Schweizerische
Fachschule
TEKO

Walkie-Talkie Pi

Mobile, dezentrale Sprachkommunikation

Diplomarbeit 2025

Auftraggeberschaft:	TEKO Schweizerische Fachschule AG
Autor/Diplomand:	Davide Bossi
Diplomcoach:	Uta Enke
Ort, Datum:	Zürich, 27.06.2025

Walkie-Talkie Pi

Mobile, dezentrale Sprachkommunikation

Autor

Davide Bossi

System Engineer Junior

8152, Opfikon (Glattpark)

+41 79 196 05 42

davide.bossi@vtg.admin.ch

Dozentin

Uta Enke

TEKO Schweizerische Fachschule

uta.enke@edu.teko.ch

Opfikon (Glattpark), Oktober 2025

Ehrenwörtliche Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der im Literaturverzeichnis angegebenen Quellen und Hilfsmittel angefertigt habe.

Die wörtlich oder inhaltlich den im Literaturverzeichnis aufgeführten Quellen und Hilfsmitteln entnommenen Stellen sind in der Arbeit als Zitat bzw. Paraphrase kenntlich gemacht.

Diese Diplomarbeit ist noch nicht veröffentlicht worden. Sie ist somit weder anderen Interessenten zugänglich gemacht noch einer anderen Prüfungsbehörde vorgelegt worden.

Zürich, 26.10.2025

Davide Bossi

Vorwort

Diese Arbeit entstand als Abschlussprojekt der Weiterbildung zum Informatiker. Dieses Dokument beschreibt nicht nur die Ergebnisse, sondern auch die gesammelten Erfahrungen und soll zur eigenen Umsetzung anregen.

Infos für Leserinnen und Leser

- Die Arbeit ist so strukturiert, dass von den Zielen über die Architektur bis zu Umsetzung, Tests und Ausblick ein roter Faden erkennbar bleibt
- Wichtige Befehle, Konfigurationsdateien und Skripte sind im Text erläutert und in den Anhängen zusammengefasst, damit Nachvollziehbarkeit und Wiederholbarkeit gegeben sind
- Abkürzungen werden im Abkürzungsverzeichnis präzisiert und Verweise im Text erleichtern die Orientierung
- Messwerte und Bewertungen sind nachvollziehbar eingeordnet
- Das vollständige Projekt mit allen Skripten und Konfigurationsdateien ist im öffentlichen GitHub-Repository dokumentiert und kann dort eingesehen oder reproduziert werden

Hinweis zum Einsatz von KI und Qualitätssicherung

- Sämtliche in dieser Arbeit verwendeten Skripte und Dateien wurden durch *ChatGPT* erstellt und im Anschluss technisch überprüft sowie in der Praxis getestet
- Alle Texte wurden mit *DeepL Write* auf Rechtschreibung, Wortwahl und Satzstellung geprüft und zusätzlich manuell gegengelesen
- Der Einsatz von KI diente der Effizienzsteigerung

Dank

Mein Dank gilt den betreuenden Personen für wertvolle Rückmeldungen, meinen Kolleginnen und Kollegen für den fachlichen Austausch sowie meinem Umfeld für Geduld und Unterstützung während der intensiven Projektphasen.

Management Summary

Herkömmliche Funkgeräte sind einfach zu bedienen, bieten aber keine Verschlüsselung und keine automatische Netzorganisation. Moderne IP-basierte Lösungen benötigen meistens eine zentrale Infrastruktur wie Router. Diese Arbeit untersucht, wie sichere Sprachkommunikation auch ohne solche Infrastruktur möglich ist.

Ziel war es, ein dezentrales System zu entwickeln, das Sprache über ein Mesh-Netzwerk überträgt. Dafür wurden drei identische Raspberry Pi mit Audioerweiterung und externen WLAN-Adaptern aufgebaut, die sich automatisch verbinden und bei Ausfällen selbst neu organisieren.

Als Software wurde Mumble gewählt, da es eine quelloffene VoIP-Lösung mit integrierter Verschlüsselung bietet. Die grösste Herausforderung lag in der Kombination von den verschiedenen Systemkomponenten zu einem vollständig autonomen System ohne Benutzeroberfläche.

Die Tests zeigen eine stabile Sprachübertragung mit guter Verständlichkeit und geringer Verzögerung. Im Freien konnten Reichweiten bis etwa 60 Meter erreicht werden und der Betrieb über Powerbank war bis zu 24 Stunden möglich.

Die Lösung ist günstig, einfach nachzubauen und nutzt ausschliesslich Open Source Software. Verbesserungen sind bei der Reichweite und einer robusteren Bauweise für den Feldeinsatz vorgesehen.

Inhaltsverzeichnis

1 Projektbeschreibung

1.1 Hintergrund und Motivation

In vielen Situationen, insbesondere in abgelegenen Regionen oder bei Ausfall von Kommunikationsinfrastrukturen, besteht ein Bedarf an alternativen und unabhängigen Kommunikationslösungen. Der klassische Sprechfunk bietet zwar eine gewisse Unabhängigkeit, stösst jedoch in Bezug auf Sicherheit, Reichweite und Flexibilität schnell an seine Grenzen. Gleichzeitig entwickeln sich IP-basierte Technologien rasant weiter und eröffnen neue Möglichkeiten für dezentrale Sprachübertragung.

Kompakte und leistungsfähige Systeme auf Basis von Einplatinencomputer ermöglichen den Aufbau autonomer, netzunabhängiger Kommunikationslösungen zu vergleichsweisen geringen Kosten. In Verbindung mit Mesh Technologien lässt sich ein flexibles System zur Sprachkommunikation entwickeln, welches ohne zentrale Infrastruktur auskommt.

Bisher existieren nur wenige vollständig dokumentierte Umsetzungen solcher Systeme, die den Schwerpunkt auf mobile Einsatzfähigkeit, Sprachkommunikation und Selbstorganisation des Netzwerks legen.

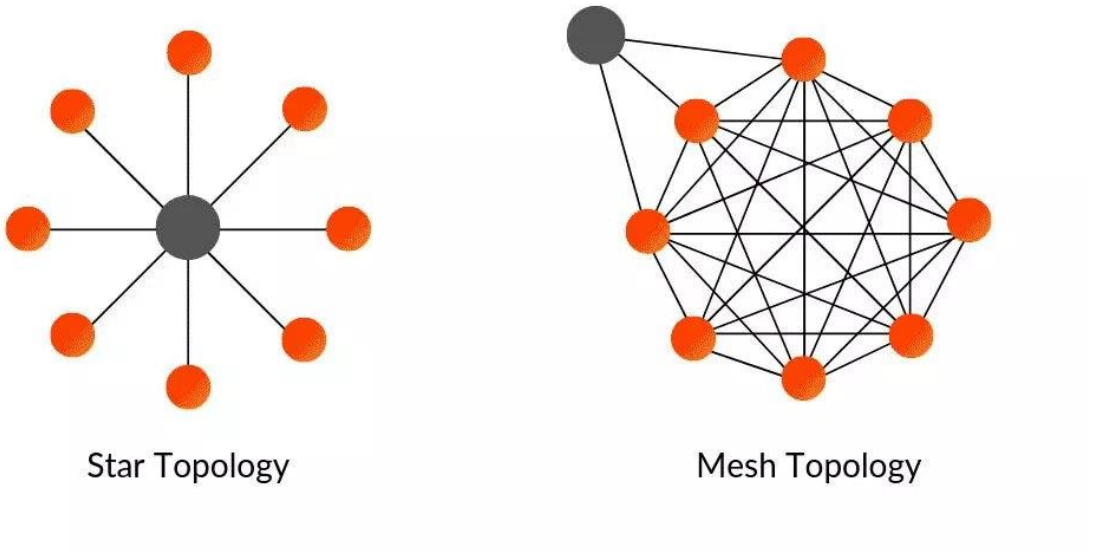

Abb. 1: Vergleich Stern vs. Mesh Topologie[1]

[1] https://dewesoft.com/de/blog/ferndatenerfassung-mit-mesh-netzwerken

1.2 Bedeutung und Zielrichtung des Projekts

Die beschriebene Ausgangslage zeigt, dass bestehende Kommunikationssysteme in kritischen Szenarien häufig nicht den Anforderungen genügen. Insbesondere in abgelegenen Gebieten oder bei Ausfall von Infrastrukturen stehen Anwender oft ohne zuverlässige Kommunikationsmöglichkeit da. Für Organisationen wie Einsatzkräfte, Rettungsdienste oder zivile Hilfsorganisationen kann dies im Ernstfall zu Einschränkungen in der Koordination und Einsatzfähigkeit führen.

Die Abhängigkeit von stabiler Kommunikation verdeutlicht, dass Ausfälle zentraler Infrastrukturen gravierende Folgen für Sicherheit, Koordination und Informationsfluss haben können. Naturkatastrophen und technische Störungen im Telefonnetz führen regelmässig zu temporären Kommunikationsausfällen.

Gleichzeitig sinken die Einstiegshürden für den Einsatz von IP-basierter Kommunikationslösungen durch die kontinuierliche Weiterentwicklung kostengünstiger Hardware und Open Source Software. Die Kombination aus steigenden Anforderungen an Sicherheit und Mobilität sowie der Verfügbarkeit moderner Mesh-Technologien zeigt, dass eine neue, flexible Kommunikationslösung dringend benötigt wird. Dies sowohl für den zivilen Bereich als auch für Einsatzorganisationen[2] mit besonderen Anforderungen an Flexibilität und Unabhängigkeit.

1.3 Rahmenbedingungen und Umfeld der Arbeit

Diese Diplomarbeit entsteht im Rahmen der Weiterausbildung zum Informatiker in Fachrichtung Systemtechnik an der TEKO Zürich. Das Projekt ist eigeninitiiert und behandelt ein praxisnahes, technisch anspruchsvolles Thema aus dem Bereich der drahtlosen Kommunikation.

Durch die Tätigkeit in der Funkkommunikationsausbildung bestehen fundierte Kenntnisse in Netzwerktechnik, IP-Kommunikation und drahtloser Übertragung. Dieses Fachwissen bildet eine solide Grundlage, um Anforderungen und Lösungsansätze zu bewerten.

Da kein externer Auftraggeber beteiligt ist, liegt die Verantwortung für Konzeption, Umsetzung und Dokumentation vollständig beim Diplomanden. Die Arbeit orientiert sich dennoch an realen Anwendungsfällen und könnte in angepasster Form auch für Einsatzkräfte oder zivile Organisationen von praktischem Nutzen sein.

[2] https://www.ar.admin.ch/de/de-software-defined-radio

1.4 Technische Problemstellung

In professionellen Einsatzszenarien basiert die Sprachkommunikation derzeit häufig auf proprietärer bzw. nicht standardisierter Hardware. Diese ist sowohl in der Anschaffung als auch im Unterhalt sehr teuer. Einzelne militärische Geräte Kosten bis zu 25’000 CHF und erfordern oft zusätzliche Software oder Module, um eine verschlüsselte Kommunikation zu ermöglichen. Dies hat nicht nur höhere Investitionskosten zur Folge, sondern auch einen höheren logistischen und technischen Aufwand.

Solche Systeme werden auch vom Militär und weiteren Einsatzkräften in Szenarien mit geringen Sicherheitsanforderungen eingesetzt, etwa bei Übungen oder unkritischen Einsätzen. Dadurch entstehen hohe Betriebskosten, obwohl in diesen Fällen auch günstigere Lösungen ausreichen würden. Im zivilen Bereich, beispielsweise beim Zivilschutz, kommen ebenfalls häufig funkgestützte Systeme auf konventioneller Hardwarebasis zum Einsatz. Diese arbeiten oft unverschlüsselt und sind nur begrenzt erweiterbar. ([01] BABS, 2005)

Forschungsergebnisse zeigen, dass kosteneffektive Mesh-Netzwerke auf COTS-Hardware eine gute Alternative darstellen können. In einer Studie wird beispielsweise der Einsatz von Beartooth MKII Radios auf IoT-Basis zur taktischen Kommunikation mit reduzierten Kosten, geringerem Stromverbrauch und kompakter Bauform beschrieben. ([02] Mekiker, 2023)

Auch die Industrie betont, dass moderne und mobile Mesh-Netzwerke im militärischen Bereich regelmässig für kostengünstige Kommunikation genutzt werden. Dies ist ein deutlicher Unterschied zu teurem Spezialfunk. ([03] Ebbutt, 2021)

Die folgende Abbildung zeigt einen Preis-Leistung Vergleich aktueller Mesh-Funkgeräte Hersteller. Deutlich wird, dass selbst Geräte mit mittelmässiger Performance wie Beartooth noch mehrere Hundert USD kosten, während leistungsstärkere Systeme wie goTenna in einem Preisbereich von über 1'500 USD liegen.

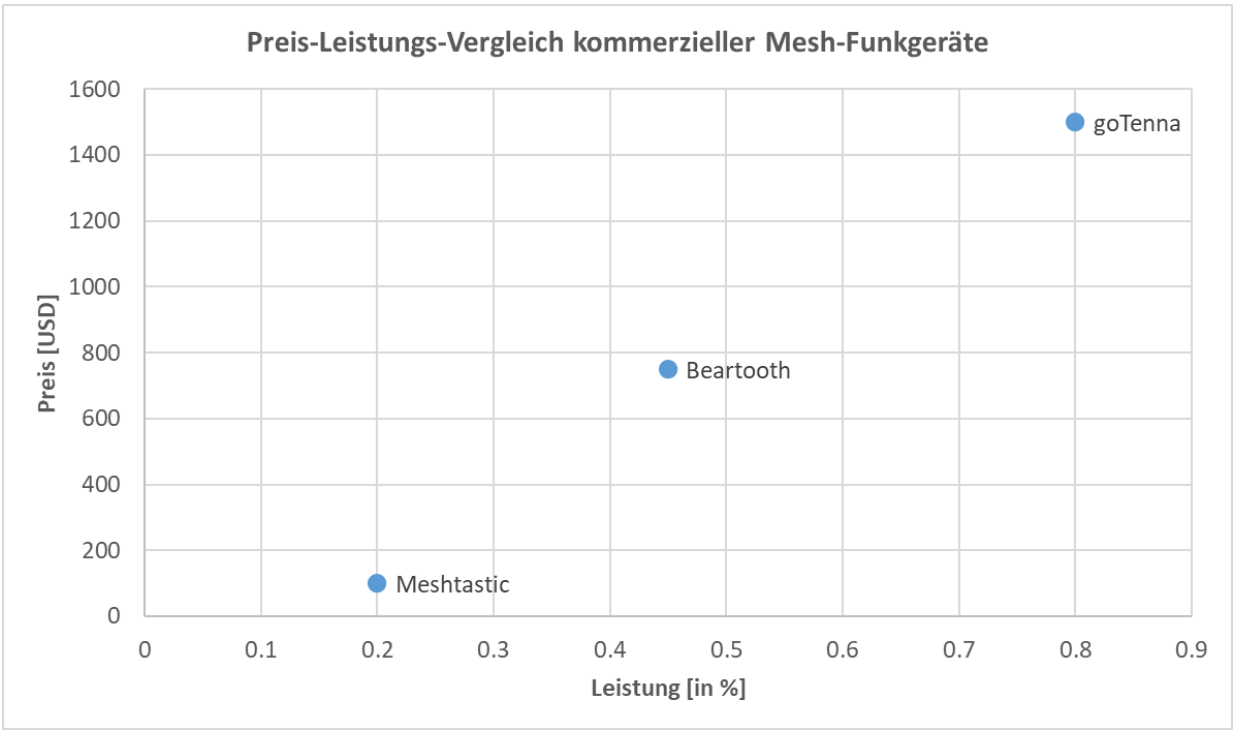

Abb. 2: Preis-Leistung Vergleich Mesh-Funkgeräte[3]

Zusammenfassung des Problems:

- Hohe Kosten und Komplexität aktueller Sprachfunkgeräte
- Unverschlüsselte Kommunikation ohne Zusatzmodule
- Geringe Mobilität und hoher logistischer und technischer Aufwand
- Wissenschaftliche Studien zeigen das Potenzial günstiger Mesh-Lösungen

[3] https://www.youtube.com/watch?v=b8bVSwhYt8U

1.5 Lösungsansatz und Konzeptidee

Das Ziel ist, ein mobiles System für Sprachkommunikation auf Basis eines mobilen Prototyps ohne zentrale Infrastruktur zu entwickeln. Mehrere kompakte Einplatinencomputer verbinden sich per WLAN zu einem Mesh und Sprachübertragung erfolgt über eine VoIP-Software. Jeder Knoten betreibt je einen VoIP-Server und einen VoIP-Client. Fällt ein Server aus, soll der Client automatisch einen erreichbaren alternativen Server im Mesh verwenden.

Für den mobilen Betrieb ist jeder Knoten mit einer Audioeinheit für Mikrofon und Lautsprecher sowie einer mobilen Stromversorgung ausgestattet. Die Steuerung der Sprachübertragung erfolgt per PTT. Alle benötigten Dienste starten nach dem Einschalten automatisch. Eine Internetanbindung ist nicht vorgesehen.

Für das System können verschiedene handelsübliche Einplatinencomputer, geeignete Mesh-Lösungen, eine VoIP-Software sowie Standardkomponenten für Audio und Energieversorgung eingesetzt werden. Welche konkreten Hardware- und Softwarekomponenten verwendet werden, wird erst im Variantenentscheid festgelegt.

Das folgende BPMN-Diagramms zeigt einen möglichen Prozess der Lösungsidee. Von der Inbetriebnahme des Geräts über den Aufbau des Mesh-Netzwerks, die VoIP-Verbindung bis hin zur Sprachübertragung per PTT und dem automatischen Re-Routing bei Knotenausfall. Dieses Diagramm dient als konzeptioneller Leitfaden für die spätere Umsetzung.

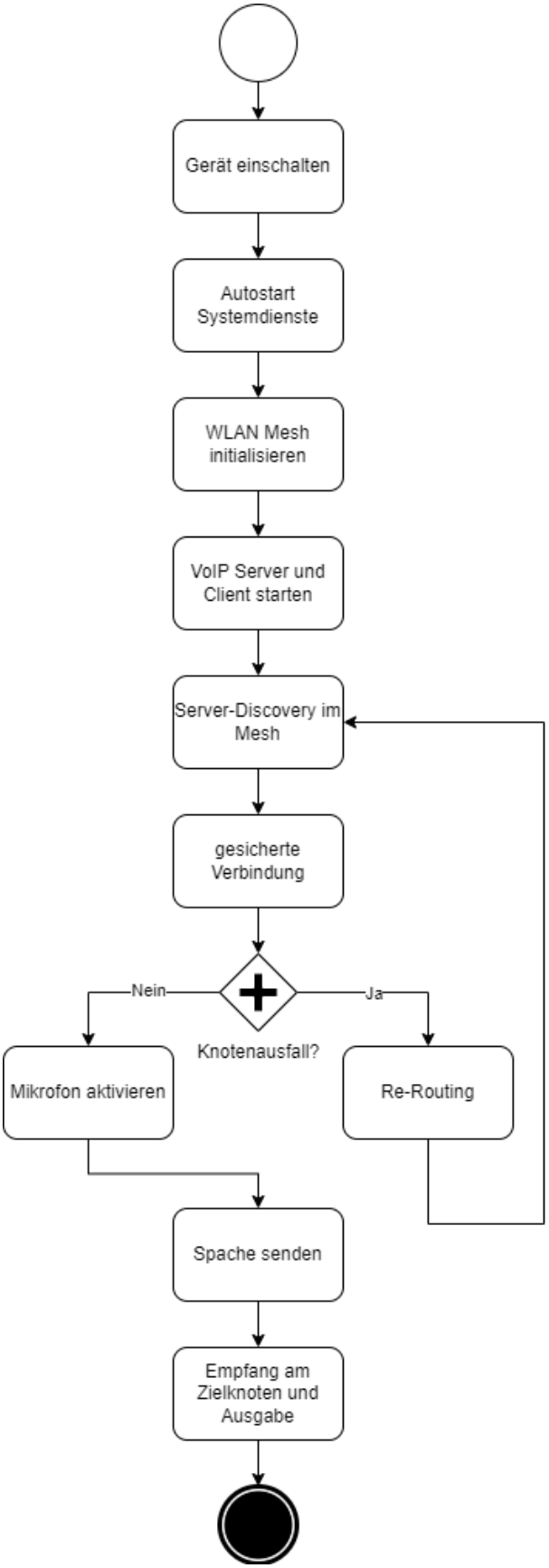

Abb. 3: BPMN eines möglichen Lösungsansatzes

1.6 Projektziele und Erfolgsdefinition

Die wichtigsten Zielsetzungen lassen sich folgendermassen zusammenfassen:

- Kostengünstige Umsetzung mit handelsüblicher HW
- Autarkes Mesh-Netzwerk
- Einfache Sprachkommunikation
- Mobiler Betrieb ohne Display
- Automatische Inbetriebnahme aller notwendigen Dienste
- Redundanz und Ausfallsicherheit

Durch die Umsetzung soll nachgewiesen werden, dass ein solches System eine funktionale und wirtschaftliche Alternative zu bestehenden Lösungen bietet. Der Fokus liegt dabei nicht nur auf der technischen Realisierung, sondern auch auf einer nachvollziehbaren Dokumentation und der Wiederverwendbarkeit des Systems für ähnliche Projekte.

