im MicroGrid als auch in der GUI zu sehen.
Die gemessenen Werte sind in Abbildung 39 grafisch dargestellt. Die Messwerte als Excel Datei sind im digitalen Anhang dieser Arbeit verfügbar.

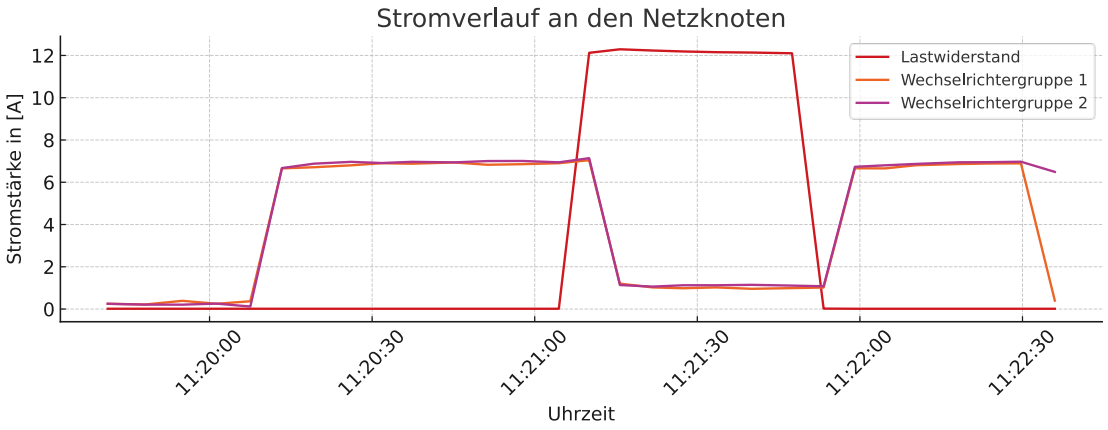

Abbildung 39: Stromverlauf an den Netzknoten in [A], eigene Darstellung

In Abbildung 39 ist auf der x-Achse der Zeitpunkt der Messung aufgetragen, auf der y-Achse des Diagramms ist der aktuelle Strom in Ampere angegeben. Zum Start der Aufzeichnung, also ab 11:20h, ist zu sehen, dass die Ladevorgänge an den beiden Knoten mit den Wechselrichtern gestartet werden. Die Stromwerte an diesen Knoten steigen auf einen Wert von ca. 7 A an. Kurz nach 11:21h wird der Lastwiderstand eingeschaltet, was

zu einem Strom von ca. 12 A an seinem Knoten führt. Die Messwerte von diesem Knoten werden mit in die Optimierung aufgenommen. Diese reagiert darauf, indem die Ladestromwerte für die beiden Wechselrichtergruppen auf einen Wert von ca. 1 A reduziert werden. Die Werte werden so weit reduziert, da die Optimierung nur einen einfachen Algorithmus nutzt, welcher nicht sehr differenziert auf die Änderungen reagieren kann. Wichtig zu sehen ist jedoch, dass die Steuerung der Geräte erfolgreich ist und die Wechselrichter herunter geregelt werden. Kurz vor 11:22h wird der Lastwiderstand ausgeschaltet. Die Optimierung erkennt dies und regelt die Wechselrichter wieder auf den ursprünglichen Wert von ca. 7 A hoch. Ganz zum Schluss ist in der Grafik nur noch die Beendigung der Demonstration zu sehen, weshalb der Strom für das eine Gerät plötzlich abfällt.

6.2. Diskussion der Ergebnisse

Wie in Kapitel 6.1. dargestellt, zeigen die Messwerte des Stroms am Knoten eine erfolgreiche Reaktion der Kommunikationsinfrastruktur im MicroGrid auf das hinzuschalten einer nicht steuerbaren Last. Die genutzte Optimierung ist hierbei nur rudimentär und wird im weiteren Verlauf des Projekts ECS4DRES durch eine umfangreichere Optimierung ersetzt. Jedoch zeigt die Reaktion der beiden steuerbaren Knoten auf die nicht steuerbare Last klar, dass sowohl die Kommunikation zwischen den Knoten über den physical layer als auch die Kommunikation zwischen dem Raspberry Pi und den steuerbaren Geräten erfolgreich umgesetzt wurde. Zur besseren Veranschaulichung der Auslastung des Netzes wird aus diesen Stromwerten der Gleichzeitigkeitsfaktor ermittelt, welcher in der Abbildung 40 dargestellt ist. Die dynamisch ermittelte Gleichzeitigkeit der Lasten lag in diesem Testnetz fast durchgängig um 0,6 was den Werten für kleine Netze durchaus entspricht, vgl. [26]. Zum Zeitpunkt des Einschaltens des Lastwiderstands steigt der Gleichzeitigkeitsfaktor auf etwa 0,9, was deutlich zu hoch ist, sinkt aber im nächsten Schritt, nach der Reaktion der Optimierung wieder auf ca. 0,5 ab. Der Verlauf des Gleichzeitigkeitsfaktors für das Demonstratornetzwerk kann der Abbildung 40 entnommen werden, wobei der Gleichzeitigkeitsfaktor *g* auf der y-Achse der Grafik dargestellt ist und die Uhrzeit wieder auf der x-Achse der Grafik dargestellt ist.