1.7 Erwarteter Nutzen und Anwendungspotenzial

Das entwickelte System soll aufzeigen, wie sich eine Sprachkommunikation ohne zentrale Infrastruktur realisieren lässt. Damit stellt es eine flexible Alternative zu bestehenden, proprietären Funklösungen dar.

Der Aufbau eines dynamischen Netzwerks mit sprachbasierter IP-Kommunikation zeigt auf, wie Netzwerktechnologien in verschiedenen Einsatzszenarien sinnvoll genutzt werden können. Durch den Einsatz von Open Source Software und kostengünstiger Hardware, ist das System auch für zukünftige Anwendungen interessant.

Darüber hinaus kann die entwickelte Lösung als Vorlage für ähnliche Projekte dienen, beispielsweise in sicherheitsrelevanten Bereichen, im Bevölkerungsschutz oder bei temporären Einsätzen ohne Netzabdeckung. Der modulare Aufbau und die ausführliche Dokumentation unterstützen eine Wiederverwendung und Weiterentwicklung durch Dritte.

1.8 Projektgrenzen

Um den Projektumfang klar einzugrenzen und die Umsetzung auf die zentralen Ziele zu fokussieren, werden bewusst folgende Einschränkungen definiert:

Keine Internetanbindung oder externe Netze

Das Mesh-Netzwerk arbeitet eigenständig. Es gibt keine Schnittstelle zu öffentlichen Netzwerken, Cloud-Diensten oder externen Servern.

Begrenzter Testumfang

Die Funktionalität wird mit bis zu drei Einplatinencomputer getestet. Die maximale Reichweite wird praxisnah auf rund 80 Meter beschränkt, inklusive mindestens einem Zwischenknoten (Hop). Tests mit mehr als drei Knoten sind nicht vorgesehen.

Ausschliesslich Standardhardware

Es wird nur handelsübliche Hardware verwendet. Modifikationen an der Hardware sind nicht Bestandteil des Projekts.

Keine Hochverfügbarkeitskonzepte

Der Fokus liegt auf der Bereitstellung einzelner VoIP-Server pro Knoten. Eine automatische Weiterleitung der Clients bei Serverausfall kann konfiguriert werden, eine vollwertige Hochverfügbarkeitslösung wird jedoch nicht umgesetzt.

Keine Benutzeroberfläche

Das System läuft im Headless-Modus ohne Display. Die Bedienung erfolgt automatisch nach dem Einschalten der Geräte, eine grafische Interaktion durch den Benutzer ist nicht erforderlich.

Kein Sicherheitskonzept

Für dieses Projekt wird kein Sicherheitskonzept entwickelt. Die Verantwortung für Sicherheitsmassnahmen liegt bei den potenziellen Weiterentwicklern oder Organisationen, die dieses System in einer produktiven Umgebung einsetzen möchten.

2 Planung

2.1 Pflichtenheft

Das Pflichtenheft bildet die verbindliche Grundlage für die Durchführung des Projekts. Es definiert die Anforderungen, Ziele und Rahmenbedingungen und dient während der gesamten Projektdauer als Referenzdokument. Die Inhalte orientieren sich an der ursprünglichen Themeneingabe, werden jedoch konkretisiert und projektspezifisch angepasst.

2.1.1 Kriterien

2.1.1.1 Sollkriterien

Diese Anforderungen müssen im Rahmen der Diplomarbeit erfüllt werden:

- Aufbau eines vollständig autonomen Mesh-Netzwerks
- Sprachübertragung über das Mesh
- Automatischer Start aller Dienste nach dem Einschalten
- Audio Ein- und Ausgabe über Sound-HAT
- Sprachübertragung über mindestens einen Zwischenknoten
- Stromversorgung über Powerbanks für einen mobilen Betrieb
- Kommunikation ausschliesslich über WLAN-Frequenz, ohne Internet oder Mobilfunk
- Ausführlich dokumentierte Installation, Konfiguration und Inbetriebnahme

2.1.1.2 Wunschkriterien

Diese Funktionen sind nicht zwingend, stellen jedoch sinnvolle Erweiterungen dar:

- Verbindung eines externen Geräts über WLAN-Hotspot eines Raspberry Pi
- Visuelle Anzeige bei Ausfall eines Mesh-Knotens
- Systemverbindung bleibt bei langsamer Bewegung stabil
- Kompatibilität mit alternativer Audiohardware, also ohne HAT
- Erweiterbarkeit durch zusätzliche Sensorik oder GPS (nur konzeptionell)

2.1.1.3 Erfolgskriterien

Zur Bewertung des Projekterfolgs gelten folgende messbare Kriterien:

- Aufbau eines stabilen Mesh-Netzes und Fallback-Funktion auf alternativen Server
- Sprachqualität ist verständlich und zuverlässig
- Vollständig autonomer Betrieb ohne manuelle Bedienung
- Erfolgreiche Tests unter realistischen Bedingungen im Feld

2.1.2 Rahmenbedingungen

Für die Durchführung des Projekts gelten die folgenden Rahmenbedingungen.

2.1.2.1 Technische Rahmenbedingungen

- Es werden ausschliesslich handelsübliche Komponenten verwendet
- Das System basiert auf Open Source Software und weiteren frei verfügbaren Tools
- Die Kommunikation erfolgt über WLAN, ohne Router oder Access Point
- Der Einsatz von kabelgebundener Infrastruktur oder Mobilfunk ist nicht vorgesehen
- Verwendung eines leichtgewichtigen grafischen Benutzerinterfaces für Konfiguration und Tests

2.1.2.2 Organisatorische Rahmenbedingungen

- Eigenständige Durchführung durch den Diplomanden, ohne externen Auftraggeber
- Die Betreuung erfolgt durch einen zugewiesenen Diplomcoach
- Die Diplomarbeit ist öffentlich und darf weiterverwendet werden
- Die komplette Diplomarbeit wird auf **GitHub**[4] zur Verfügung gestellt
- Für das Projekt steht ein Zeitrahmen von Anfang Mai bis Ende Oktober zur Verfügung

2.1.2.3 Umgebungsbedingungen

- Die Tests finden unter praxisnahen Bedingungen statt, z. B. im Freien oder in einem Gebäude mit eingeschränkter Netzabdeckung
- Eine Reichweite von ca. 80 Metern wird als realistisches Einsatzszenario angenommen

[4] https://github.com/kalidavide/Walkie_Talkie_Pi

2.1.3 Produktbeschreibung

Das System besteht aus mehreren mobilen Knoten, die über ein drahtloses Mesh-Netzwerk miteinander kommunizieren. Die Hauptfunktion des Systems ist die IP-basierte Sprachkommunikation zwischen mehreren Geräten ohne den Einsatz einer zentralen Infrastruktur.

Jeder Knoten verfügt über folgende Funktionen:

Selbstorganisierendes Mesh-Netzwerk

Die Knoten verbinden sich automatisch über ein WLAN-Mesh und bilden ein dynamisches, dezentrales Netzwerk.

Sprachkommunikation

Jeder Knoten betreibt einen VoIP-Server als auch einen VoIP-Client. Bei Ausfall eines Servers können die Clients automatisch auf einen erreichbaren alternativen Server im Mesh ausweichen.

Audio-Hardware

Für die Sprachaufnahme und -wiedergabe kommt ein Sound-HAT zum Einsatz. Ein physikalischer Taster steuert die Sprachübertragung und dient als PTT-Funktion.

Autonomer Betrieb

Nach dem Einschalten starten alle erforderlichen Dienste automatisch. Eine Benutzeroberfläche oder manuelle Konfiguration im Betrieb ist nicht erforderlich.

Mobiler Einsatz

Die Stromversorgung erfolgt über handelsübliche Powerbanks. Das System ohne feste Installation mobil einsetzbar.

Fallback Funktion

Das System soll so konzipiert werden, dass es auch bei Ausfall einzelner Komponenten funktionsfähig bleibt. Fällt ein VoIP-Server auf einem Knoten aus, erkennen die Clients dies automatisch und verbinden sich selbständig mit einem alternativen erreichbaren Server innerhalb des Mesh-Netzes.

2.1.4 Projektabwicklung

Die Durchführung des Projekts erfolgt in mehreren Phasen, orientiert am IPERKA-Modell. Jede Phase wird mit Zielen, Aufgaben und Zeitvorgaben bearbeitet.

Informieren:
Recherche zu Mesh-Netzwerken, VoIP-Software, Audio-Hardware und WLAN-Topologien sowie Auswertung vergleichbarer Systeme.

Planung:
Erstellung des Pflichtenhefts, des Zeitplans sowie einer Risikoanalyse. Durchführung einer Analyse der Systemfunktionalität mit Rollen- und Use-Case-Definition.

Entscheiden:
Evaluierung, Auswahl und Beschaffung der finalen Hardware- und Softwarekomponenten. Festlegung der Softwarearchitektur und Netzwerktopologie. Erstellung des technischen Konzepts.

Realisieren:
Aufbau des Mesh-Netzwerks, Installation und Konfiguration von Server- und Client-Dienste sowie Durchführung erster Audioübertragungen. Einrichtung der Autostart-Dienste und Dokumentation der Konfiguration.

Kontrollieren:
Durchführung technischer Tests zu Sprachqualität und Netzwerkstabilität. Reichweitentests mit Zwischenknoten. Dokumentation der Ergebnisse.

Auswerten:
Reflexion der Umsetzung, Analyse von Hindernissen und Verbesserungsmöglichkeiten sowie Bewertung des Einsatzpotenzials. Erstellung der vollständigen Projektdokumentation und Vorbereitung der Präsentation.

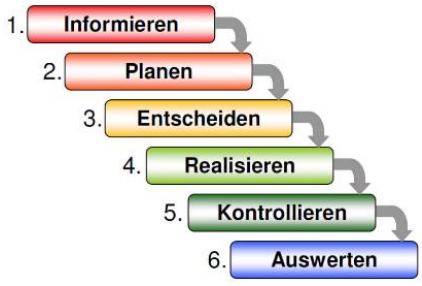

Abb. 4: IPERKA-Prinzip[5]

[5] https://konstrukteur-in.ch/iperka/

2.1.5 Projektrisiken

2.1.5.1 Risikoanalyse

Im Rahmen des Projekts werden potenzielle Risiken identifiziert, die das Systems beeinträchtigen könnten. Jedes Risiko wird mit geeigneten Massnahmen zur Risikominimierung ergänzt.

Pos.	Risiko	Massnahmen
1	RAM des SBC reicht nicht für alle Dienste	Ressourcenüberwachung
2	WLAN-Mesh funktioniert nicht mit integriertem WLAN-Modul des gewählten SBC	Einsatz externer USB WLAN-Adapter
3	VoIP-Client verbindet sich nicht automatisch mit alternativem Server	Automatische Serverumschaltung testen
4	Audio-Hardware inkompatibel	Frühzeitig testen
5	PTT funktioniert nicht zuverlässig	Alternativen prüfen
6	Projektverzögerung durch Hardwareprobleme	Zeitpuffer im Plan, verschiedene Komponenten frühzeitig beschaffen

Tab. 1: Risiken

2.1.5.2 Risikomatrix

Die Risiken werden anhand von zwei Kriterien bewertet und in der nachfolgenden Risiko-Matrix visualisiert:

- Eintrittswahrscheinlichkeit: gering / mittel / hoch
- Auswirkung: gering / mittel / hoch

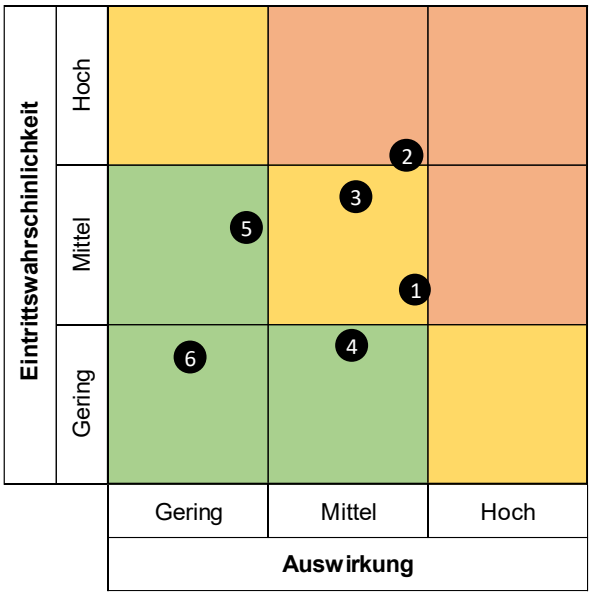

Abb. 5: Risiko-Matrix

2.1.5.3 Annahmen

Für die erfolgreiche Durchführung des Projekts gelten folgende Annahmen:

- Die eingesetzten Einplatinencomputer sind funktionsfähig und mit der gewählten Software und Betriebssystem kompatibel
- Die WLAN-Kommunikation funktioniert zuverlässig
- Der VoIP-Client lässt sich unter dem eingesetzten Betriebssystem mit GUI konfigurieren
- Die Stromversorgung über Powerbanks liefert ausreichend Laufzeit für die Testphasen
- Die Fehlersuche im Feld funktioniert zuverlässig
- Der Projektzeitrahmen bleibt frei von externen, unvorhergesehenen Einflüssen

2.1.6 Abnahmekriterien

Das Projekt wird als erfolgreich abgeschlossen angesehen, wenn…:

- … das entwickelte System im Feld autark betrieben werden kann
- … Sprache zuverlässig über das Mesh-Netzwerk übertragen wird
- … alle erforderlichen Systemdienste automatisch starten, sobald das Gerät eingeschaltet wird
- … die geplanten Funktionstests unter realistischen Bedingungen durchgeführt und dokumentiert werden
- … das komplette Projekt auf GitHub hochgeladen wird

2.2 Zeitplanung

2.2.1 Zeitliche Projektplanung mit Feinkriterien

Für eine termingerechte Durchführung des Projekts wird ein Zeitplan in Form eines Gantt-Diagramms erstellt. Die Planung wird in Microsoft Excel umgesetzt und basiert auf angepassten Vorlagen von Microsoft Create[6]. Dabei werden die Projektphasen nach dem IPERKA-Modell gegliedert und mit Start- und Endterminen versehen.

Die zeitliche Einteilung orientiert sich am Zeitrahmen der Diplomarbeit und berücksichtigt alle Arbeitsschritte von der Informationsbeschaffung über die Umsetzung bis hin zur Dokumentation und Abgabe. Die einzelnen Aufgaben werden in sinnvolle Phasen unterteilt, um Abhängigkeiten und zeitliche Puffer erkennbar zu machen.

Die Koordination und Durchführung der Gespräche mit dem Diplomcoach werden im Zeitplan nicht aufgeführt, da diese dynamisch und je nach Bedarf zwischen Diplomand und Coach abgestimmt werden. Sie finden begleitend zu der Diplomarbeit statt. Eine Übersicht und Zusammenfassung dieser Gespräche ist im **Anhang A** dokumentiert.

Zeitplan Diplomarbeit

Projektname	Walkie-Talkie Pi
Projektstart	05.05.2025
Stand	20.07.2025

Pos	Aufgabe / Tätigkeit	Status	Beginn [Datum]	Endet nach [Tagen]	Ende [Datum]
	Monat				
	KW				30
	Wochentag				Mi
	Datum				23.7
10	**Projektanalyse und Konzeption**		07.07.2025	14	20.07.2025
11	Zieldefinition und Anforderungen	Erledigt	09.07.2025	3	11.07.2025
12	Erstellung eines groben Systemkonzepts	Erledigt	14.07.2025	3	16.07.2025
13	Erstellen des Pflichtenheft	Erledigt	17.07.2025	4	20.07.2025
14	Pflichtenheft fertiggestellt	Erledigt	20.07.2025	0	20.07.2025
15	**Hardwareaufbau und Tests**		21.07.2025	14	03.08.2025
16	Zusammenbau Raspberry Pi mit Soundmodul	Geplant	21.07.2025	2	22.07.2025
17	Konfiguration von Linux und Sound HAT	Geplant	23.07.2025	2	24.07.2025
18	Mikrofon und Lautsprecher testen	Geplant	25.07.2025	2	26.07.2025
19	Stromversorgung prüfen (Powerbank, Spannung)	Geplant	27.07.2025	1	27.07.2025
20	Dokumentation der Hardware	Geplant	28.07.2025	4	31.07.2025
21	Hardware funktioniert vollständig	Geplant	03.08.2025	0	03.08.2025
22	**Softwareintegration**		04.08.2025	7	10.08.2025
23	Mumble-Server aufsetzen	Geplant	04.08.2025	1	04.08.2025
24	Mumble-Client installieren und konfigurieren	Geplant	05.08.2025	2	06.08.2025
25	Taster für Push-to-Talk einrichten	Geplant	07.08.2025	1	07.08.2025
26	Sprachübertragung zwischen Geräten testen	Geplant	08.08.2025	2	09.08.2025
27	Sprachübertragung erfolgreich getestet	Geplant	10.08.2025	0	10.08.2025
28	**WLAN-Mesh Netzwerk**		11.08.2025	14	24.08.2025
29	Einrichtung von batman-adv auf allen Pi	Geplant	11.08.2025	2	12.08.2025
30	Verbindungstest im Mesh	Geplant	13.08.2025	3	15.08.2025
31	Optimierung & Stabilität prüfen	Geplant	16.08.2025	2	17.08.2025

Monat														
KW	30						31							
Wochentag	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So			
Datum	24.7	25.7	26.7	27.7	28.7	29.7	30.7	31.7	1.8	2.8	3.8			

Abb. 6: Auszug Zeitplan Diplomarbeit, Gantt-Diagramm

[6] https://create.microsoft.com/de-de/templates/gantt-diagramme

2.2.2 Wichtige Meilensteine chronologisch

Für den erfolgreichen Verlauf des Projekts werden Meilensteine definiert, die den Fortschritt in abgrenzbaren Etappen strukturieren. Diese Meilensteine dienen als Orientierungspunkte im Projektverlauf und ermöglichen eine Überprüfung von Ergebnissen und Zielerreichung. Sie werden basierend auf der Projektstruktur im Zeitplan festgelegt und mit konkreten Terminen versehen.

In der folgenden Tabelle sind die wichtigsten Meilensteine chronologisch aufgeführt:

Meilenstein	**Termin**
Offizieller Projektstart	07.07.2025
Pflichtenheft fertiggestellt	20.07.2025
Hardware funktioniert vollständig	03.08.2025
Sprachübertragung erfolgreich getestet	10.08.2025
Mesh-Netzwerk läuft stabil	24.08.2025
Benutzerinterface fertiggestellt	07.09.2025
Tests und Optimierung abgeschlossen	21.09.2025
Dokumentation fertig	12.10.2025
Abgabe der Diplomarbeit	29.10.2025
Präsentation der Diplomarbeit	20.11.2025

Tab. 2: Meilensteine

3 Analyse der Systemfunktionalität

Dieses Kapitel beschreibt die funktionalen Anforderungen des geplanten Kommunikationssystems und zeigt auf, welche Aufgaben und Interaktionen innerhalb des Systems vorgesehen sind. Ziel ist es, aus Sicht der Anwender und technischen Akteure zu analysieren, was das System leisten muss und welche Funktionen für den Betrieb wesentlich sind.

Auf dieser Grundlage werden zwei Use Cases abgeleitet, die den praktischen Einsatz im Feld darstellen.

3.1 Funktionsübersicht und Informationssammlung

Das geplante Walkie-Talkie System soll eine mobile Sprachkommunikation zwischen mehreren Knoten ermöglichen, ohne dass eine zentrale Infrastruktur benötigt wird.

Die Geräte kommunizieren über ein selbstorganisierendes WLAN-Mesh Netzwerk. Jede Einheit fungiert gleichzeitig als Sender, Empfänger und Router.

Für die Umsetzung sind folgende Hauptfunktionen erforderlich:

- Autonomer Systemstart
- Selbstheilendes Mesh-Netzwerk
- Sprachkommunikation über VoIP
- PTT-Funktion
- Fallback-Mechanismus
- Mobiler Betrieb

3.2 Rollen im System

Im Gesamtsystem treten zwei Akteure mit unterschiedlichen Aufgaben und Anforderungen auf.