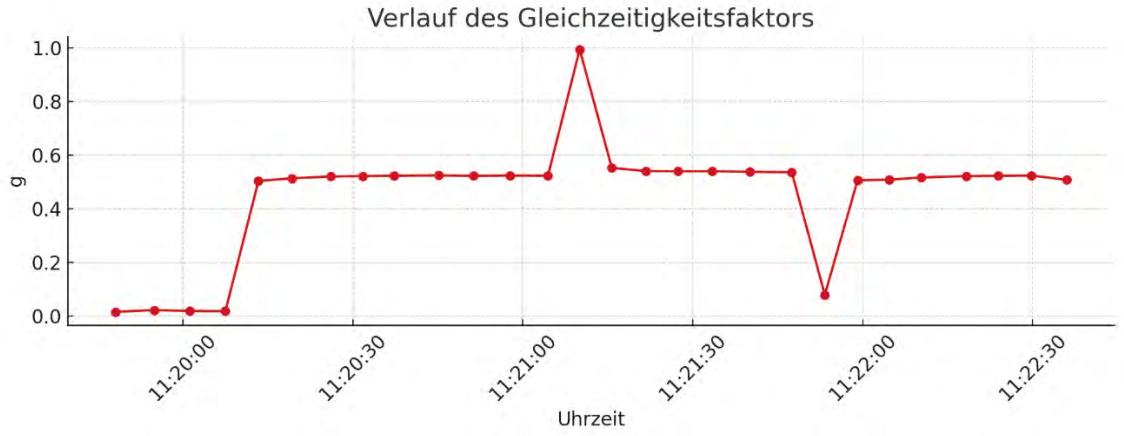

Abbildung 40: Verlauf des Gleichzeitigkeitsfaktors für das Demonstratornetz, eigene Darstellung.

Dadurch, dass das Netzwerk aus den Raspberry Pis kontinuierlich Messwerte austauscht und mit der Optimierung aufsteigende Lasten im Netz reagiert, konnte durch diese Lastverschiebung der Gleichzeitigkeitsfaktor des Testnetzes reduziert werden. Die dezentrale Kommunikation muss im weiteren Verlauf des Projektes noch um weitere physical layer, vgl. Kapitel 3.1.3, und eine größere Anzahl an Netzknoten zur Kommunikation erweitert werden, um die Technologie weiter zu testen. Dies wird in Zukunft besonders wichtig werden, um eine Alternative zur Lösung der Steuerbox FNN zu bieten und besonders für Hauseigentümer mit Home Energy Management System eine Technologie zur Verfügung zu stellen, welche sowohl die Steuerung des Eigenverbrauchs als auch die Netzverträglichkeit der Elektromobilität erhöht.

7. Fazit und Ausblick

Die zentrale Aufgabe dieser Masterarbeit, der Aufbau eines Demonstrators zur dezentralen Regelung von steuerbaren Lasten, wurde erfolgreich abgeschlossen. Dabei zeigt diese Masterarbeit, dass die dezentrale Kommunikation und Regelung von steuerbaren Lasten technisch möglich sind und sich im Labor der TH Köln umsetzen lassen. Es ist möglich ein dezentrales Netz zwischen drei Raspberry Pis aufzubauen, welche Nachrichten austauschten und auf die Lastwechsel im MicroGrid reagierten, vgl. Kapitel 6.1.. Auch war es möglich über die GUI Kundenwünsche einzugeben und somit die Ladevorgänge zu modulieren. Die Ansteuerung der Geräte funktionierte hierbei automatisch als Reaktion auf die eingehenden Nachrichten mit den Optimierungsergebnissen. Der in der Arbeit aufgebaute Demonstrator zeigt, dass eine dezentrale Kommunikation und Steuerung möglich ist und legt somit den Grundstein für eine skalierbare Lösung.