Rolle	Beschreibung	Hauptanforderungen
Funker (Aktor 1)	Bedient ein mobiles Gerät zur Sprachübertragung im Feld	Einfache Bedienung, stabile Verbindung, PTT-Steuerung, mobile Energieversorgung
Empfänger (Aktor 2)	Nimmt Sprachübertragungen entgegen, beobachtet den Netzwerkstatus	Störungsfreier Empfang, Zuverlässigkeit, optionale Sprachübertragung

Tab. 3: Rollen im System

Diese Rollen bilden die Grundlage für die folgenden funktionalen Use Cases.

3.3 Use Case 1 | Funker (mobiler Akteur)

Ziel:

Sichere Sprachkommunikation im Feld über das dezentrale Mesh-Netz mit möglichst geringem Bedienaufwand.

Akteur:

Mobiler Funker mit einem Funkgerät, Lautsprecher und PTT-Taste.

Voraussetzungen:

- Gerät ist mit Strom versorgt
- Alle erforderlichen Dienste starten automatisch
- Mesh-Netzwerk ist aktiv und mindestens ein VoIP-Server erreichbar

Ablauf:

- Funker schaltet das Gerät ein
- Das Gerät initialisiert das Mesh und verbindet sich automatisch mit den anderen Knoten
- Der VoIP-Client startet und verbindet sich mit dem VoIP-Server
- Über eine PTT-Taste aktiviert der Funker das Mikrofon und spricht
- Die Sprachdaten werden digitalisiert, verschlüsselt und über das Mesh gesendet
- Beim Loslassen der Taste wird das Mikrofon deaktiviert
- Fällt der aktive Server aus, verbindet sich der Client automatisch mit einem Ersatzserver
- Nach Wiederverfügbarkeit erfolgt die automatische Rückschaltung auf den Primärserver

Ergebnis:

Der Funker kann zuverlässig und ohne manuelle Konfiguration mit anderen Teilnehmern im Mesh kommunizieren.

3.4 Use Case 2 | Empfänger (stationärer, teilmobiler Akteur)

Ziel:

Dauerhafte Überwachung und Empfang von Sprachübertragungen im Mesh-Netz, optional auch eigene Sprachübertragung.

Akteur:

Stationärer Knoten, z. B. in einem Kommandoposten oder Fahrzeug

Voraussetzungen:

- Gerät eingeschaltet und mit dem Mesh verbunden
- VoIP-Client aktiv, VoIP-Server erreichbar
- Lautsprecher und Mikrofon korrekt konfiguriert

Ablauf:

- Gerät startet automatisch und tritt dem Mesh-Netz bei
- Der VoIP-Client verbindet sich mit dem aktiven Server
- Eingehende Sprachübertragungen werden direkt über den Lautsprecher ausgegeben
- Bei Netzunterbruch oder Knotenausfall wechselt der Client selbstständig zum nächsten erreichbaren Server
- Der Empfänger kann optional selbst sprechen, indem er die PTT-Taste betätigt
- System überwacht fortlaufend die Verbindungsqualität und Audioverfügbarkeit

Ergebnis:

Kontinuierlicher, störungsfreier Empfang aller übertragenen Sprachdaten mit optionaler Rücksprachemöglichkeit.

3.5 Zusammenfassung

Die beiden dargestellten Use Cases beschreiben die wichtigsten Anwendungsfälle des geplanten Kommunikationssystems und zeigen, welche Funktionen für die Umsetzung erforderlich sind.

Beide Akteure greifen auf dieselbe technische Basis zu, unterscheiden sich jedoch in ihrer Nutzung und in der Interaktionshäufigkeit mit dem System.

Diese Szenarien bilden die Grundlage für die anschliessenden möglichen Varianten und die Auswahl der konkreten Hardware- und Softwarekomponenten.

4 Vergleich möglicher Varianten und technischer Komponenten

Auf Grundlage der in Kapitel 3 beschriebenen Systemfunktionen und Use Cases werden in diesem Kapitel mögliche technische Realisierungsvarianten untersucht und miteinander verglichen.

Während das vorherige Kapitel die funktionalen Anforderungen und Abläufe aus Sicht der Anwender definierte, erfolgt nun die Betrachtung der technologischen Umsetzung. Ziel ist es, die geeignetste Kombination aus Hardware- und Softwarekomponenten zu ermitteln, die die im Pflichtenheft und in den Use Cases definierten Anforderungen erfüllt.

Im Fokus stehen dabei ausschliesslich Einplatinencomputer, die jeweils mit unterschiedlichen Kommunikationsmodulen kombiniert werden. Diese Geräteklasse bietet ein Verhältnis zwischen Leistungsfähigkeit, Energieverbrauch und Erweiterbarkeit und eignet sich damit besonders für den mobilen Betrieb im Feld.

Dazu werden drei grundlegende Ansätze zur Realisierung des Kommunikationssystems analysiert, die sich im Bezug auf Übertragungstechnologie, Energieverbrauch, Komplexität und Kosten unterscheiden. Anschliessend erfolgt eine Bewertung dieser Varianten, aus der die Entscheidung für die Umsetzung des Prototyps abgeleitet wird.

Variante 1	Variante 2	Variante 3
		
SDR-Modul	**LoRa-Modul**	**WLAN**

Abb. 7: Übersicht der betrachteten Systemvarianten

4.1 Vergleich möglicher Varianten

4.1.1 Variante 1: Einplatinencomputer mit SDR-Funkmodul

Ein möglicher Ansatz basiert auf leistungsfähigen Einplatinencomputern, die mit SDR-Modulen wie dem HackRF One ([12] Ossmann, 2025) oder dem CaribouLite kombiniert werden. Diese Module arbeiten als klassische Transceiver und ermöglichen die Sprachübertragung über definierte Funkfrequenzen. Mithilfe geeigneter SDR-Software lassen sich Modulation, Verschlüsselung und Kanalauswahl flexibel anpassen.

Im dargestellten Szenario übernehmen die SDR-Module die eigentliche Funkübertragung, während der Einplatinencomputer die Steuerung, Signalverarbeitung und Audiointegration übernimmt. Die Stromversorgung erfolgt mobil über Powerbanks, sodass ein Einsatz im Feld möglich wäre.

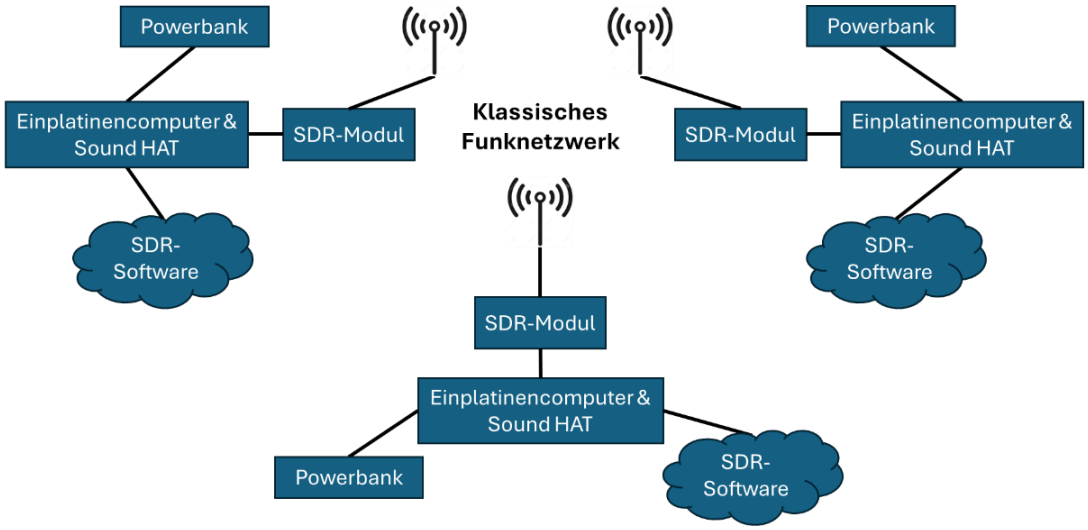

Abb. 8: Variante 1 mit SDR-Funkmodul

Vorteile:

- Hohe Flexibilität bei Frequenzwahl, Protokollen und Verschlüsselung
- Professionelle Entwicklungsumgebungen wie GNU-Radio verfügbar
- Direkter Zugang zu Funktechnologien für Forschung und Ausbildung

Nachteile:

- Hoher Energieverbrauch
- Hohe Kosten pro Gerät
- Komplexe Implementierung
- Regulatorische Einschränkungen: Die Nutzung von Frequenzen in der Schweiz unterliegt den gesetzlichen Vorgaben des Bundesamts für Kommunikation (BAKOM)

4.1.2 Variante 2: LoRa basiertes Mesh-Netzwerk

Ein weiterer Ansatz ist der Einsatz von Low Power Wide Area Technologien wie LoRa. Mehrere Knoten werden mit LoRa-Modulen ausgestattet und bilden ein Mesh-Netzwerk. LoRa bietet grosse Reichweite und geringen Energieverbrauch[7], ist jedoch stark in der Datenrate begrenzt. Sprachübertragung wäre nur mit starker Kompression möglich.

Im dargestellten Szenario übernehmen die Einplatinencomputer die Sprachverarbeitung, während die LoRa-Module für die eigentliche Übertragung zuständig sind. Der Betrieb erfolgt ebenfalls mobil über Powerbanks.

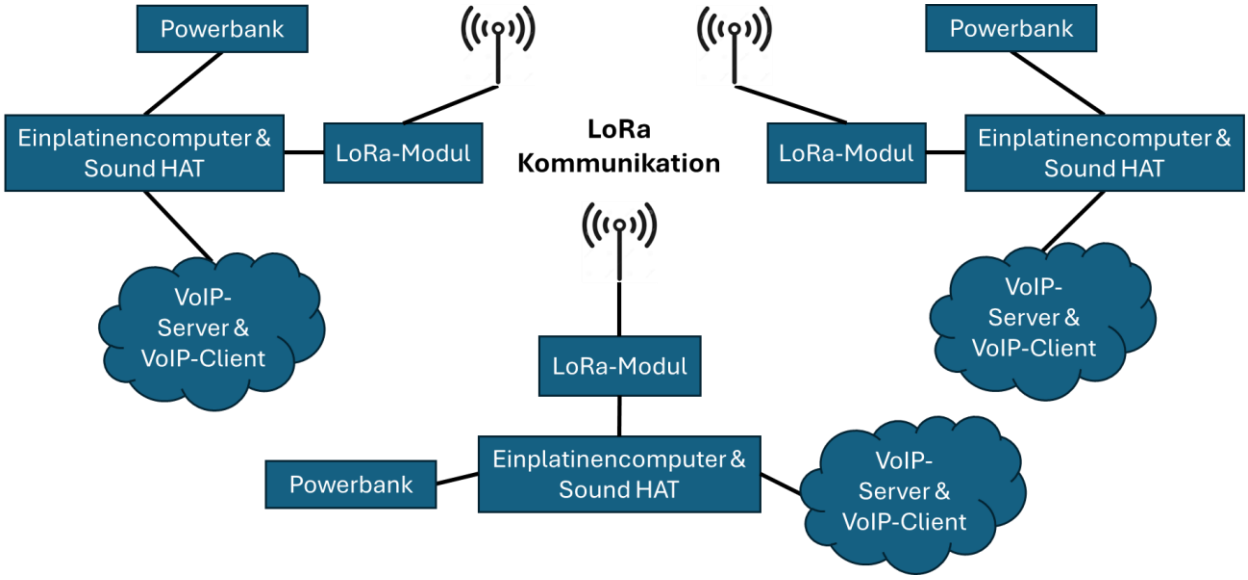

Abb. 9: Variante 2 mit LoRa-Funkmodul

Vorteile:

- Sehr grosse Reichweite, mehrere Kilometer auch ohne Zwischenknoten
- Niedriger Energieverbrauch, dadurch lange Laufzeiten mit kleinen Akkus
- Hohe Robustheit auch in schwierigem Gelände oder bei Hindernissen

Nachteile:

- Sehr niedrige Datenrate, nur bedingt für Sprachkommunikation geeignet
- Eingeschränkte Sprachqualität und erhöhte Latenzzeiten
- Zusätzlicher Implementierungsaufwand für Sprachcodierung, Kompression und Protokollierung

[7] https://www.waveshare.com/sx1262-868m-lora-hat.htm

4.1.3 Variante 3: Einplatinencomputer mit WLAN-Mesh

Beim dritten Ansatz kommen handelsübliche Einplatinenrechner zum Einsatz, die über WLAN zu einem Mesh-Netzwerk verbunden sind. Die Sprachübertragung erfolgt IP-basiert über eine VoIP-Software. Audioeinheiten mit PTT-Funktion ermöglichen eine einfache Bedienung.

Im dargestellten Szenario werden mehrere SBC über WLAN direkt miteinander verbunden. Jeder Knoten betreibt sowohl einen VoIP-Server als auch einen VoIP-Client. Die Stromversorgung erfolgt mobil über Powerbanks, sodass das gesamte System ohne externe Infrastruktur funktionsfähig bleibt.

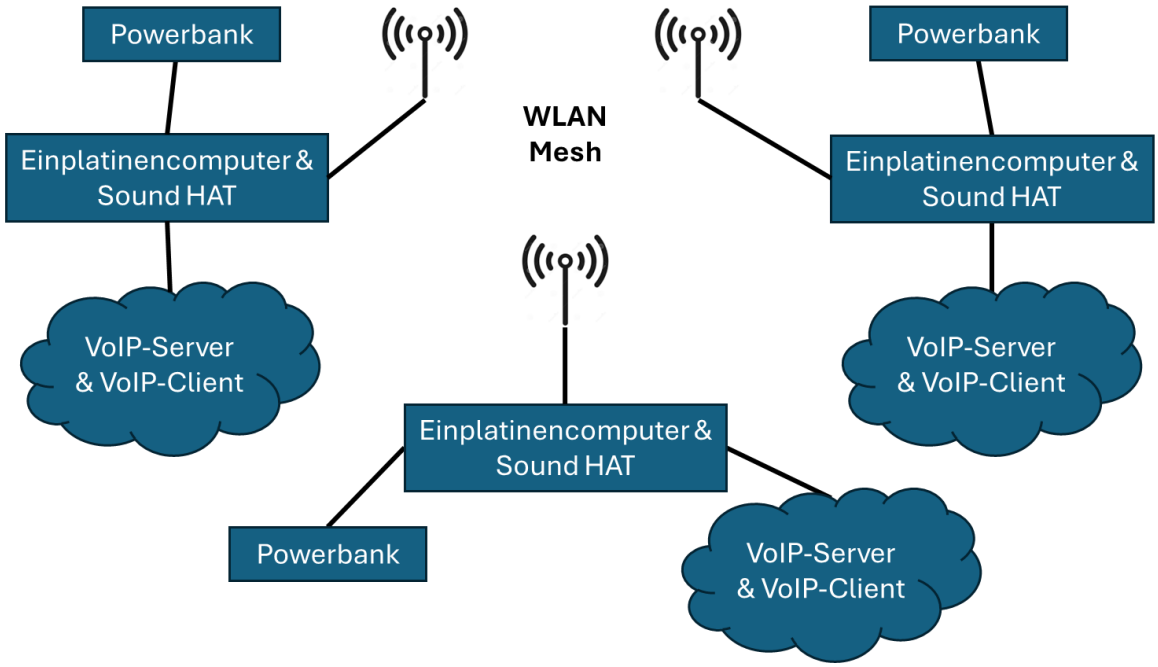

Abb. 10: Variante 3 mit WLAN-Mesh

Vorteile:

- Sehr kostengünstig durch Einsatz handelsüblicher Hardware
- Flexibel erweiterbar durch Hinzufügen weiterer Mesh-Knoten
- Verschlüsselte Kommunikation mit VoIP-Software
- Open Source Software ermöglicht Anpassung und Weiterentwicklung

Nachteile:

- Begrenzte Reichweite im Vergleich zu klassischen Funktechnologien wie LoRa oder SDR
- Erhöhter Konfigurationsaufwand, da mehrere Dienste eingerichtet werden müssen
- Energiebedarf etwas höher als bei Low Power Technologien wie LoRa

4.2 Variantenentscheid mit Begründung

Die drei Varianten werden hinsichtlich Kosten, Energieverbrauch, Komplexität und Sprachqualität bewertet.

Kriterium	SDR-Modul	LoRa	WLAN-Mesh
Kosten und Aufwand	hoch	mittel	niedrig
Reichweite	mittel	sehr hoch	mittel
Sprachqualität	hoch	gering	hoch
Energiebedarf	hoch	sehr niedrig	niedrig
Komplexität	sehr hoch	mittel	mittel
Echtzeit-Sprache	gut	unzureichend	sehr gut

Tab. 4: Bewertung der Varianten nach technischen und funktionalen Kriterien

Bewertung:

- *Variante 1* ist technisch interessant, aber zu komplex und energieintensiv
- *Variante 2* überzeugt durch Reichweite und Energieeffizienz, ist jedoch für Echtzeit Sprache ungeeignet
- *Variante 3* bietet den besten Kompromiss zwischen Sprachqualität, Kosten, Mobilität und Umsetzbarkeit

Entscheid

Für die Realisierung des Prototyps wird **Variante 3 mit WLAN-Mesh** gewählt.

Diese Variante erfüllt alle definierten Soll-Kriterien aus der Systemfunktionalität, lässt sich mit frei verfügbarer Open Source Software umsetzen und ist praxistauglich für mobile Einsätze.

4.3 Hardware

4.3.1 Vergleich Einplatinencomputer

Für die Umsetzung werden verschiedene Einplatinencomputer, sogenannte SBC, evaluiert.

Zentrale Kriterien sind dabei:

- Leistungsfähigkeit wie CPU, RAM
- Stromverbrauch und Wärmeentwicklung
- Grösse und Gewicht
- Kompatibilität mit Audio-Hardware und Mesh-WLAN
- Kompatibilität mit Linux und Open Source SW
- Verfügbarkeit und Preis

Diese Modelle werden in Betracht gezogen und in der folgenden Tabelle untereinander verglichen:

- Raspberry Pi Zero 2 WH[8]
- Banana Pi BPI-M4 Zero[9]
- Radxa Zero 3W[10]
- Raspberry Pi 4B[11]
- Orange Pi Zero 2[12]

[8] https://www.raspberrypi.com/products/raspberry-pi-zero-2-w/

[9] https://banana-pi.eu/produit/banana-pi-bpi-m4-zero/

[10] https://radxa.com/products/zeros/zero3w

[11] https://www.raspberrypi.com/products/raspberry-pi-4-model-b/

[12] http://www.orangepi.org/html/hardWare/computerAndMicrocontrollers/details/Orange-Pi-Zero-2W.html

Modell	CPU	RAM	WLAN	Verfügbarkeit	Stromverbrauch	Preis (CHF)	Besonderheit
Raspberry Pi Zero 2 WH	Quad-Core ARM Cortex-A53 1 GHz	512 MB	2.4 GHz (OnBoard)	hoch (pi-shop.ch, berrybase.ch)	sehr gering	ca. 20.–	Sehr kompakt, günstig, offiziell unterstützt
Banana Pi BPI-M4 Zero	Quad-Core Cortex-A53 1.5 GHz	1 GB	2.4 / 5 GHz	mittel (banana-pi.eu, aliexpress)	gering bis mittel	ca. 30.–	Normaler HDMI, USB-C
Radxa Zero 3W	Quad-Core Cortex-A55 bis 2 GHz	1 bis 8 GB	WiFi 6 2.4 / 5 GHz	mittel bis gering (digikey, mouser)	mittel	ca. 40.–	Sehr leistungsstark, aber schwer erhältlich
Raspberry Pi 4B	Quad-Core bis 1.8 GHz	2 bis 8 GB	2.4 / 5 GHz	hoch (pi-shop.ch, berrybase.ch)	hoch	ca. 50 bis 90.–	Sehr leistungsfähig, energieintensiv
Orange Pi Zero 2	Quad-Core 1.5 GHz	1 bis 4 GB	2.4 GHz	mittel (aliexpress)	gering bis mittel	ca. 30.–	Kompatibilität nicht durchgehend getestet

Tab. 5: Vergleich von SBC

Legende zur Verfügbarkeit:

Hoch: direkt bei Schweizer oder EU-Händlern gut verfügbar

Mittel: verfügbar, aber mit längeren Lieferzeiten oder begrenzten Quellen

Gering: schwer oder nur über asiatische Plattformen mit unklarem Support

Bewertung

Der **Raspberry Pi Zero 2 WH** bleibt die bevorzugte Wahl. Er bietet eine kompakte, stromsparende Plattform mit hervorragender Softwarekompatibilität, breiter Community-Unterstützung sowie nahtloser Integration von Mesh-Protokollen und Audio-Hardware. Die Pinbelegung ist dokumentiert und unterstützt den Betrieb gängiger Sound-HATs ohne zusätzliche Anpassungen.

Der **Banana Pi BPI-M4 Zero** ist eine interessante Alternative. Er bietet eine höhere CPU-Taktfrequenz, moderne Schnittstellen wie USB-C und HDMI und mehr Arbeitsspeicher. Jedoch ist die Linux-Unterstützung insbesondere im Bereich Audio-HAT und GPIO-Kommunikation weniger ausgereift. Auch die Nutzung von Mesh-Protokollen ist nicht dokumentiert.

Der **Radxa Zero 3W** bietet mit bis zu 4 GB RAM und WiFi 6 eine sehr leistungsfähige Plattform. Er ist jedoch eher auf Performance-Anwendungen ausgelegt. Aufgrund eingeschränkter Softwareunterstützung ist mit einem deutlich höheren Integrationsaufwand zu rechnen.

Der **Raspberry Pi 4B** ist zwar die leistungsstärkste Variante im Vergleich, jedoch für ein tragbares, energieeffizientes System ungeeignet. Der deutlich höhere Stromverbrauch, die grosse Bauform und die Notwendigkeit einer aktiven Kühlung sprechen gegen seinen Einsatz im mobilen Betrieb.

Der **Orange Pi Zero 2** wirkt auf den ersten Blick vielversprechend, zeigt jedoch Schwächen in der Treiberunterstützung, insbesondere mit Audio-Hardware und bei Netzwerkprotokollen. Dies könnte zu Stabilitätsproblemen und Inkompatibilitäten führen.

4.3.2 Vergleich Audio-Hardware

Für die Sprachübertragung über einen VoIP-Client ist die Auswahl einer geeigneten Audio-Hardware entscheidend. Es werden mehrere Sound-HATs und externe Lösungen untersucht, die unterschiedliche Vor- und Nachteile aufweisen.

Zentrale Kriterien sind dabei:

- Unterstützung von Duplex-Betrieb
- PTT-Kompatibilität
- Treiberunterstützung unter gängigen OS
- Schnittstellen und Anschlüsse
- Grösse und Mobilität
- Verfügbarkeit und Preis

Die folgenden Audiomodule werden in Betracht gezogen und in der Tabelle untereinander verglichen:

- Waveshare WM8960 Audio HAT[13]
- Seeed Studio ReSpeaker 2-Mics HAT[14]
- Adafruit Voice Bonnet -Two Speakers + Two Mics[15]
- USB-Soundkarte (Beispiel vom Hersteller CREATIVE)[16]

[13] https://www.waveshare.com/wm8960-audio-hat.htm

[14] https://www.seeedstudio.com/ReSpeaker-2-Mics-Pi-HAT.html

[15] https://www.adafruit.com/product/4757

[16] https://de.creative.com/p/sound-blaster/sound-blasterx-g1

Modell	Audio In / Out	Physischer Anschluss	Kompatibilität mit gängigen OS	Mobil tauglich	Verfügbarkeit	Preis (CHF)	Bemerkungen
Waveshare WM8960 Audio HAT	2 x Mikrofon Stereo In / Out	40-Pin GPIO (HAT)	sehr gut	Ja	hoch	ca. 19.00	Integrierter Taster, gute Doku durch Community, I2S-Protokoll
Seeed Studio Re-Speaker 2-Mics HAT	2 x Mikrofon Stereo In / Out	40-Pin GPIO (HAT)	sehr gut	Ja	mittel	ca. 25.00	Lautsprecher müssen separat bestellt werden
Adafruit Voice Bonnet	Stereo In / Out	40-Pin GPIO (HAT)	gut	Ja	mittel	ca. 30.00	Sehr kompakt, teurer als Konkurrenz
CREATIVE Sound BlasterX G1 (USB)	Stereo In / Out	USB-A	Plug & Play	Ja	hoch	ca. 24.00	Gute Qualität, aber grösser und keine Möglichkeit mit GPIO zu arbeiten

Tab. 6: Vergleich Audio HW

Legende zur Verfügbarkeit:

Hoch: direkt bei Schweizer oder EU-Händlern gut verfügbar

Mittel: verfügbar, aber mit längeren Lieferzeiten oder begrenzten Quellen

Bewertung

Der **Waveshare WM8960 Audio HAT** bietet sowohl Audioeingang als auch -ausgang in Stereoqualität und lässt sich direkt über den 40-Pin GPIO mit dem RPi verbinden. Durch den integrierten Taster lässt sich die PTT-Funktionalität ohne zusätzliche externe Hardware umsetzen.

Der **Seeed Studio ReSpeaker 2-Mics HAT** ist ebenfalls ein solides Audio-Modul, das speziell für Sprachprojekte konzipiert wurde. Es verfügt über zwei hochwertige MEMS-Mikrofone und nutzt ebenfalls den GPIO-Anschluss. Im Vergleich zum Waveshare HAT ist dieses Modell bei ähnlichen Funktionen etwas teurer im Einkauf.

Das **Adafruit Voice Bonnet** vereint zwei Mikrofone und drei LED's in einem kompakten Format und wird ebenfalls über den 40-Pin GPIO angeschlossen. Die Lösung ist sehr platzsparend, jedoch in der Community noch wenig dokumentiert. Für einfache Projekte geeignet, jedoch nicht für das angestrebte Ziel geeignet.

Die **CREATIVE Sound BlasterX G1** ist eine USB-Soundkarte, die Plug & Play unter Linux funktioniert und eine gute Audioqualität bietet. Aufgrund ihrer Grösse, dem höheren Energieverbrauch und der fehlenden GPIO-Anbindung ist sie jedoch primär für stationäre Tests geeignet. In einem mobilen Setup mit RPi stellt sie keine ideale Lösung dar, insbesondere da keine integrierte PTT-Steuerung möglich ist.

4.3.3 Entscheidung und Begründung

Die Entscheidung fällt auf den **Raspberry Pi Zero 2 WH** in Kombination mit dem **Waveshare WM8960 Audio HAT**. Diese Konfiguration erfüllt alle Anforderungen hinsichtlich Leistung, Energieverbrauch, Mobilität und Kompatibilität und bietet gleichzeitig ein ausgezeichnetes Preis-Leistungs-Verhältnis.

Ein bewusster Entscheid wird getroffen, keine Variantenbewertung für zusätzliche Hardwarekomponenten wie WLAN-Adapter, microSD-Karten, Powerbanks oder OTG-Adapter vorzunehmen. Diese Komponenten sind standardisierte, handelsübliche Zubehörteile, deren Unterschiede für die Systemfunktionalität unerheblich sind.

Die Auswahl erfolgt pragmatisch nach Verfügbarkeit, Preis-Leistung und bisherigen Erfahrungswerten. Eine detaillierte Variantenanalyse bietet hier keinen technischen Mehrwert und wäre für den Projektzweck nicht zielführend.

4.4 Software

4.4.1 Vergleich Betriebssysteme

Nach der Wahl der Hardware wird auf den Vergleich verschiedener Linux-Distributionen verzichtet. Stattdessen werden ausschliesslich der verschiedenen Betriebssysteme von **Raspberry Pi OS**[17] betrachtet.

Folgende Varianten werden in die Vorüberlegungen einbezogen:

Variante	**Architektur**	**Desktopum-gebung**	**Ressour-cenbedarf**	**Kompatibilität**
Raspberry Pi OS Lite	32-bit	Keine (Headless)	sehr gering	Volle Kompatibilität
Raspberry Pi OS Lite	64-bit	Keine (Headless)	gering	Nicht geeignet aufgrund der Architektur
Raspberry Pi OS mit Desktop	32-bit	Pixel GUI	gering bis mittel	geeignet für Setup und Debugging
Raspberry Pi OS mit Desktop	64-bit	Pixel GUI	hoch	Nicht empfohlen für Pi Zero 2 WH

Tab. 7: Vergleich RPi OS

Bewertung

Aus dem Vergleich wird deutlich, dass die Varianten des Raspberry Pi OS unterschiedliche Schwerpunkte setzen:

- Die Lite-Versionen zeichnen sich durch ihren geringen Ressourcenbedarf aus und eignen sich besonders für headless Anwendungen
- Die Desktop-Varianten bieten durch die grafische Benutzeroberfläche zusätzlichen Komfort bei Einrichtung, Diagnose und Debugging, erfordern jedoch deutlich mehr Rechenleistung
- Hinsichtlich der Architektur zeigt sich, dass 32-bit-Versionen eine bessere Kompatibilität mit der eingesetzten Hardware bieten, während 64-bit-Varianten nur eingeschränkt empfohlen werden

[17] https://www.raspberrypi.com/software/operating-systems/

4.4.2 Vergleich Mesh-Protokolle

Für den Aufbau eines dynamischen, dezentralen Mesh-Netzwerks auf Basis von WLAN werden zwei Protokolle eingehend evaluiert:

- batman-adv (Better Approach To Mobile Adhoc Networking – advanced)
- OLSR (Optimized Link State Routing)

Diese Auswahl zeigt zwei unterschiedliche Ansätze: Layer-2-Routing (Ethernet-basiert) und Layer-3-Routing (IP-basiert). Die Entscheidung für diese beiden Varianten erfolgt bewusst, da sie sich in Architektur, Routing-Verhalten und Konfigurationsaufwand grundlegend unterscheiden und jeweils in bekannten Mesh-Systemen zum Einsatz kommen.

Merkmal	**batman-adv**[18]	**OLSR**[19]
Netzwerkschicht (OSI-Modell)	Layer 2	Layer 3
Konvergenzverhalten	sehr schnell, automatisch	mittel (IP-Routingtabellen erforderlich)
Konfiguration	Einfache Einrichtung	Manuelle IP-Adressierung notwendig
Debugging / Logging	batctl	Komplexere Routingübersicht mit olsrd
Verfügbarkeit unter Raspberry Pi OS	Kernelmodul vorinstalliert	als Paket installierbar
Ressourcenverbrauch	gering	gering bis mittel

Tab. 8: Vergleich Mesh-Protokolle

Bewertung

Der Vergleich zeigt, dass batman-adv und OLSR unterschiedliche technische Konzepte und Einsatzschwerpunkte verfolgen:

- batman-adv integriert sich auf Layer 2 in bestehende Netzwerke und ermöglicht eine automatische Erkennung und Verwaltung der Knoten, was den Konfigurationsaufwand reduziert
- OLSR arbeitet auf Layer 3 und verwendet IP-basiertes Routing, wodurch eine explizite Adressierung und Verwaltung der Routingtabellen erforderlich wird. Dies bietet mehr Kontrolle, jedoch ist mit höherem Verwaltungsaufwand zu rechnen

[18] https://www.open-mesh.org/projects/open-mesh/wiki

[19] https://github.com/ANRGUSC/Raspberry-Pi-OLSRd-Tutorial

4.4.3 Vergleich VoIP Software

Bei der Recherche nach geeigneter VoIP-Software für den RPi Zero 2 WH stösst man im Internet fast ausnahmslos auf zwei etablierte Lösungen: **Mumble**[20] und **Linphone**[21]. Beide Systeme werden in der Raspberry-Pi Community häufig empfohlen, wenn es um Sprachübertragung über IP geht.

Im Rahmen dieses Projekts wird geprüft, welche dieser beiden Lösungen sich besser für den dezentralen Betrieb in einem WLAN-basierten Mesh-Netzwerk eignet. Die Anforderungen an die eingesetzte VoIP-Software leiteten sich direkt aus dem Projektziel ab.

Leistungsstarke Enterprise-Lösungen wie Asterisk, FreeSWITCH oder Kamailio wurden im Zuge der Voranalyse[22] ebenfalls betrachtet, jedoch aus folgenden Gründen nicht weiterverfolgt.

Diese Systeme setzen in der Regel auf eine zentrale Serverarchitektur, sind deutlich komplexer in der Konfiguration und benötigen mehr Rechenleistung, als ein RPi sinnvoll bereitstellen kann. Zudem liegt der Fokus dieser Plattformen auf klassischer SIP-Infrastruktur, was der dezentralen, autonom arbeitenden Mesh-Topologie widerspricht.

Die technische und konzeptionelle Ausrichtung dieser Plattformen ist somit nicht vereinbar mit dem Ziel eines einfachen, mobilen und netzunabhängigen VoIP-Systems, welches umgesetzt werden soll.

[20] https://www.mumble.info/

[21] https://www.linphone.org/en/homepage-linphone/

[22] https://raspberry.piaustralia.com.au/blogs/voip-phone-system/setting-up-a-voip-phone-system-on-raspberry-pi-4

Merkmal	Mumble	Linphone
Architektur	Server + Client	Peer-to-Peer oder zentraler SIP-Server
Verschlüsselung	TLS	TLS, SRTP
Betrieb ohne zentrale Instanz	Ja	Nein, SIP-Server erforderlich
Headless- / CLI-Betrieb	Vollständig möglich	Eingeschränkt
Ressourcenverbrauch	gering	mittel
GUI für Setup und Debugging	Ja	Ja
PTT-Unterstützung	Integriert	Abhängig vom genutzten Client
Mesh-Netzwerk kompatibel	Noch nicht getestet	Ja, getestet mit AREDN-Mesh

Tab. 9: Mumble vs. Linphone

Bewertung

Der Vergleich zeigt, dass Mumble und Linphone unterschiedliche technische Ansätze verfolgen:

- Mumble setzt auf eine klassische Server-Client-Architektur mit integrierter Verschlüsselung und Unterstützung für PTT. Der Betrieb ist vollständig ohne grafische Oberfläche möglich, was für automatisierte oder headless Umgebungen vorteilhaft ist
- Linphone basiert auf SIP und unterstützt sowohl Peer-to-Peer-Kommunikation als auch den Einsatz zentraler Server. Dadurch ist es flexibel in bestehenden Kommunikationsinfrastrukturen einsetzbar, erfordert jedoch in der Regel eine zentrale Instanz für den Verbindungsaufbau
- In Bezug auf die Einbindung in Mesh-Netzwerke existieren für Linphone bereits erprobte Ansätze wie z. B. im Mesh-Projekt AREDN[23], während Mumble bisher nicht in solchen Umgebungen getestet wurde

[23] https://www.youtube.com/watch?v=hGMhpvF8b7Y

4.4.4 Entscheidung und Begründung

Als Betriebssystem wird **Raspberry Pi OS mit Desktop (32-bit)** ausgewählt. Diese Variante bietet eine grafische Oberfläche, die insbesondere in der Entwicklungs- und Testphase hilfreich ist. Trotz der grafischen Umgebung ist das System ressourcenschonend genug, um auf dem RPi stabil zu laufen.

Im Produktivbetrieb starten alle Dienste automatisch im Hintergrund, ohne dass eine Benutzerinteraktion über die GUI erforderlich ist. Für den Fall, dass es zu Problemen mit der grafischen Version kommen sollte, steht als alternative Backup-Lösung die Lite-Variante ohne Desktop bereit.

Zur Umsetzung des selbstorganisierenden Mesh-Netzwerks wird das Layer-2-Protokoll **batman-adv** gewählt. Es lässt sich einfach konfigurieren und läuft stabil unter RPi OS, somit entfällt die Notwendigkeit komplexer Routingtabellen. Alternativen wie OLSR wurden geprüft, jedoch wegen des höheren Konfigurationsaufwands und der geringeren Mobilitätstoleranz nicht berücksichtigt.

Für die Sprachübertragung fällt die Wahl auf **Mumble**, welches auf jedem Gerät lokal betrieben werden kann. Diese Lösung ermöglicht eine verschlüsselte, dezentral betriebene Sprachkommunikation mit geringer Latenz und integriertem PTT-Support. Zudem ist Mumble vollständig automatisierbar und lässt sich via systemd in das Startverhalten des Systems einbinden.

4.5 Wirtschaftlichkeitsbetrachtung der gewählten Komponenten

4.5.1 Projektkosten

Für die Umsetzung des Projekts werden gezielt kostengünstige, handelsübliche Komponenten eingesetzt. Da viele der benötigten Geräte und Zubehörteile bereits vorhanden sind, beschränken sich die effektiven Projektkosten auf die zusätzlich beschaffte Hardware. Die folgenden Angaben beziehen sich auf die tatsächlichen Einkaufspreise inkl. Mehrwertsteuer (8.1 %) und Versandkosten.

Pos.	Komponente	Menge	Einzelpreis (CHF)	Gesamtpreis (CHF)	Händler	Produktnummer
1	Raspberry Pi Zero 2 WH	3	19.90	67.60	pi-shop.ch	12430
2	Micro-USB Netzteil zu RPi Zero	3	7.90	23.70	berrybase.ch	RPIMNT25W
3	SanDisk microSD 128 GB	2	12.65	25.30	berrybase.ch	619659188450
4	WM8960 Sound-HAT inkl. Lautsprecher	3	18.90	56.70	berrybase.ch	RPI-STAHAT
5	Powerbank 10’000 mAh	2	8.55	17.10	temu.com	RV136026
6	HDMI zu USB Capture Card (4K)	1	3.30	3.30	temu.com	TP15429
7	Set aus drahtlose Maus und Tastatur	1	11.80	11.80	temu.com	KA34665
8	RPi Zero Zubehörset (HDMI, OTG, GPIO)	3	1.70	5.10	aliexpress.com	32801547059

Tab. 10: Kostenübersicht Projekt

Bereits vorhandene Komponenten (nicht gekauft):

- 1x USB-Hub
- 1x Fritz!Box 7530
- 1x microSD 128 GB
- 1x Powerbank (unbekannter Hersteller 8400 mAh)
- Diverse Kabel (HDMI, Strom, Netzwerk)
- Verschiedene USB WLAN-Adapter

Bemerkungen:

- Die Tabelle bildet ausschliesslich projektbezogene Neuanschaffungen ab
- Softwarekosten entfallen vollständig, da ausschliesslich Open Source SW eingesetzt wird
- Private Infrastruktur, wie Laptop, Monitor usw. werden nicht berücksichtigt
- Die gesamten Projektkosten belaufen sich auf ca. ***210.– CHF***

4.5.2 Kostenvergleich ähnlicher Varianten

Zur Bewertung der Wirtschaftlichkeit wird die gewählte Systemarchitektur mit zwei alternativen Ansätzen verglichen, die ähnliche Ziele verfolgen, jedoch andere technische Schwerpunkte setzen. Dabei liegt der Fokus auf den Hardwarekosten pro Knoten sowie den Gesamtkosten für ein vergleichbares Setup mit drei Einheiten.

Alle Preisangaben sind in Schweizer Franken, gerundet und verstehen sich inklusive Mehrwertsteuer.

Variante	Beschreibung	Einzelkosten pro Einheit in	Bemerkung
LoRa-basierte Sprachkommunikation	Verwendung von Mikrocontrollern, wie z. B. ESP32 + LoRa-Modul, mit analoger Audioübertragung	ca. 95.–	• Niedrige Datenrate • Sprachqualität stark begrenzt
Kommerzielle, lizenzfreie PMR-Funkgeräte	Fertige Handfunkgeräte mit PTT-Funktion, wie z. B. Midland XT70 Set[24]	ca. 80.–	• Gute Reichweite • keine Verschlüsselung • keine IP-Kommunikation

Tab. 11: Kostenvergleich ähnlicher Varianten

Auswertung:

Der Vergleich zeigt deutlich, dass die gewählte Lösung auf Basis des RPi ein optimales Verhältnis zwischen Kosten, Funktionalität und Erweiterbarkeit bietet.

Einfache Funkgeräte sind zwar günstiger im Betrieb, aber funktional stark eingeschränkt. LoRa-basierte Varianten übersteigen dagegen den finanziellen und technischen Rahmen des Projekts.

Mit einem durchschnittlichen Hardwarepreis von rund **75 CHF** pro Einheit kann das geplante System kostengünstig vervielfältigt und für Tests im Feld flexibel erweitert werden.

Zusätzlich entfallen sämtliche Lizenz- und Softwarekosten, da ausschliesslich Open Source Komponenten verwendet werden.

[24] https://www.digitec.ch/de/s1/product/midland-xt70-pro-12-km-walkie-talkie-36877622

5 Technische Konzeption des Zielsystems

Die technische Konzeption beschreibt die Soll-Architektur des geplanten Kommunikationssystems. Sie überführt die Anforderungen aus Pflichtenheft und Themeneingabe in ein implementierbares Design. Sie definiert Komponenten, Schnittstellen, Datenflüsse, Betriebsabläufe sowie Synchronisationsmechanismen. Das Kapitel bildet somit die Grundlage, an der sich die nachfolgende Umsetzung, die Tests sowie die Abnahme orientieren.