Trotzdem besteht weiterhin Entwicklungsbedarf. Zum einen sollte die GUI dahingehend weiterentwickelt werden, dass sie noch deutlicher herausstellt, dass es sich um eine GUI zum Start von Ladevorgängen für Privatpersonen handelt. Auch die Aufnahme der Messwerte kann dahingehend automatisiert werden, dass diese von den angeschlossenen Geräten direkt abgefragt werden, ohne dass dies manuell bestätigt, werden muss. Auch ein Test mit einem größeren Netzwerk von Raspberry Pis steht derzeit noch aus.
Auch die Einbindung von weiteren Geräten wie zum Beispiel weiteren Wechselrichtern, Wärmepumpen und weiteren Messgeräten über die Schnittstelle Modbus TCP, sowie die Einbindung der OCCP fähigen Wallboxen in den laufenden Betrieb des Demonstrators an sich soll in Zukunft noch umgesetzt werden.

Literatur

[1] Agora Energiewende und Forschungsstelle für Energiewirtschaft e. V., "Haushaltsnahe Flexibilitäten nutzen. Wie Elektrofahrzeuge, Wärmepumpen und Co. die Stromkosten für alle senken können.," 2023.

[2] VDE|FNN, "FNN-Hinweis Netzintegration Elektromobilität," 2019.

[3] V. D. FNN, "VDE FNN Hinweis "Anforderungen an die technische Ausgestaltung der physikalischen und logischen Schnittstellen der Steuerungseinrichtung zum Anschluss und zur Übermittlung des Steuerbefehls an eine steuerbare Verbrauchseinrichtung oder ein Energie-Management-System“," 2025.

[4] Bundesamt für Justiz, *"Energiewirtschaftsgesetz vom 7. Juli 2005 (BGBl. I S. 1970, 3621), das zuletzt durch Artikel 1 des Gesetzes vom 21. Februar 2025 (BGBl. 2025 I Nr. 51) geändert worden ist"*, 2005.

[5] E. Waffenschmidt und I. Stadler, "Swarm Grids for a decentralized power grid control," 2025.

[6] Prof. Dipl.-Ing. Matthias Grandel, Prof. Dipl.-Ing. Volker Wachenfeld, M.A. Claudius Kübler, "Entwicklung eines Demonstrators für priorisiertes Laden (im Förderprojekt: Kundenorientiertes Energiemanagement mit autonomer Lastregelung)," 2024.

[7] Bundesamt für Sicherheit in der Informationstechnik, "TR-03116 Kryptographische Vorgaben für Projekte der Bundesregierung - Teil 3," 2025.

[8] VDE Verband der Elektrotechnik, "Ausprägung der digitalen Schnittstelle an steuerbaren Einrichtungen oder an einem Energie-Management-System," 2024.

[9] Bundesamt für Sicherheit in der Informationstechnik, "Protection Profile for a Smart Meter Gateway (SMGW-PP) - Version 2.0 - 13.12.2024 - Certification-ID: BSI-CC-PP-0073-V2," 2014.

[10] Bundesamt für Sicherheit in der Informationstechnik, "Technische Richtlinie BSI TR-03109-1 - Anforderungen an die Interoperabilität der Kommunikationseinheit eines intelligenten Messsystems - Version 2.0 - 13.12.2024,"

[11] Bundesamt für Sicherheit in der Informationstechnik, "Detailspezifikationen zur TR-03109-5 - Anforderungen an die Interoperabilität eines CLS-Kommunikationsadapters - TR-03109-5 Version 1.0 - Datum:2023-11-24, Commit:aeb06737," 2023.

[12] Bundesnetzagentur, "BK6-22-300, Beschluss vom 27.11.2023,"

[13] EEBus Initiative e.V., "EEBUS Introduction," 2020.

[14] EEBus Initiative e.V., "EEBus Technical Specification Smart Home IP," 2019.

[15] EEBus Initiative e.V., "EEBus SPINE Technical Specification Protocol Specification," 2023.

[16] M. K. S. Cheshire, "Multicast DNS," 2013.

[17] BDEW Bundesverband der Energie- und Wasserwirtschat e.V., "Smart Grids Ampelkonzept Ausgestaltung der gelben Phase," 2015.

[18] elenia Institut für Hochspannungstechnik und Energiesysteme, "Jahresbericht 2022/2023," 2024.

[19] Holder, Ralph und Susanne Frank, "16 Monate unter Strom Abschlussbericht zum NETZlabor E-Mobility-Carré," 2021.

[20] Dr. Selma Lossau und Daniel Wetzel, "DIE E-MOBILITYALLEE Das Stromnetz-Reallabor zur Erforschung des zukünftigen E-Mobility-Alltags," 2019.

[21] Doris Johnsen und Daniel Strommenger, "GESTEUERTES LADEN VON ELEKTROFAHRZEUGEN ÜBER PREISANREIZE Anwendungsbeispiele und Handlungsbedarf KURZSTUDIE," 2020.

[22] BDEW Bundesverband, "Konkretisierung des Ampelkonzepts im Verteilungsnetz," 2017.