Die Schwerpunkte dieses Kapitels sind:

- Architekturprinzipien der Knoten
- Netzwerktopologie und IP-Konzept
- Komponentenmodelle für Hardware und Software
- Datenflüsse für Audio und Steuerung
- Betriebs- und Startkonzept
- Monitoring und Diagnose

Rückverweise auf konkrete Implementierungsschritte erfolgen in der Umsetzung.

5.1 Systemüberblick

Das Zielsystem besteht aus mehreren identischen, mobil betreibbaren Knoten. Diese verbinden sich selbstständig zu einem drahtlosen WLAN-Mesh und ermöglichen so eine verschlüsselte Sprachkommunikation ohne zentrale Infrastruktur.

Im Zentrum steht der „Walkie-Talkie“-Betriebsmodus mit PTT. Alle Systemdienste starten vollautomatisch beim Einschalten. Die Energieversorgung erfolgt über Powerbanks, sodass die Geräte im Feld unabhängig von Netzstrom und Internet betrieben werden können.

Hauptfunktionen je Knoten

- Aufbau und Erhalt des Mesh über den USB WLAN-Adapter
- Mumble-Server und -Client im Parallelbetrieb
- Audio I/O über den WM8960-HAT, inkl. PTT-Taster
- Autostart sämtlicher Dienste
- Lokaler Betrieb ohne Internet

Betriebsmodi

- Betriebsmodus *Entwicklung*: Managementzugriff über separaten WLAN-Router
- Betriebsmodus *Feld*: isoliertes Mesh ohne Internet

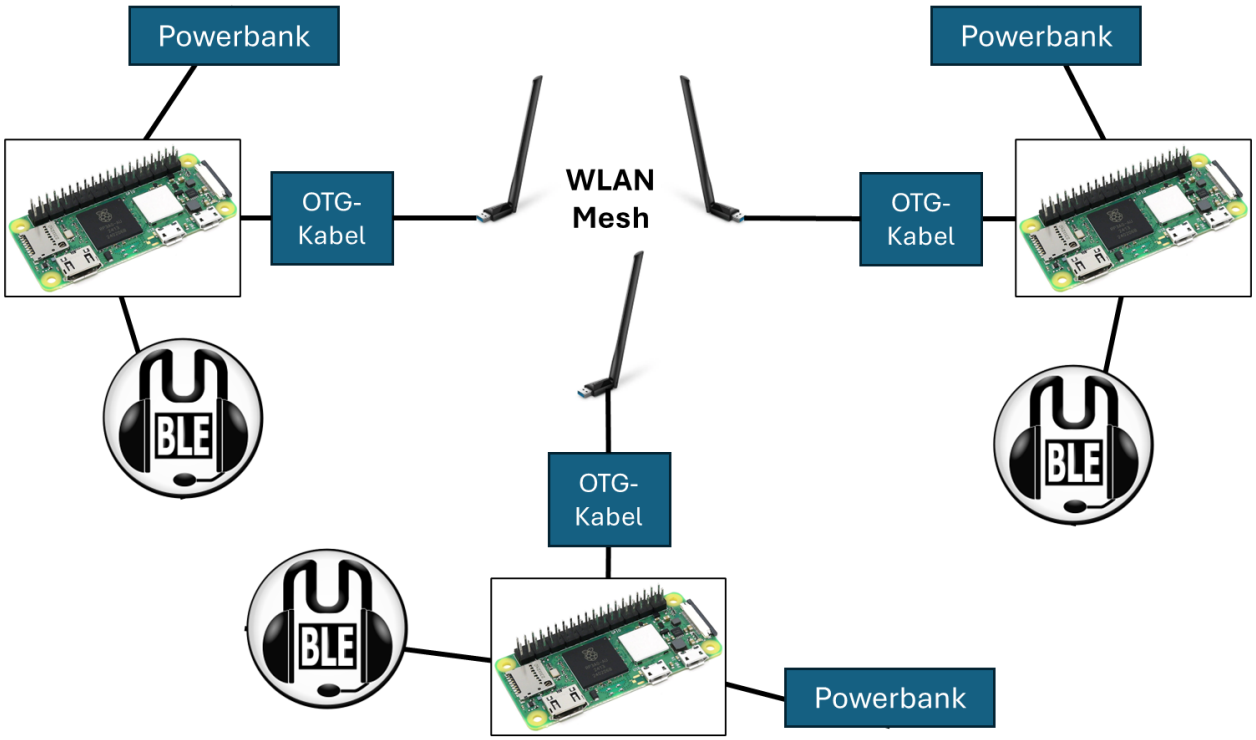

Abb. 11: Systemüberblick

5.2 Systemarchitektur und Netzwerktopologie

Die Architektur basiert auf identischen, autonomen Knoten, die sich zu einem Layer-2-Mesh zusammenschliessen. Jeder Knoten stellt die Dienste Audio, VoIP und Mesh lokal bereit. Die Kopplung zwischen Funk, Mesh, Audio und VoIP erfolgt bewusst schlank, damit der Betrieb ohne zentrale Router und ohne DHCP möglich ist.

5.2.1 Architekturprinzipien

Jeder Knoten kann senden, empfangen und Pakete weiterleiten und die Vernetzung erfolgt als Layer-2-Mesh mit batman-adv. Die Weiterleitung geschieht über das logische Interface *bat0*, damit IP-Dienste ohne klassisches Routing funktionieren. Die Pfade im Mesh werden dynamisch anhand der Linkqualität ausgewählt. Ändert sich die Position oder fällt ein Knoten aus, findet batman-adv automatisch einen alternativen Pfad. Audio und VoIP sind funktional vom Managementzugang getrennt und das produktive Mesh transportiert ausschliesslich die Sprachkommunikation.

Das Verfügbarkeitsziel wird mit Fallback erreicht, dies Aufgrund der begrenzenten Funktionalität vom Mumble Server. Die Mumble-Clients arbeiten mit einer priorisierten Serverliste. Ist der bevorzugte Server nicht erreichbar, verbinden sie sich automatisch mit einem alternativen Server im Mesh. Der Bedienfokus liegt auf einer einfachen Inbetriebnahme im Feld. Nach dem Einschalten starten alle benötigten Dienste automatisch, die Aufnahme wird über den PTT-Taster gesteuert und bleibt nur während gedrückter Taste aktiv.

5.2.2 Netzwerktopologie

Die Netzwerktopologie basiert auf einem drahtlosen Layer-2-Mesh. Physisch funkt jeder Knoten über einen externen USB WLAN-Adapter im Ad-hoc-Modus. Dieses Interface wird in *batman-adv* eingebunden. Die Logikschicht bildet **bat0**, über das alle Sprachpakete laufen.

Die Schnittstellen und Rollen sind klar getrennt:

- **wlan0 (OnBoard):** Managementzugang im Entwicklungsbetrieb im getrennten Subnetz für SSH und VNC, dies ist im Feldbetrieb nicht aktiv
- **wlan1:** physisches Funkinterface für das Mesh im Ad-hoc-Modus
- **bat0:** logische Mesh-Schnittstelle von batman-adv, diese erhält die feste Mesh-IP und transportiert sämtliche Verbindungen

Der Systemstart folgt einem definierten Ablauf, damit das Mesh vor den Applikationsdiensten bereitsteht:

1. Laden des WM8960-Overlays und der Systemtreiber
2. Mesh-Service setzt wlan1 in den Ad-hoc-Modus, bindet es in batman-adv ein, aktiviert bat0
3. bat0 erhält die feste Mesh-IP
4. PTT-Service initialisiert GPIO und setzt Mikrofon systemweit auf aus
5. Mumble-Server startet lokal
6. Mumble-Client verbindet sich mit der priorisierten Serveradresse im Mesh

Die Topologie verhält sich selbstheilend und ist skalierbar. Es sind mehrere Hops möglich und die Pfadlänge hängt von der Funkreichweite sowie der Position der Knoten ab. Störungen, Bewegung oder Abschattung führen automatisch zu einer neuen Routenwahl. Der Datenverkehr bleibt lokal, Gateways ins Internet sind nicht vorgesehen.

Das Managementnetz wird während Tests über eine Fritz!Box bereitgestellt, bleibt jedoch im späteren Einsatz abgeschaltet. Im Feldbetrieb bildet das Mesh-Netz die alleinige Kommunikationsbasis.

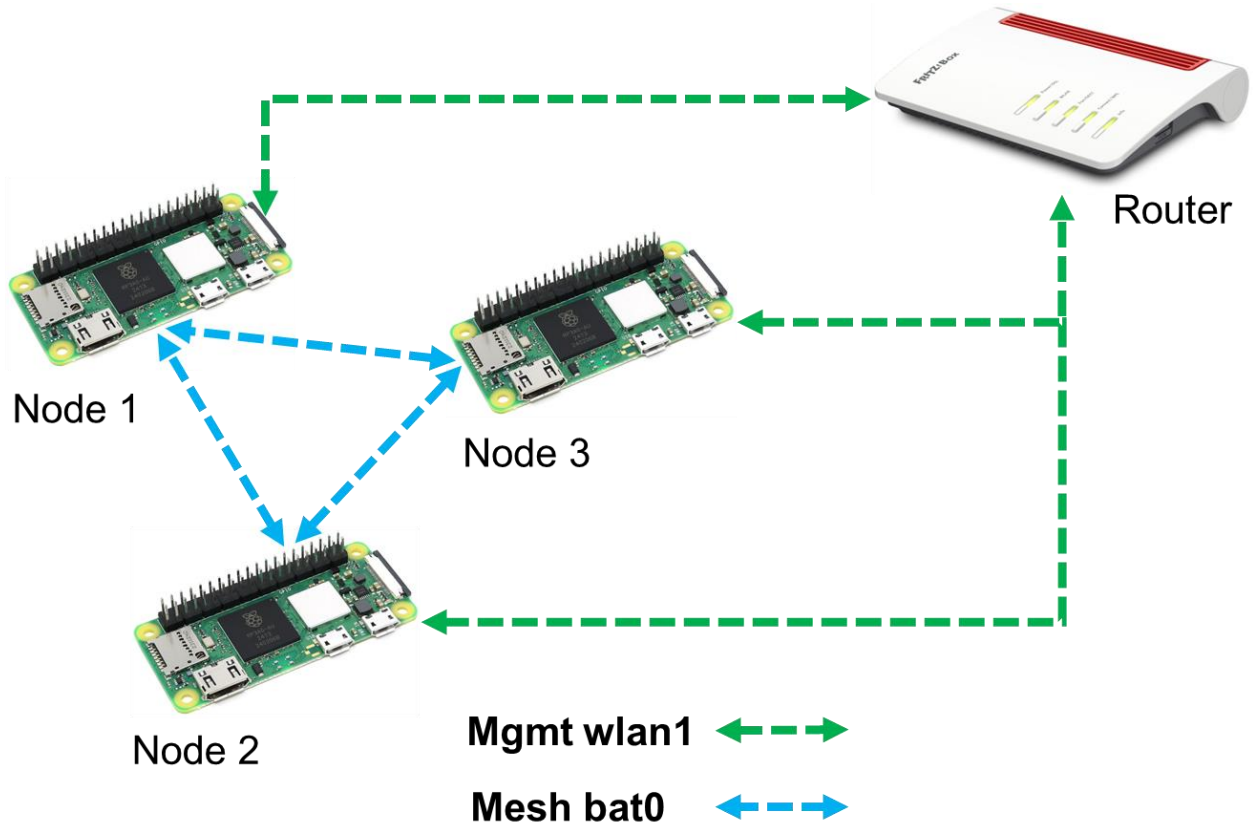

Abb. 12: Netzwerktopologie

5.3 IP-Konzept und Adressierung

Das Ziel dieses IP-Konzepts ist eine klare Trennung von Managementzugang und produktiver Sprachkommunikation, um asymmetrisches Routing und unnötige Abhängigkeiten zu vermeiden.

Die Adressierung ist statisch, damit alle Knoten auch ohne DHCP und ohne Internet zuverlässig arbeiten.

Trennungsprinzip der Netze

- *Managementnetz* auf wlan0: *172.30.5.0/24*
  Nutzung ausschliesslich in der Entwicklung für SSH, VNC und Administration
- *Produktives Mesh* auf bat0: *10.30.5.0/24*
  Immer aktiv, trägt alle VoIP-Verbindungen im Feldbetrieb

Folgende IP-Zuordnung soll je Knoten umgesetzt werden:

Knoten	Hostname	User	Interface	IP-Adresse
RPi Zero Node 1	pi-node-1	user01	wlan0	172.30.5.10
			wlan1 / bat0	10.30.5.10
RPi Zero Node 2	pi-node-2	user02	wlan0	172.30.5.20
			wlan1 / bat0	10.30.5.20
RPi Zero Node 3	pi-node-3	user03	wlan0	172.30.5.30
			wlan1 / bat0	10.30.5.30

Tab. 12: IP-Zuordnung

5.4 Komponentenmodell Hardware

5.4.1 Hardware pro Knoten

Um Austauschbarkeit und einfache Wartung sicherzustellen, ist jeder Knoten identisch aufgebaut. Folgende Hardware wird pro Knoten verwendet:

Komponente	Typ	Zweck
Einplatinencomputer	Raspberry Pi Zero 2 WH	Zentrale Systemplattform
Speicher	microSD-Karte	Betriebssystem und Software
Audio-Hardware #1	WM8960 Sound-HAT	PTT und Anschluss für Speaker
Audio-Hardware #2	8 Ohm, 5W Speaker	Voice Wiedergabe
WLAN-Adapter	Ralink WLAN-Chipsatz	Verbindung im Mesh-Netzwerk
Stromversorgung	Powerbank	Mobiler Betrieb
USB-Adapter	Micro USB-B zu USB-A OTG	Anschluss von WLAN-Adapter

Tab. 13: Hardware pro Knoten

Nach dem Zusammenbau ergibt sich eine kompakte Einheit, die mechanisch stabil ist und die elektrischen Verbindungen über standardisierte Steckverbinder herstellt. Eine Montageanleitung folgt in der Umsetzung.

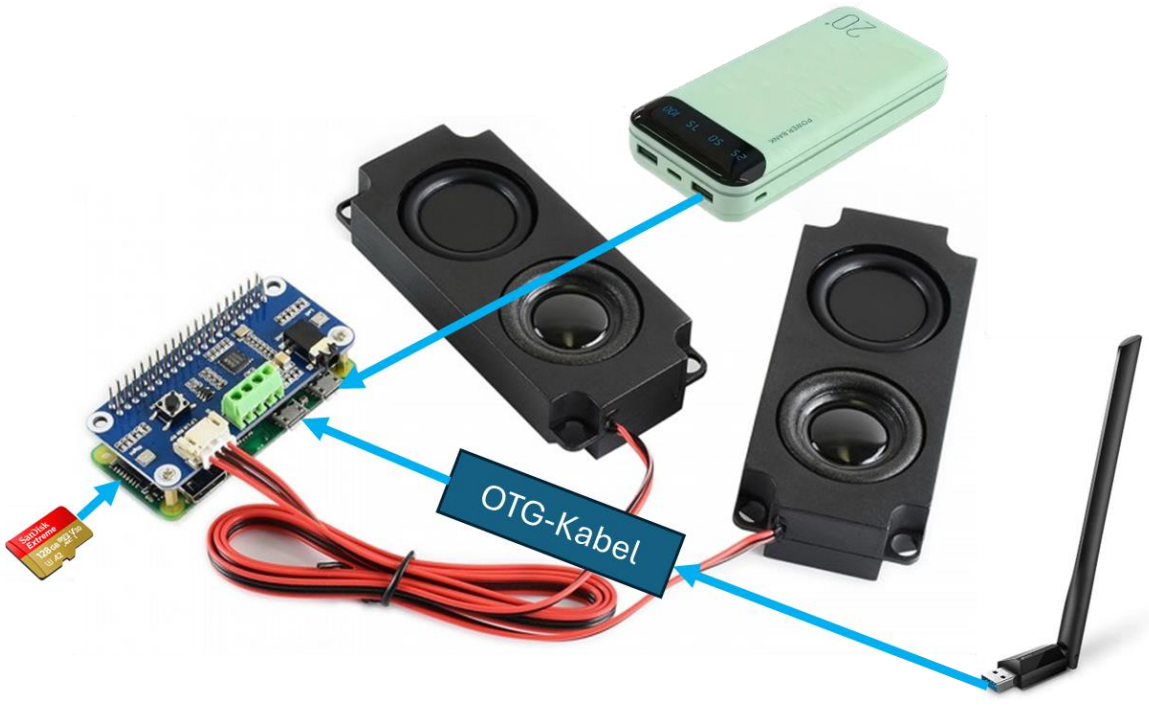

Abb. 13: Knoten mit kompletter Hardware

5.4.2 Spezifikationen der Hardware

In diesem Abschnitt werden die technischen Daten und die relevanten Eigenschaften jeder Hauptkomponente kurz beschrieben. Ziel ist, die betrieblichen Anforderungen und die Schnittstellen zwischen den Komponenten zu erläutern. Die aufgelisteten Werte sind für das Projekt relevant. Die vollständigen Datenblätter sind in den Quellen zu finden oder werden in der Umsetzung berücksichtigt.

5.4.2.1 Raspberry Pi Zero 2 WH

Der Raspberry Pi Zero 2 WH ist ein kompakter Einplatinencomputer mit einem 40-Pin GPIO Header. Er dient als zentrale Rechenplattform für das Betriebssystem, die Mesh-Protokolle, die VoIP-Software sowie für Skripte zur Steuerung von PTT und Audiosystem.

Wesentliche technische Merkmale:

- Arbeitsspeicher: 512 MB
- Schnittstellen: 40-Pin GPIO, Micro-USB, Mini-HDMI
- Funk: Onboard WLAN 2.4 GHz und Bluetooth
- Vorbestückter GPIO-Header bei WH-Version (W = Wireless, H = Header)

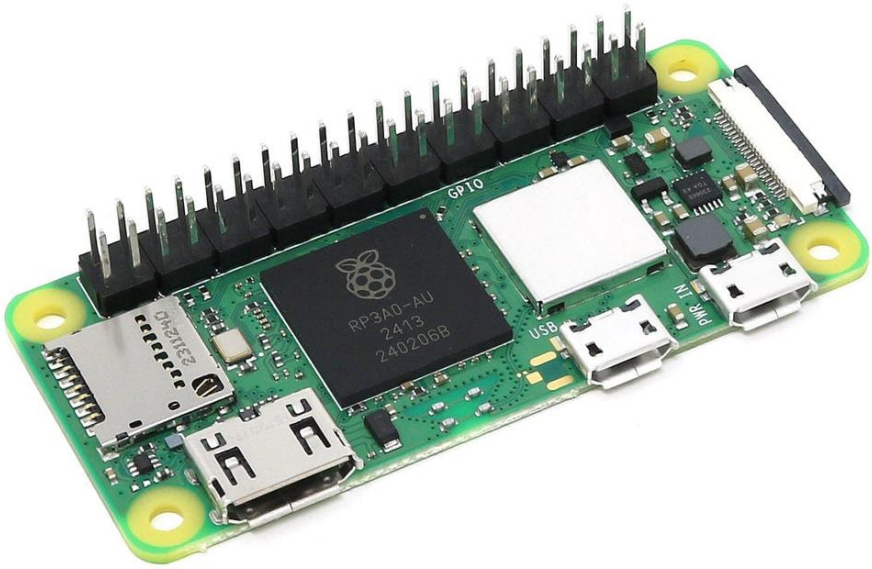

Abb. 14: RPi Zero 2 WH

Die 40-Pin GPIO Schnittstelle ermöglicht die direkte Anbindung des Audio-HAT über I2S sowie das Abfragen des PTT-Buttons. Bei Bedarf kann über GPIO ein externer Taster oder eine LED-Signalisierung ergänzt werden.

5.4.2.2 WM8960 Hi-Fi Sound Card HAT

Der HAT ist eine für den RPi ausgelegte Audioerweiterung, die den WM8960 Codec als Herzstück nutzt. Er bietet Ein- und Ausgänge für Mikrofone und Lautsprecher, einen integrierten PTT-Taster und wird über I2S für Audiodaten mit dem Pi verbunden. Die Steuerung erfolgt über I2C. Das Modul eignet sich für einfache Sprach- und Aufnahmeaufgaben und ist in der Community gut dokumentiert.

Wesentliche technische Merkmale

- Codec: WM8960
- Schnittstellen: Audio über I2S, Steuerung über I2C, GPIO / 40-Pin HAT-Header
- Aufnahme: integrierte MEMS-Mikrofone
- Wiedergabe: 3.5 mm Klinke und Lautsprecher-Klemmen
- Stromversorgung: Versorgt über 5 V des Raspberry Pi

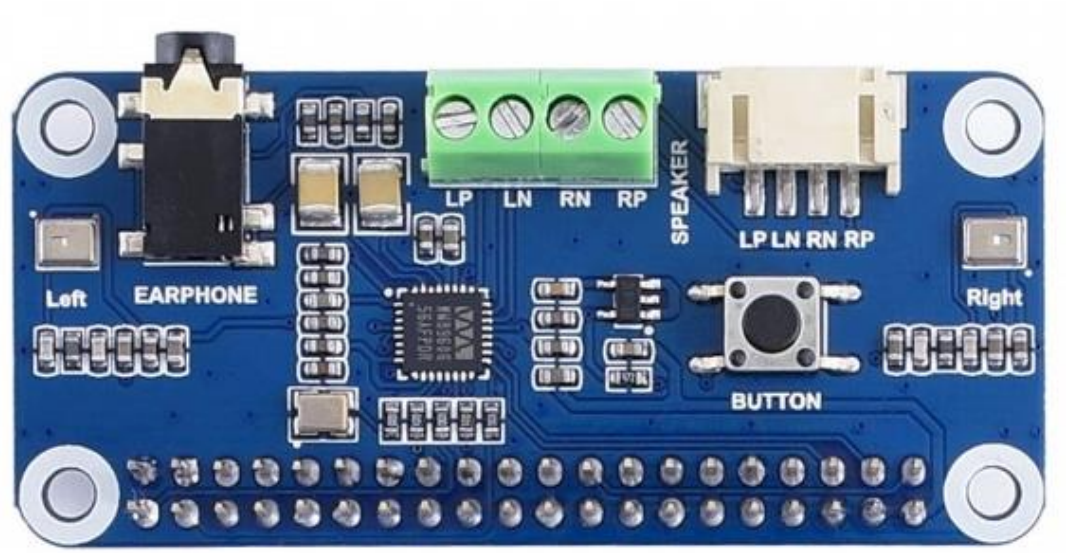

Abb. 15: WM8960 Hi-Fi Sound Card HAT[25]

Der HAT stellt die physische Schnittstelle für PTT, Mikrofon und Lautsprecher bereit. Der integrierte Taster vereinfacht die Implementierung der PTT-Funktion, da er direkt per GPIO ausgelesen werden kann. Die I2S Anbindung ermöglicht eine zuverlässige Audioübertragung mit niedriger Latenz im lokalen System.

[25] https://www.waveshare.com/wm8960-audio-hat.htm

5.4.2.3 WLAN-Adapter

Um Flexibilität hinsichtlich Kanalwahl und Modus zu erhalten, werden für das Mesh externe WLAN-Adapter eingesetzt. Ein wichtiger Auswahlfaktor ist der verwendete WLAN-Chipsatz, da dieser die Linux-Treiberunterstützung und die Unterstützung der benötigten Betriebsmodi bestimmt.

Wesentliche technische Merkmale

- Unterstützte Betriebsmodi: IBSS (Ad-Hoc) oder mesh point (802.11s)
- Treiber: Muss im Linux-Kernel vorhanden sein, um Plug-and-Play zu funktionieren
- Frequenzband: Es wird nur 2.4 GHz genutzt
- Stromaufnahme: USB-betrieben

Abb. 16: WLAN-Adapter mit Ralink Chipsatz RT5370[26]

Die Adapter wurden überwiegend aus asiatischen Quellen beschafft. Entscheidend ist der verbaute Chipsatz und nicht die Markenbezeichnung. Vor der Nutzung sollte ein Voraussetzungscheck unter der eingesetzten OS-Version durchgeführt werden.

Die genutzten WLAN-Chipsätze sind vom Hersteller MediaTek / Ralink Technology, Corp. vom Typ RT2870/RT3070 und RT5370 Wireless Adapter. ([13] MediaTek, 2025)

[26] https://de.aliexpress.com/item/1005008142040949.html

5.4.2.4 Weitere Hardware

Neben den Hauptkomponenten kommen im System einige standardisierte Zusatzkomponenten zum Einsatz. Diese sind für den stabilen Betrieb und die Mobilität der Knoten erforderlich. Dazu gehören die microSD-Karte, die Powerbank, das OTG-Kabel und der Lautsprecher.

Als Speichermedium wird eine **microSD-Karte** mit 128 GB Kapazität verwendet. Sie dient als Boot- und Systemlaufwerk, auf dem das OS, die Software und die Konfigurationsdateien abgelegt sind. Bewährte Marken wie SanDisk oder Samsung garantieren eine zuverlässige Datenspeicherung und sind für den Dauerbetrieb auf dem RPi geeignet. Die Vorbereitung der Karte erfolgt mit dem *Raspberry Pi Imager*, über den das benötigte Systemimage direkt auf die Karte geschrieben werden kann.

Die Stromversorgung jedes Knotens erfolgt über eine **Powerbank** mit einer Kapazität von 10’000 mAh. Diese ermöglicht einen vollständig mobilen Betrieb ohne externe Netzverbindung. Dabei ist ein stabiler Ausgang von 5 V bei mindestens 2 A Stromstärke wichtig, um auch bei höherer Systemlast eine stabile Versorgung zu gewährleisten. Eine automatische Abschaltung der Powerbank muss entweder deaktivierbar sein oder durch eine konstante Systemlast verhindert werden, da der geringe Ruhestrom des RPi im Leerlauf sonst zu einem unbeabsichtigten Ausschalten führen könnte.

Für den Anschluss des WLAN-Adapters wird ein Micro-USB-B zu USB-A **OTG-Kabel** verwendet. Danke dieses Kabels ist es möglich, USB-Peripheriegeräte an den OTG-fähigen Anschluss des RPi anzuschliessen. Dadurch kann der externe WLAN-Adapter sicher und mechanisch mit dem Gerät verbunden werden.

Zur Sprachwiedergabe wird ein **Lautsprecher** mit einer Impedanz von 8 Ohm und einer Nennleistung von 5 W verwendet. Er wird direkt an die entsprechenden Steckklemmen des WM8960 Audio HAT angeschlossen. Das Modul kann kleine Lautsprecher direkt antreiben, sodass keine zusätzliche Verstärkereinheit erforderlich ist.

5.5 Komponentenmodell Software

Für den Betrieb der einzelnen Knoten wird ausschliesslich Open Source Software eingesetzt. Die Auswahl der Komponenten erfolgt unter Berücksichtigung der Kompatibilität mit dem RPi Zero 2, der Stabilität im Dauerbetrieb sowie der geringen Systemressourcen des Geräts.

5.5.1 Software pro Knoten

Jeder Knoten ist softwareseitig identisch aufgebaut, um Kompatibilität und vereinfachte Wartung sicherzustellen.

Die nachfolgende Übersicht zeigt die eingesetzten Softwarekomponenten:

Komponente	Typ / Paket	Zweck
Betriebssystem	Raspberry Pi OS (32-bit) mit Desktop	Leichtgewichtiges Debian-System mit GUI, welches hilfreich für Konfiguration, Entwicklung und Diagnose
Netzwerk / Routing	batman-adv, batctl	Kernelmodul und CLI-Tool zur Bildung und Überwachung eines selbstorganisierenden Mesh-Netzwerks
Sprachkommunikation	Mumble-Server und Mumble-Client	Sprachübertragung über VoIP innerhalb des Mesh-Netzes
Audiosoftware	PulseAudio, alsamixer, WM8960-Treiber, alsa-utils	Audioaufnahme, Wiedergabe und Steuerung über GPIO-Taster
Automatisierung und Skripte	Python 3	Steuerlogik für PTT und Fallback-Mechanismen
Zeitsynchronisation	chrony	Lokale NTP-Synchronisation zwischen den Knoten

Tab. 14: Software pro Knoten

Auf dem Entwicklungsrechner wird zusätzlich folgende Software genutzt:

- **OBS Studio:** Video- und Bildschirmaufnahme für die Testdokumentation
- **VNC Viewer (RealVNC):** Fernzugriff auf das Pi-Desktop GUI
- **Raspberry Pi Imager:** Schreiben des System-Images auf die microSD-Karte

5.5.2 Spezifikation der Software

Die folgenden Abschnitte beschreiben die wichtigsten Softwarekomponenten und deren Funktion im Gesamtsystem.

5.5.2.1 Raspberry Pi OS (32-Bit) mit Desktop

Das *Raspberry Pi OS (32-bit) mit Desktop* ist eine leichtgewichtige Linux-Distribution auf Debian-Basis, die für die Hardware des RPi optimiert wurde.

Die grafische Oberfläche ermöglicht eine komfortable Nutzung von Anwendungen wie dem Mumble-Client und unterstützt insbesondere die Entwicklung, Erstkonfiguration und Fehlersuche.

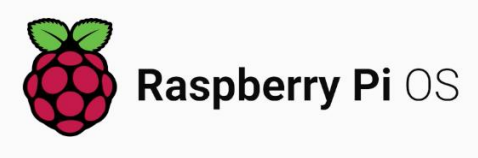

Abb. 17: Raspberry Pi OS[27]

5.5.2.2 batman-adv und batctl

Das Kernelmodul *batman-adv* stellt ein Layer-2 Mesh-Protokoll bereit, das eine dynamische, selbstheilende Netzwerkstruktur zwischen den Knoten aufbaut.

Mit dem Kommandozeilenwerkzeug *batctl* lassen sich Konfiguration, Routingstatus und Linkqualität überwachen. Dadurch kann der Zustand des Mesh-Netzes im laufenden Betrieb analysiert werden.

Abb. 18: batman-adv von Open-Mesh[28]

[27] https://commons.wikimedia.org/wiki/File:Raspberry_Pi_OS_Logo.png

[28] https://www.pngkit.com/downpic/u2a9o0y3o0e6q8q8_open-batman-adv/

5.5.2.3 Mumble

5.5.2.3.1 Server

Jeder Knoten betreibt einen eigenen Mumble-Server im Hintergrund. Dadurch entsteht eine dezentrale Struktur, bei der kein zentraler Server erforderlich ist. Die Kommunikation zwischen Client und Server ist durchgehend per TLS verschlüsselt, sodass die Vertraulichkeit und Integrität der Audioübertragung im Mesh gewährleistet sind.

Da die derzeit genutzte Mumble-Version 1.3.4-4 keine nativen Cluster-Funktionalitäten bietet, wird bewusst ein Fallback-Mechanismus eingeführt. Die Clients können ihre Verbindung überwachen und bei einem Ausfall eines Servers automatisch zu einem anderen erreichbaren Server wechseln.

Laut einem Forenbeitrag ist geplant, dass Clients künftig mehrere Serververbindungen gleichzeitig unterstützen. Allerdings erfordert das „grosse Änderungen an der Architektur“, weshalb eine Umsetzung erst nach einiger Zeit erfolgen kann. ([15] Mumble, 2025)

5.5.2.3.2 Client

Der Mumble-Client wird auf jedem Knoten installiert und verbindet sich automatisch mit dem konfigurierten Server. Die grafische Benutzeroberfläche erleichtert die Erstkonfiguration und Audio-Diagnose. Für den autonomen Betrieb im Feld wird der Client über ein Python-Skript gesteuert und per systemd automatisch gestartet.

Aktuell existiert primär nur eine GUI-Version des Clients. Ein offizieller, reiner Terminalclient ohne GUI ist im Mumble-Projekt nicht etabliert. Einige externe Projekte versuchen, Terminal- oder CLI-Clients zu realisieren, wie z. B. das Rust-Projekt **mum** als Daemon/CLI-Client für Mumble. Dies ist jedoch nicht Teil des offiziellen Mumble-Projekts. ([16] mum-rs, 2025)

Ob eine offizielle Terminalversion von Mumble ohne GUI in zukünftigen Versionen vorgesehen ist, lässt sich anhand der offiziellen Quellen nicht bestätigen. In den offiziellen Mumble-Diskussionen und Roadmaps findet sich bislang keine verlässliche Ankündigung, dass ein GUI-loser Client Bestandteil des Projekts werden soll.

Abb. 19: Mumble[29]

[29] https://www.mumble.info/

5.5.2.4 Audiosoftware

Die Audiosoftware bildet eine zentrale Komponente des Kommunikationssystems, da sie Aufnahme, Wiedergabe und Steuerung der Sprachsignale übernimmt. Der eingesetzte Audio-Stack basiert vollständig auf Open Source Technologien und ist speziell auf den Embedded-Betrieb des RPi Zero 2 WH abgestimmt.

Die Softwarekomponenten greifen dabei in mehreren Schichten ineinander, vom Anwendungsprogramm bis zur physischen Audiohardware.

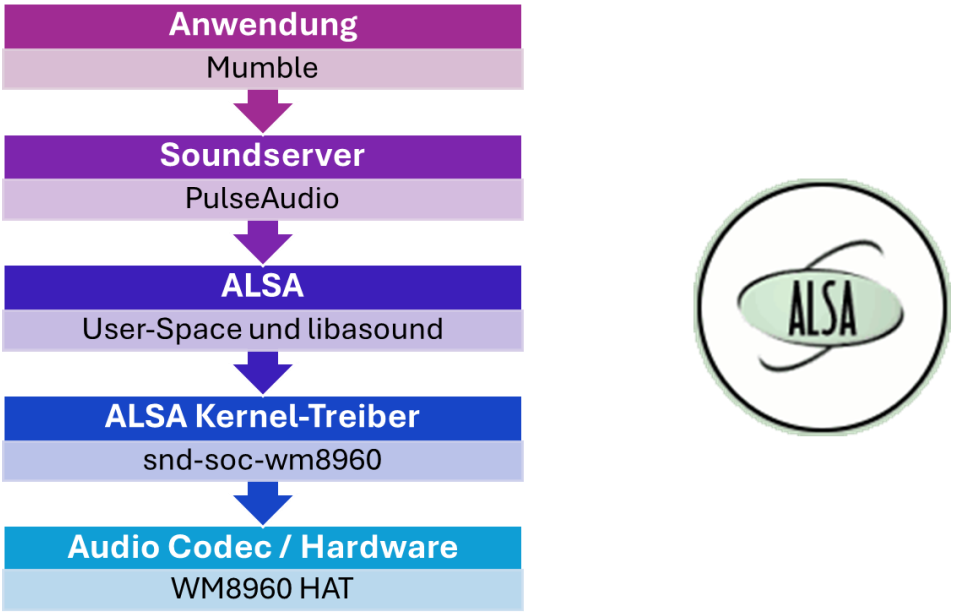

Abb: 20: Audio-Stack-Struktur

1. **Anwendungsebene**

Anwendungen wie Mumble oder VLC erzeugen bzw. empfangen Audiodaten.

Sie nutzen standardmässig Schnittstellen wie PulseAudio, PipeWire oder direkt ALSA, um Audio-Streams an den System-Soundserver oder direkt an die Treiber zu übergeben.

2. **Soundserver (PulseAudio)**

Der Soundserver vermittelt zwischen mehreren gleichzeitigen Anwendungen und den darunterliegenden Audio-Schnittstellen.

Er ermöglicht das gleichzeitige Abspielen und Aufnehmen mehrerer Quellen, das Mischen von Kanälen sowie die Anpassung von Lautstärke und Routing.

Für den Embedded-Betrieb auf den Knoten kann PulseAudio beibehalten oder vollständig deaktiviert werden, falls eine direkte Steuerung über ALSA bevorzugt wird.

3. **ALSA (Advanced Linux Sound Architecture)**

ALSA ist die Kern-Audioarchitektur von Linux und ermöglicht den Zugriff auf Audiogeräte aus dem Userspace. Sie besteht aus der Bibliothek *libasound*, den Hilfswerkzeugen *alsa-utils* und verschiedenen Konfigurationsdateien.

Mit dem Befehl `alsamixer` kann der Benutzer im Terminal den Soundmixer öffnen und über die Pfeiltasten die Lautstärke-Pegel für Ein- und Ausgang einstellen.

Testbefehle wie `speaker-test`, `arecord -d 3 test.wav` oder `aplay test.wav` ermöglichen die Überprüfung der Funktionalität des Audio-Subsystems.

4. **ALSA-Kernelmodul (snd-soc-wm8960)**

Das Kernelmodul *snd-soc-wm8960* gehört zum ALSA-SoC-Subsystem und stellt die Schnittstelle zum WM8960-Audio-Codec her. Es sorgt für die Kommunikation zwischen CPU-DAI, wie z. B. Broadcom SoC des RPi und Codec-DAI über den I2S-Bus.

Die Aktivierung erfolgt über ein Device-Tree-Overlay in der Datei **/boot/config.txt**, wodurch der Treiber beim Booten automatisch geladen wird.

5. **Hardwareebene (WM8960-Codec)**

Der WM8960 ist ein digital-analog Audio-Codec, der über I2S und I2C mit dem RPi kommuniziert. Er wandelt analoge Mikrofon- und Lautsprechersignale in digitale Datenströme um und umgekehrt. Das Soundmodul ist fest auf dem RPi montiert und stellt sowohl die physikalischen Anschlüsse als auch eine GPIO-Schnittstelle bereit.

Zusätzlich stellt es regelbare Verstärker und Filter bereit. Der integrierte Taster dient als PTT-Auslöser und wird über ein Python-Skript via GPIO eingelesen.

Hinweis:

Im Rahmen des Projekts wird PulseAudio als Standardschnittstelle gewählt, da diese Lösung eine stabile Kommunikation mit dem Mumble-Client gewährleistet und eine flexible Steuerung der Audioflüsse ermöglicht.

Ein direkter Betrieb nur über ALSA hätte zwar eine etwas geringere Latenz, jedoch führt das zu einer komplexeren Konfiguration und eingeschränkten Kompatibilität mit grafischen Anwendungen.

5.5.2.5 Python

Python ist eine weit verbreitete Programmiersprache, die sich besonders gut für automatisierte Aufgaben, Skripting, KI-Anwendungen und die Systemsteuerung eignet. Sie wird häufig auf Embedded-Systemen wie dem RPi eingesetzt, da sie leichtgewichtig, flexibel und gut dokumentiert ist.

Beim offiziellen RPi OS ist Python bereits vorinstalliert, in der Regel in Form von *Python 3*, der aktuellen Standardversion für moderne Skripte. Zusätzlich steht mit *pip* auch der Paketmanager für Python-Module zur Verfügung, sodass sofort eine vollständige Python-Umgebung genutzt werden kann.

Python kommt für mehrere zentrale Aufgaben zum Einsatz. Ein mit KI entwickeltes Skript überwacht den GPIO-Eingang eines PTT-Tasters und aktiviert bzw. deaktiviert das Mikrofon im System. Ein weiteres Skript implementiert einen Fallback-Mechanismus, der bei einem Verbindungsverlust automatisch eine Neuverbindung zum Server einleitet. Alle Python-Skripte werden über systemd-Dienste beim Systemstart automatisch ausgeführt und ermöglichen so einen autonomen Betrieb ohne Benutzereingriff.

Abb. 21: python[30]

[30] https://igate-eg.com/what-is-python/

5.5.2.6 chrony

Für eine grundlegende Zeitsynchronisation zwischen den Knoten wird *Chrony* verwendet. Chrony ist eine schlanke und moderne Implementierung des Network Time Protocols, die speziell für instabile Netzwerke, kurze Verbindungszeiten und Embedded-Systeme optimiert wurde. Im Gegensatz zu klassischen NTP-Diensten reagiert Chrony schnell auf Zeitabweichungen und eignet sich gut für Umgebungen mit mobilen Geräten.

Ein Knoten agiert als lokaler NTP-Master, während sich die anderen Knoten im Netzwerk automatisch mit ihm synchronisieren. Dies gewährleistet eine konsistente Systemzeit auf allen Geräten. Das ist besonders wichtig für Log-Einträge, Ereigniskorrelation und eine präzise Analyse und Diagnose im Fehlerfall.

Abb. 22: chrony[31]

[31] https://chrony-project.org/index.html

5.6 Datenflüsse für Audio

Die Datenflüsse im System lassen sich in zwei Hauptpfade gliedern: Audioaufnahme und Audiowiedergabe. Diese Prozesse laufen auf jedem RPi lokal und werden durch die Netzwerkverbindung über das Mesh synchronisiert.

5.6.1 Audioaufnahme | Sendepfad

Wird die PTT-Taste gedrückt, wird über den GPIO-Pin 17 ein Event ausgelöst. Der Dienst ptt.service setzt daraufhin das Mikrofon im Soundmodul via amixer auf ON.

Sobald die Sprachübertragung aktiv ist, erfasst ALSA den PCM-Audiostream und übergibt ihn an den Mumble-Client. Dieser komprimiert die Daten mit dem Opus-Codec, verschlüsselt sie mit TLS, übergibt die Daten an den Mumble-Server und dieser überträgt die Audiodaten über das Interface bat0 zu den Clients.