[23] Dr. Birgit Haller, Dr. Ole Langniß, Dr. Albrecht Reuter, Nicolas Spengler, Hg. *1,5°Csellsius: Energiewende zellulär - partizipativ - vielfältig umgesetzt*. Stuttgart: C/sells Selbstverlag c/o Dr. Langniß Engergie & Analyse, 2020.
[24] E. Waffenschmidt. "GridMaximizer." Zugriff am: 8. Juni 2025. [Online.] Verfügbar: http://www.100pro-erneuerbare.com/projekte/2024-02_GridMaximizer/GridMaximizer.htm
[25] E. Waffenschmidt, "Stromnetze für Erneuerbare Energien Vorlesung TH-Köln SS 2024 Prof. Dr. Eberhard Waffenschmidt Lastprofile," 2024.
[26] V. D. FNN, "FNN Studie Gleichzeitigkeitsfaktoren," 2021.
[27] 25. Eberhard Waffenschmidt, Markus de Koster, Christian Hotz, Sergej Baum, Ingo Stadler, "Decentralized Grid Control Using Power Grid State Estimation," *CIRED 2023 - 27th International Conference on Electricity Distribution, Rome, Italy*, Paper No. 11419, 2023.
[28] Eberhard Waffenschmidt, Christian Hotz, Sergej Baum, Ingo Stadler, "Swarm Grids: Verteilte Stromnetzsteuerung für verteilte erneuerbare Energieerzeugung," 2022.
[29] Christian Hotz, Sergej Baum, Eberhard Waffenschmidt, Ingo Stadler, "Topology Estimation in Low Voltage Grids Using Wallbox Charging Data Recordings," *CIRED workshop on E-mobility and power distribution systems, Porto, Portugal, 2.-3. June 2022*, Paper no. 1358. [Online]. Verfügbar unter: http://www.100pro-erneuerbare.com/publikationen/2022-06-Hotz-CIRED/Hotz-Estimating_Topologies.htm
[30] Bundesnetzagentur für Elektrizität, Gas, Telekommunikation, Post und Eisenbahnen Referat 615 Marktbeobachtung SMARD und Bundeskartellamt Arbeitsgruppe Energie-Monitoring, "Monitoringbericht 2024," 2025.
[31] M. org, "MODBUS Application Protocol 1 1 b," 2012.
[32] Open Charge Alliance, "Open Charge Point Protocol 1.6," 2017.
[33] cloudflare. "Was ist das User Datagram Protocol." Zugriff am: 10. Juni 2025. [Online.] Verfügbar: https://www.cloudflare.com/de-de/learning/ddos/glossary/user-datagram-protocol-udp/
[34] Salvatore Sanfillipo. "redis." Zugriff am: 9. Juni 2025. [Online.] Verfügbar: https://github.com/redis/redis?tab=readme-ov-file#what-is-redis
[35] Sebastián Ramírez. "FastAPI: FastAPI framework, high performance, easy to learn, fast to code, ready for production." Zugriff am: 9. Juni 2025. [Online.] Verfügbar: https://fastapi.tiangolo.com/
[36] Docker Inc. "dockerdocs: Get Started." Zugriff am: 9. Juni 2025. [Online.] Verfügbar: https://docs.docker.com/get-started/
[37] PyCharm. "Docker." Zugriff am: 10. Juni 2025. [Online.] Verfügbar: https://www.jetbrains.com/help/pycharm/docker.html
[38] Muhamed Kerim Aydin, Paul Jakob Grüterich. "SYP Projekt." Zugriff am: 26. Juni 2025. [Online.] Verfügbar: https://gitlab.nt.fh-koeln.de/gitlab/syp24/team01
[39] Lukas Alexander Mischker, "Broadband powerline and LoRa as a communication protocol for smart metering in a Swarm Grid," 2024.
[40] Johannes Kruse, "Einsatz von Powerline-Communication (PLC) in wechselrichtergestützten Microgrids," 2023.
[41] Leif Terstegen, "Suitability of 5G as a communication protocol for smart meter gateways“, report master project," 2025.
[42] Ahmed Omer, "Entwicklung einer Smart Meter Funktionalität und deren Integration in eine Steuerungsbox für Ladesäule," 2024.
[43] "devicecom github." Zugriff am: 18. Juni 2025. [Online.] Verfügbar: https://github.com/cire-thk/devicecom

[44] "Validating 'maxLength' in schema['properties']['chargePointSerialNumber'] #128." Zugriff am: 9. Juni 2025. [Online.] Verfügbar: https://github.com/mobilityhouse/ocpp/issues/128
[45] Mohamed Alaaser, The Mobility House. "OCPP Git Repository." Zugriff am: 9. Juni 2025. [Online.] Verfügbar: https://github.com/mobilityhouse/ocpp

Anhang

Der Anhang dieser Arbeit ist ausschließlich digital auf der beigefügten CD verfügbar, da es sich um Programmcode handelt.