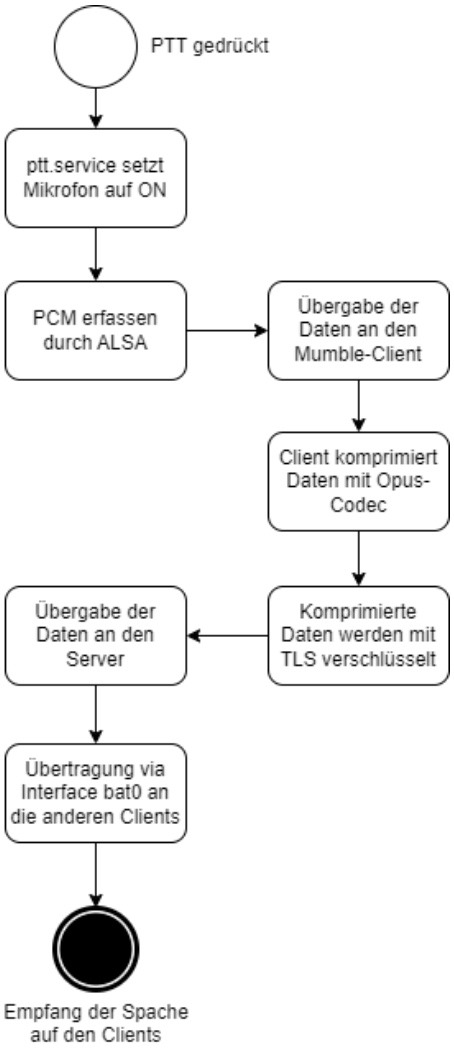

Abb. 23: Flussdiagramm Audio Sendepfad

5.6.2 Audiowiedergabe | Empfangspfad

Beim Empfang von Sprachdaten dekodiert der Mumble-Client die verschlüsselten Opus-Pakete und übergibt die PCM-Daten an ALSA. Von dort werden sie über den WM8960-DAC an den Lautsprecher ausgegeben.

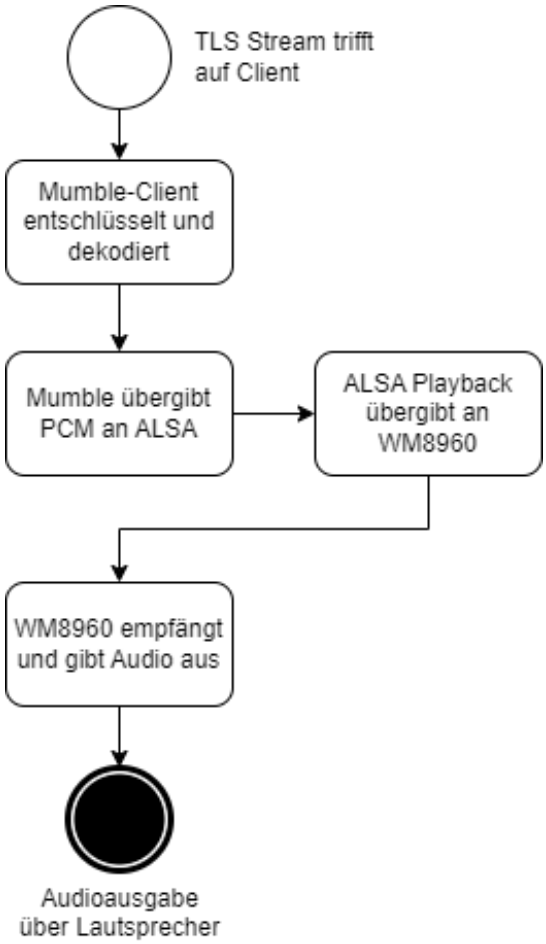

Abb. 24: Flussdiagramm Audio Empfangspfad

5.7 Betriebs- und Startkonzept

Das Betriebs- und Startkonzept definiert den Ablauf nach dem Einschalten eines Knotens sowie den automatischen Verbindungsaufbau, die Wiederherstellung der Kommunikation bei Störungen und die Rückkehr in den Normalbetrieb. Ziel ist ein vollständig autonomer Systemstart ohne Benutzereingriff, bei dem alle Dienste in der richtigen Reihenfolge aktiviert werden.

5.7.1 Boot- und Startreihenfolge

Nach dem Einschalten initialisiert das System automatisch die Treiber und Netzwerkkomponenten. Die Startreihenfolge wird über systemd gesteuert, wodurch sich Abhängigkeiten zwischen den Diensten gezielt definieren lassen.

Wesentlich ist, dass das Mesh-Netzwerk vollständig aufgebaut ist, bevor die VoIP-Dienste gestartet werden. Nur so ist gewährleistet, dass der Mumble-Client beim Start einen erreichbaren Server findet.

Ablauf nach dem Systemstart:

- *Kernel- und Gerätetreiber*
  Laden der Hardwaretreiber für Audio und WLAN
- *mesh.service*
  Aktiviert das Interface wlan1, setzt es in den Ad-hoc-Modus, bindet es in batman-adv ein und setzt eine statische IP im Mesh-Subnetz
- *ptt.service*
  Initialisiert das Audio-Subsystem und konfiguriert die GPIO-Pins für PTT
- *mumble-server.service*
  Startet den lokalen VoIP-Serverdienst auf jedem Node
- *mumble-client.service*
  Startet automatisch den Mumble-Client im Headless-Modus und verbindet sich mit dem Primärserver

5.7.2 Normalbetrieb

Im Normalbetrieb verbindet sich jeder Client nach dem Bootvorgang automatisch mit dem Primärserver *pi-node-1*. Dieser feste Auto-Connect ist die stabilste Variante, da keine unnötigen Wechsel zwischen den Servern auftreten. Die Sprachkommunikation erfolgt vollständig über das Mesh-Netzwerk, ohne zentrale Infrastruktur.

Die PTT-Funktion aktiviert den Audioeingang und überträgt die Sprache verschlüsselt via Mumble über das Mesh an die verbundenen Clients.

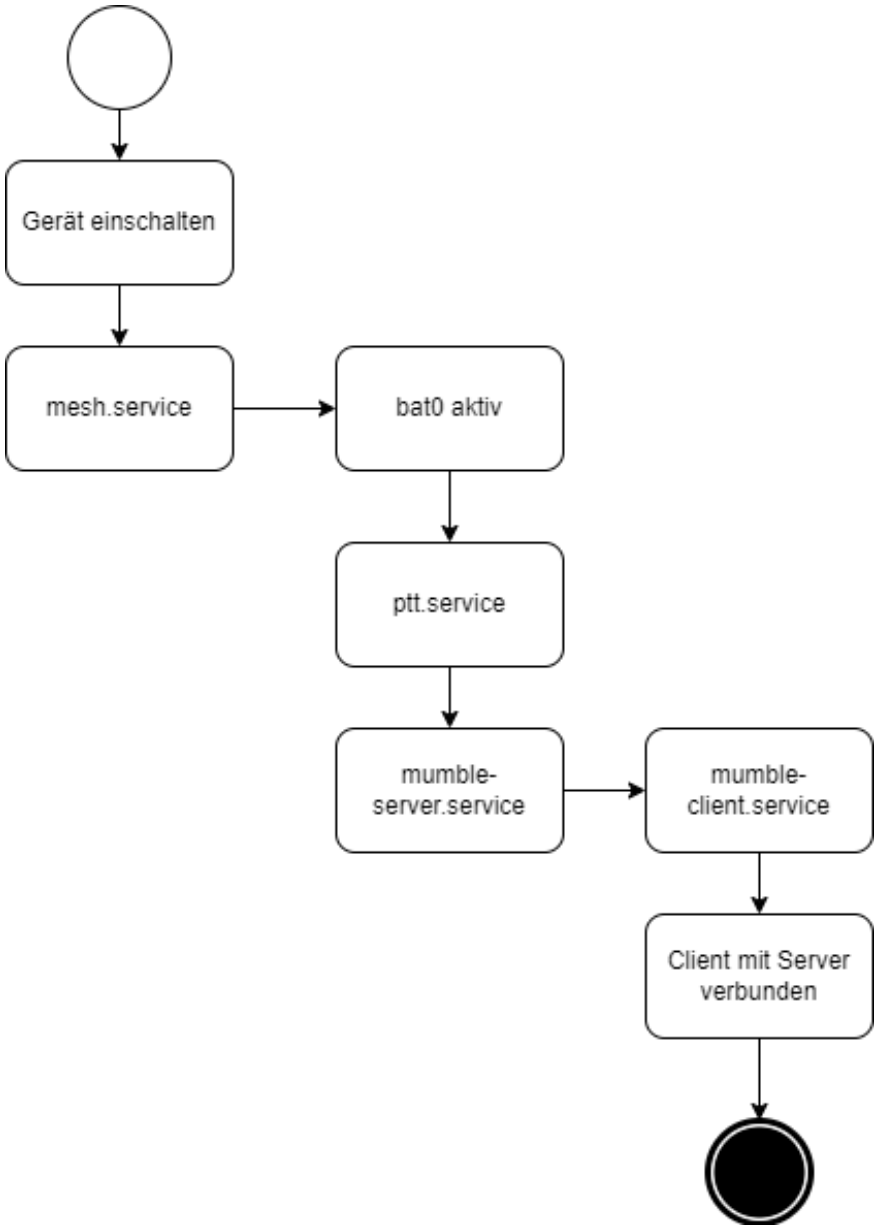

Abb: 25: Flussdiagramm Normalstart eines Nodes

5.7.3 Fallbackstrategie

Die Fallback-Logik sorgt bei Ausfall des Primärservers für eine fast unterbrechungslose Kommunikation. Das Python-Skript überwacht die aktive Verbindung in regelmässigen Abständen.

Wird keine Antwort vom Server empfangen, führt das Skript die folgenden Schritte aus:

1. **Abbruch der aktuellen Verbindung**
   Der Clientprozess wird beendet.
2. **Umschaltung auf den nächsten verfügbaren Server**
   Verbindung zu Node 2 oder Node 3 wird versucht.
3. **Rückwechsel**
   Sobald der Primärserver wieder erreichbar ist, erfolgt automatisch ein Rückwechsel auf diesen bevorzugten Server.

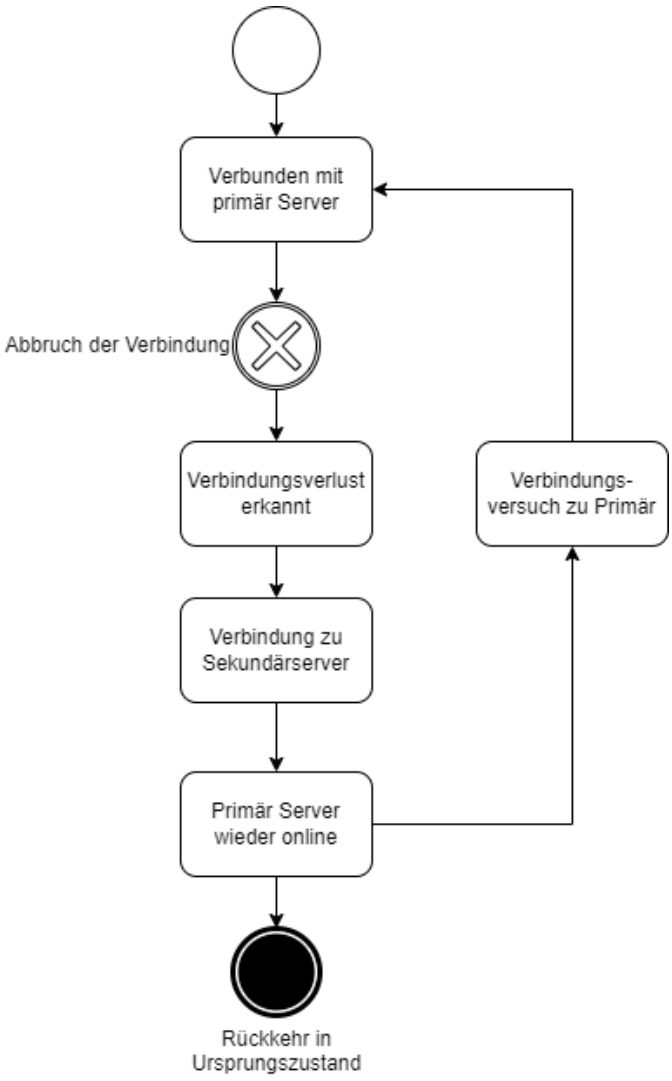

Abb. 26: Flussdiagramm der Fallback-Logik

5.8 Monitoring und Diagnose

Um den stabilen Betrieb des Mesh-Netzwerks und der Sprachkommunikation sicherzustellen, sind Überwachung und Diagnose entscheidend. Im mobilen Einsatz muss das System auch ohne zusätzliche Infrastruktur zuverlässig analysierbar sein.

Dazu stehen mehrere integrierte Werkzeuge auf jedem RPi zur Verfügung, welche Netzwerk-, Audio- und Systemzustände erfassen und analysieren können.

5.8.1 Verfügbare Monitoring-Werkzeuge

Das System nutzt ausschliesslich lokal installierte Open Source Tools.

Die wichtigsten Überwachungsbefehle sind in der folgenden Übersicht zusammengefasst:

Kategorie	Befehl / Tool	Zweck
Netzwerk	`sudo batctl n`	Zeigt erreichbare Nachbarn im Mesh
	`sudo batctl tr <IP>`	Pfadanalyse zum Zielknoten inkl. Anzahl Hops
	`sudo batctl ping <IP>`	Erreichbarkeitstest über Mesh
Audio	`arecord -l, aplay -l`	Listet erkannte Soundkarten
	`sudo alsamixer`	Pegelsteuerung, Mute/Unmute
Dienste	`systemctl status <Service des Servers>`	Statusanzeige des VoIP-Servers
	`systemctl --user status <Service des Clients>`	Statusanzeige des VoIP-Clients
System	`htop`	Anzeige der CPU- und RAM-Auslastung
	`chronyc tracking`	Analyse von Zeitquelle und Zeitdrift

Tab. 15: Werkzeuge zur System- und Netzüberwachung

5.8.2 Erstdiagnose

In der Entwicklungsumgebung kommen zusätzliche Tools wie RealVNC zum Einsatz. Diese ermöglichen die Fernanzeige der Desktop-Sitzung, um Audio- und Netzwerkstatus visuell zu prüfen. Zur Analyse in der Testumgebung wird zudem die Software OBS Studio verwendet, um den Videostream anzuzeigen.

Typische Störungen lassen sich mit wenigen Kommandos schnell eingrenzen.

Die wichtigsten Fehlerbilder und deren mögliche Ursachen sind in der folgenden Tabelle dargestellt:

Symptom	**Mögliche Ursache**
Keine Verbindung im Mesh	WLAN-Adapter, Entfernung Nachbarn
Kein Audioeingang	PTT funktioniert nicht oder deaktiviert
Kein Audioausgang	Wiedergabekanal in alsamixer gemutet
Fallback greift nicht	Entfernung Nachbarn, Status Server
Zeitabweichung	Kein Zeitserver erreichbar

Tab. 16: Typische Fehlerbilder

5.9 Zusammenführung von Hardware und Software

Die Gesamtfunktion des Systems entsteht erst durch das präzise Zusammenspiel von Hard- und Software. Jeder physische Baustein des Prototyps ist einem oder mehreren logischen Diensten zugeordnet. Diese Zuordnung bestimmt, wie Audio-, Netzwerk- und Steuerungsdaten im Betrieb verarbeitet werden und welche Abhängigkeiten zwischen einzelnen Komponenten bestehen.

5.9.1 Kopplungstabelle

Die folgende Kopplungstabelle stellt alle Systemkomponenten mit ihren zugehörigen Softwarediensten, Schnittstellen sowie den relevanten Abhängigkeiten dar.

Ebene	Hardwarekomponente	Zugeordnete Software / Dienst	Schnittstellen / Protokolle	Abhängigkeiten
Audio	WM8960 Sound-HAT	ALSA, PulseAudio, ptt.service	I2S, I2C, GPIO17	Kernelmodul aktiv
Netzwerk	USB-WLAN-Adapter	batman-adv, mesh.service	wlan1 (Ad-Hoc), bat0 (Bridge)	Channel, ESSID, MTU
Plattform	Raspberry Pi Zero 2 WH	systemd	GPIO, USB	SD-Karte, Netzversorgung
Speicher	microSD 128 GB	Raspberry Pi OS, Configfiles	–	Dateisystem
Stromversorgung	Powerbank 10 000 mAh	–	USB 5 V	Lastabhängige Abschaltung
Zeitbasis	Chrony	chrony.service	UDP 123 (lokal), RTC	Netzwerk erreichbar
VoIP	Mumble Server und Client	mumble-server.service, mumble.service	TCP 64738 (TLS)	Audio, Mesh
Steuerung	PTT-Taste am WM8960	ptt.service	GPIO17, amixer	Audio aktiv
Orchestrierung	systemd-Units	–	D-Bus, Sockets	alle Dienste

Tab. 17: Zusammenspiel HW und SW

Ein wichtiger Bestandteil ist die enge Kopplung der Hardware- und Softwarekomponenten. Die verwendeten Dienste sind modular aufgebaut, jedoch voneinander abhängig. So muss beispielsweise das Mesh-Netzwerk vor den VoIP-Diensten aktiv sein und die Audio-Treiber müssen vor der PTT-Steuerung initialisiert werden.

Mit Hilfe von systemd-Abhängigkeiten und Autostart-Skripten wird sichergestellt, dass alle Komponenten in der korrekten Reihenfolge starten.

5.9.2 Gesamtaufbau des Kommunikationssystems

Die folgende Abbildung veranschaulicht die physische und logische Kopplung der Systemkomponenten. Sie verdeutlicht die Interaktion zwischen der Hardware und Softwarediensten.

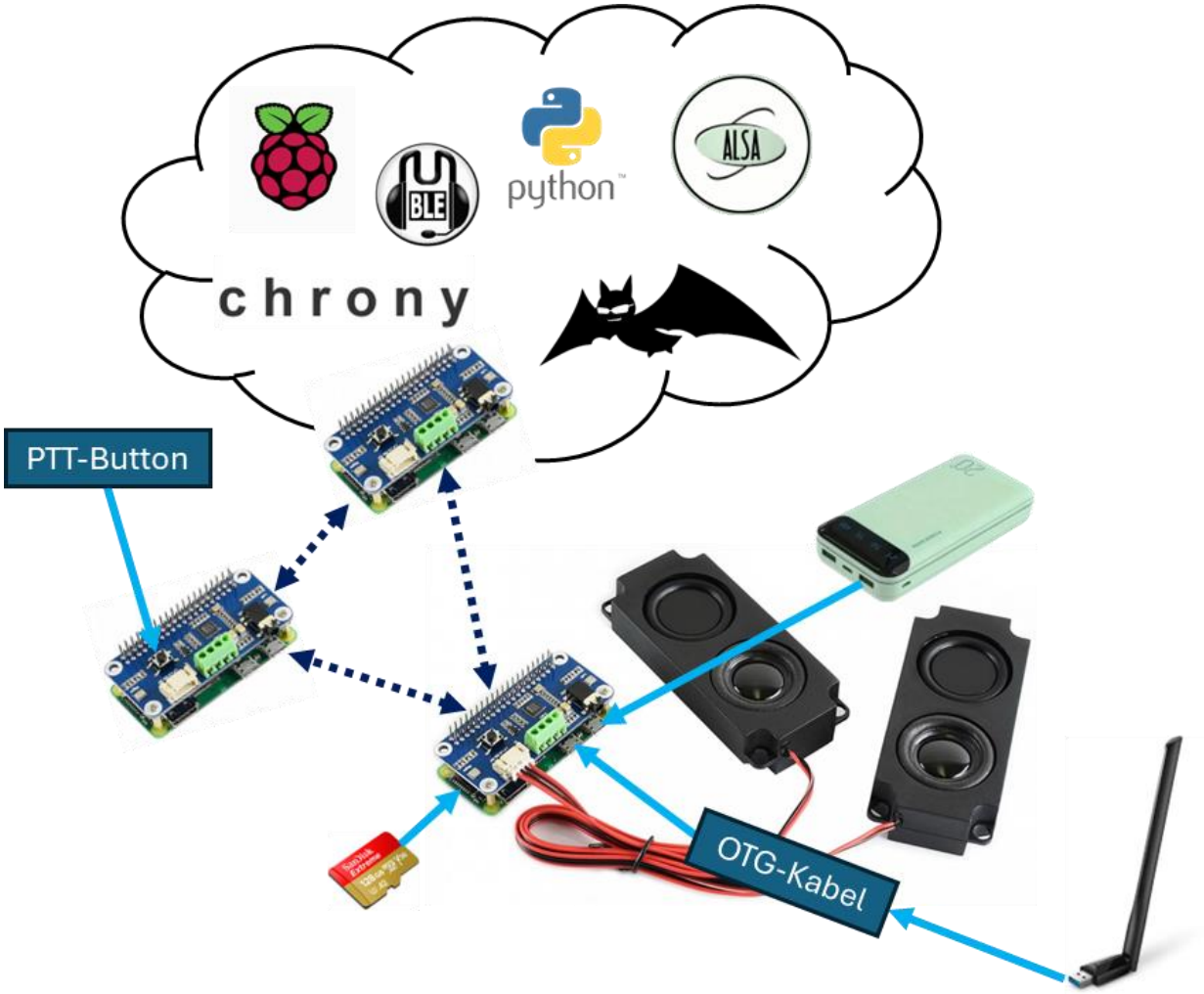

Abb. 27: Gesamtaufbau des Kommunikationssystems

5.10 Entwicklungsumgebung im Heimnetzwerk

Zur Entwicklung und sicheren Erprobung des Systems wird zu Hause ein separates Netzwerk aufgebaut. Dieses besteht aus einem zusätzlichen Router des Typs AVM Fritz!Box, der physisch mit dem bestehenden Heimnetzwerk (192.168.1.0/24) verbunden ist.

Die Fritz!Box bezieht ihre IP-Adresse dynamisch via DHCP vom Heimnetzwerk und stellt intern ein getrenntes Subnetz 172.30.5.0/24 zur Verfügung, in dem alle RPi betrieben werden. Dadurch entsteht ein abgeschottetes Netz, das unabhängig vom Heimnetz arbeitet, jedoch kontrollierten Zugriff von aussen erlaubt.

Der Zugriff auf die einzelnen RPi-Knoten erfolgt vom Entwicklungsrechner aus über:

- SSH für die Terminalsteuerung
- VNC für den grafischen Zugriff

Durch diese Struktur sind die Geräte jederzeit erreichbar, ohne direkt in das Heimnetz integriert zu sein. Die genaue Inbetriebnahme und Konfiguration der Fritz!Box ist im nächsten Kapitel dokumentiert.

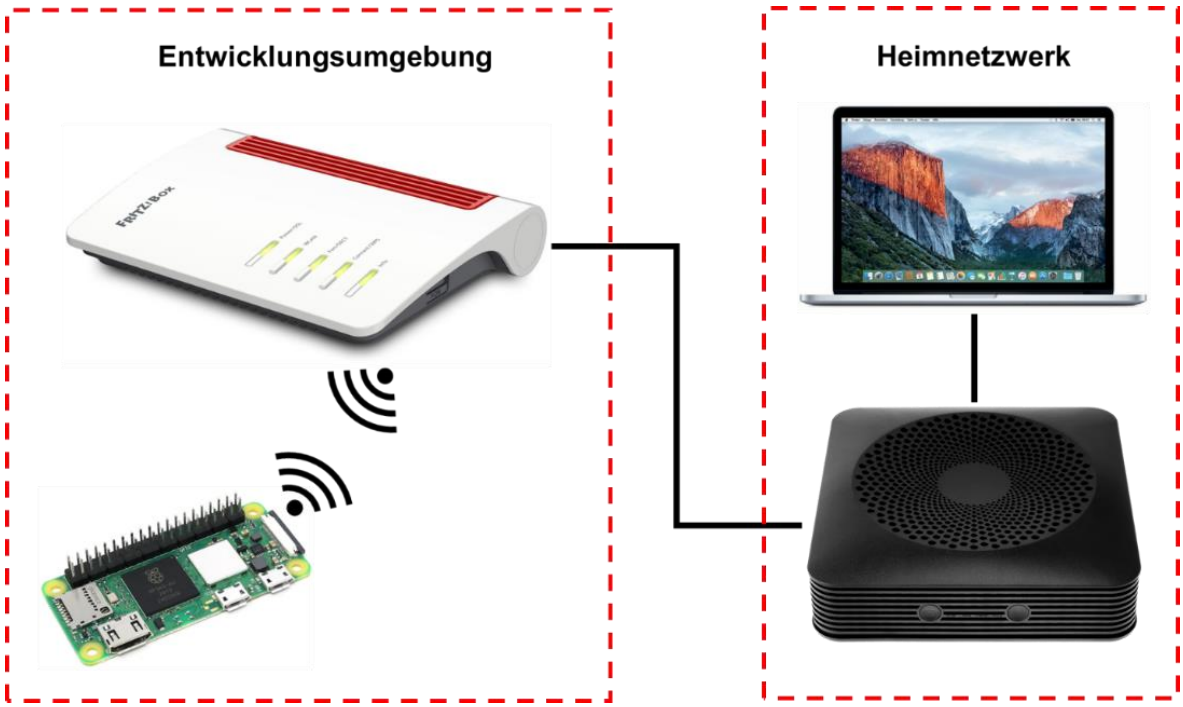

Abb. 28: Entwicklungsumgebung im Heimnetzwerk

6 Technische Umsetzung des mobilen Prototyps

6.1 Aufbau der Entwicklungsumgebung im Heimnetzwerk

Die Fritz!Box fungiert als zweiter Router bzw. kaskadierter Router, um die Entwicklungsumgebung vom Heimnetzwerk abzugrenzen. ([04] AVM, 2025)

Um einen neutralen Router bereitzustellen, wird dieser auf die Werkseinstellungen zurückgesetzt. Dafür wird ein Computer benötigt, der direkt mit dem Router per Kabel verbunden ist. Am besten wird der Anschluss LAN2 verwendet, da LAN1 später als WAN-Schnittstelle konfiguriert wird.

Es wird empfohlen, die folgenden Schritte ohne Internetverbindung durchzuführen. Das bedeutet, dass der Computer vom Netz getrennt ist und direkt mit der Fritz!Box verbunden wird. Der Grund dafür ist, dass so Probleme im Heimnetzwerk, beispielsweise durch mehrere aktive DHCP-Server, vermieden werden.

Um auf das Web-GUI des Routers zuzugreifen, kann im Webbrowser entweder *http://fritz.box* oder die Standard IP-Adresse *http://192.168.178.1* eingegeben werden. Danach erscheint das Anmeldefenster des Routers, in das man sich mit dem Passwort einloggt. Dieses befindet sich in den meisten Fällen auf der Unterseite des Routers.

Nach erfolgreicher Anmeldung kann der Router über **System → Sicherung → Werkseinstellungen** vollständig zurückgesetzt werden. Dazu wird rechts das Feld *Werkseinstellungen laden* ausgewählt und die Warnmeldung mit **OK** bestätigt.

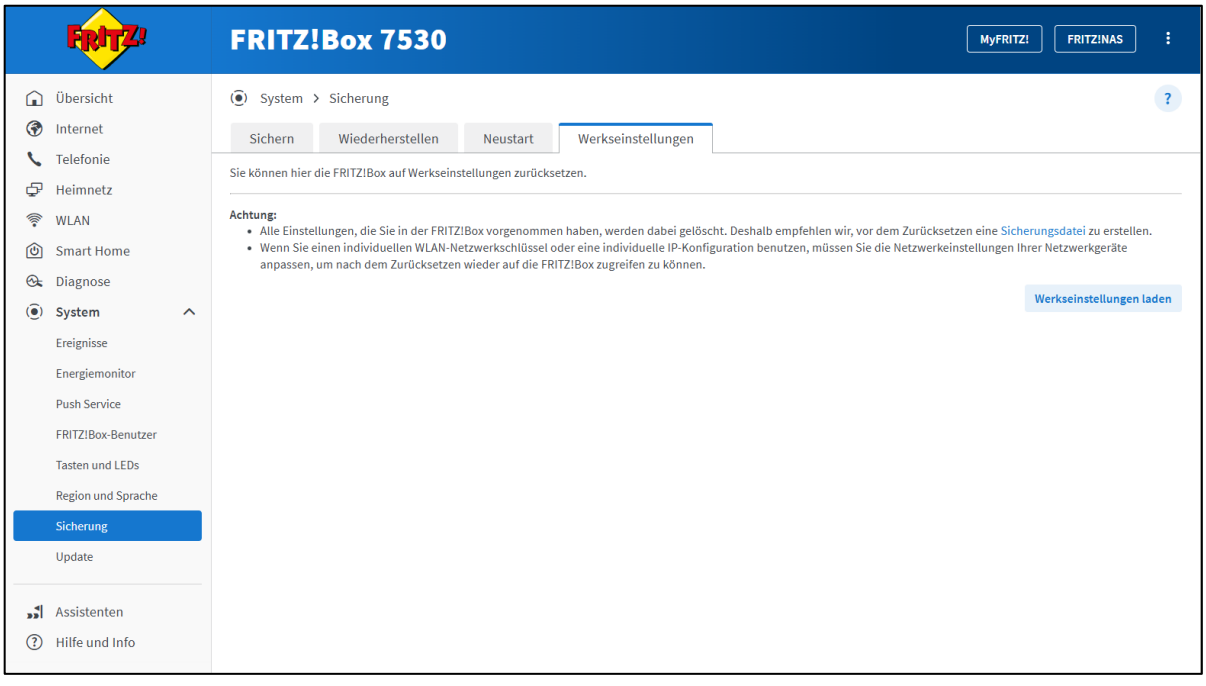

Abb. 29: Zurücksetzten der Fritz!Box

Nachdem der Router auf die Werkseinstellungen zurückgesetzt und die erneute Anmeldung erfolgreich durchgeführt wurde, muss er neu initialisiert werden.

Zuerst werden die Zugangsdaten für den Internetzugang definiert. Dazu wird im Menü **Internet** → **Zugangsdaten** → **Internetzugang** navigiert. Da sich der Router innerhalb eines bestehenden Heimnetzwerks befindet und nicht über ein klassisches WAN, sondern über ein LAN-Kabel angeschlossen wird, wird im Dropdown-Menü der Internetanbieter *vorhandener Zugang über LAN* ausgewählt.

In diesem Modus bezieht der Router seine IP-Adresse automatisch per DHCP, in diesem Fall aus dem Adressbereich 192.168.1.0/24. Die Auswahl wird anschliessend mit **Übernehmen** unten rechts bestätigt.

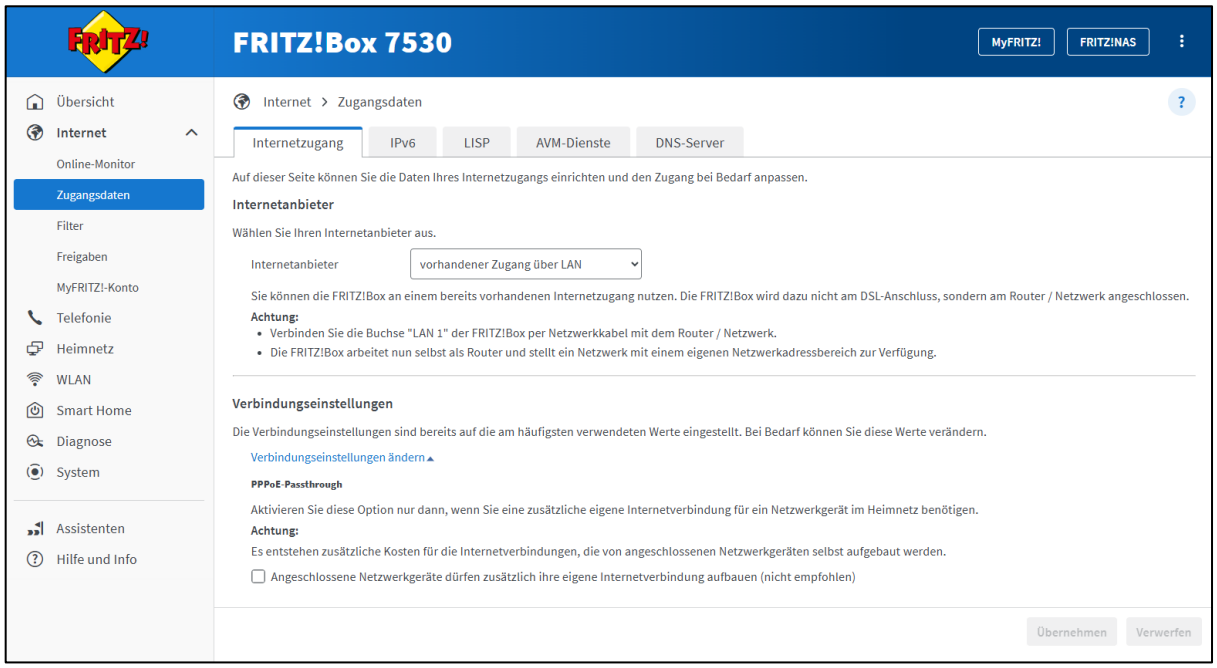

Abb. 30: Zugangsdaten Fritz!Box

Als Nächstes wird ein neuer IP-Bereich definiert. Für die Entwicklungsumgebung wird der private Adressbereich **172.30.5.0/24** festgelegt, da die RPi in diesem Adressbereich konfiguriert werden.

Zur Konfiguration wird im Menü **Heimnetz** → **Netzwerk** → **Netzwerkeinstellungen** navigiert. Anschliessend scrollt man nach unten und klickt auf *weitere Einstellungen*, damit der Reiter IP-Adressen sichtbar wird. Dort können über die Schaltfläche *IPv4-Einstellungen* die Netzadresse, Subnetzmaske und der DHCP-Bereich angepasst werden.

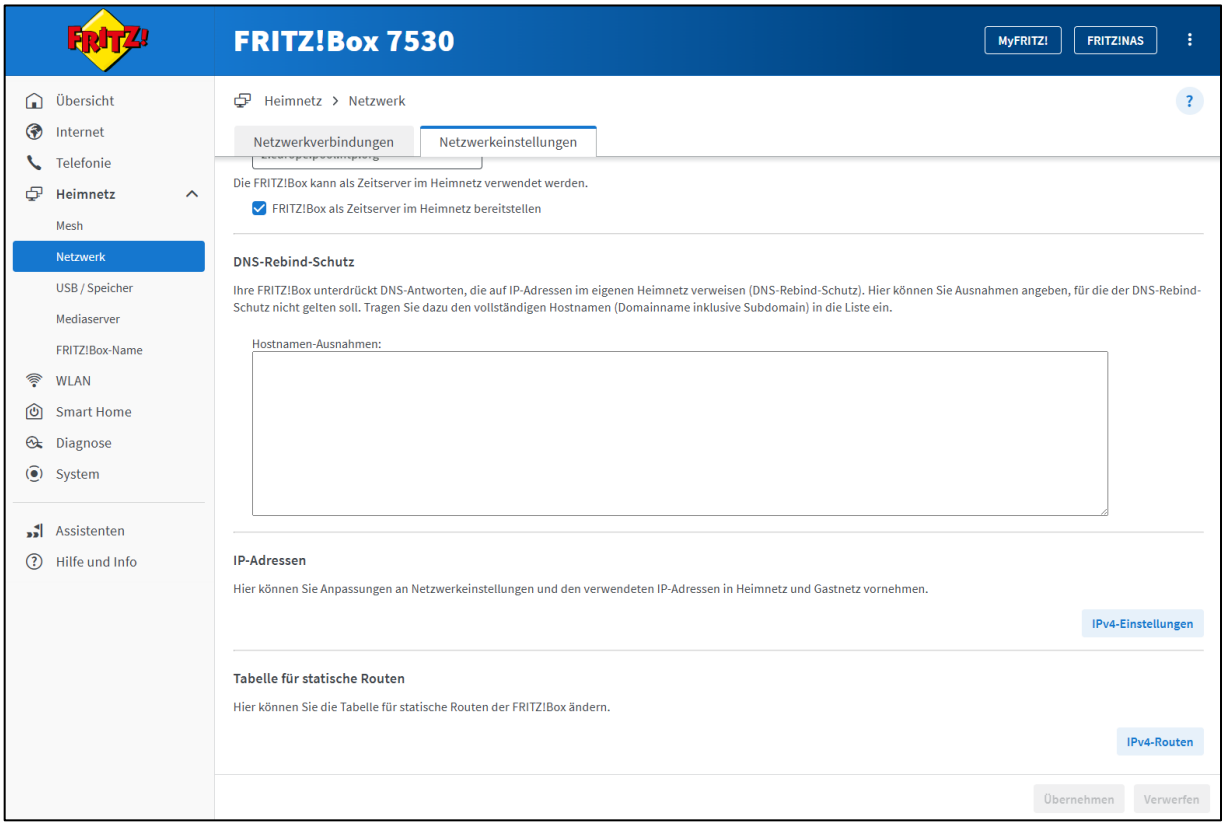

Abb. 31: Netzwerkeinstellungen IPv4

Folgende Konfigurationen werden vorgenommen:

- IPv4-Adresse des Routers: 172.30.5.1
- Subnetzmaske: 255.255.255.0

Anschliessend wird der DHCP-Server aktiviert, indem das Kontrollkästchen *DHCP-Server aktivieren* aktiviert wird. Danach wird der DHCP-Bereich definiert.

Um direkt angeschlossene Geräte automatisch mit einer IP-Adresse zu versorgen, wird der DHCP-Bereich 172.30.5.100 – 172.30.5.200 gewählt und aktiviert. Wird der DHCP-Server nicht aktiviert, muss jedem angeschlossenen Computer manuell eine statische IPv4-Adresse aus dem definierten IP-Bereich zugewiesen werden. Durch die Angabe einer festen IP-Adresse für den DNS-Server wird sichergestellt, dass alle Geräte in der Entwicklungsumgebung die Fritz!Box als zentralen Namensauflöser verwenden.

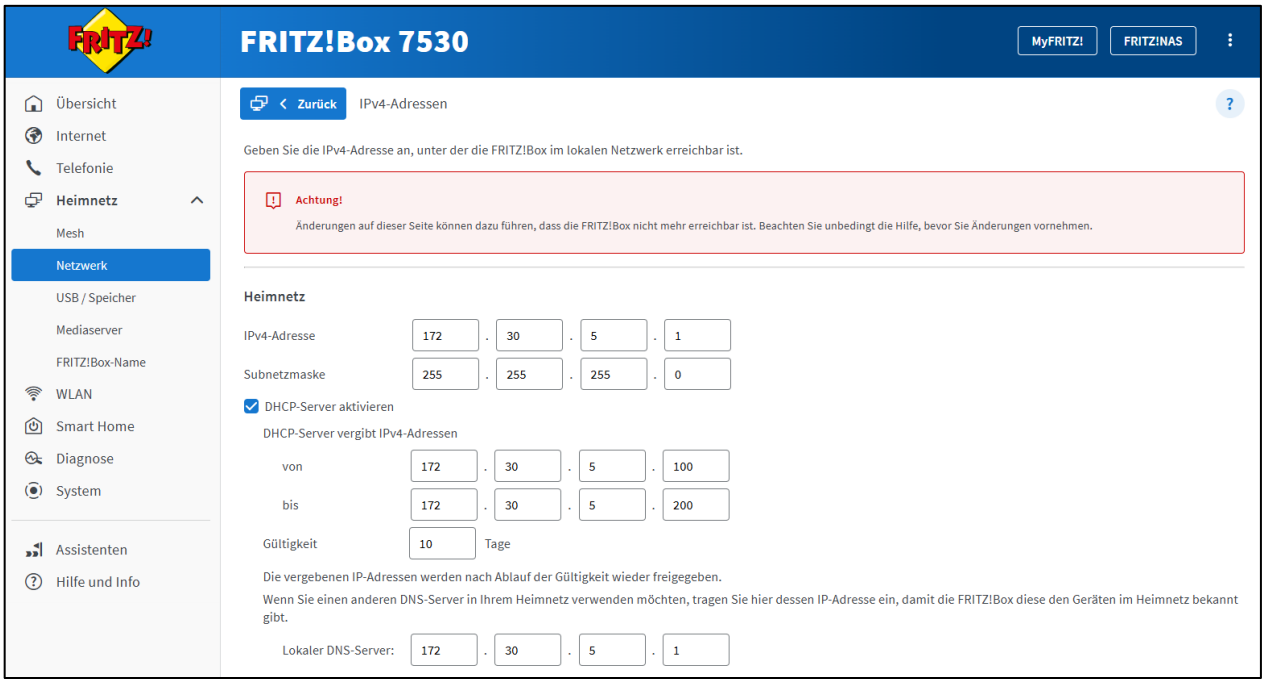

Abb. 32: IPv4 Konfiguration Fritz!Box

Die vorgenommenen Einstellungen werden unten auf der Seite mit einem Klick auf **Übernehmen** gespeichert. **Achtung:** Nach dem Speichern wird die bestehende Verbindung automatisch getrennt, da der Router dann nicht mehr über die Standardadresse 192.168.178.1, sondern über die neue Adresse 172.30.5.1 erreichbar ist.

Nach erfolgreicher Konfiguration verbindet man den Computer mit dem WLAN der Fritz!Box, indem die entsprechende SSID ausgewählt und das WLAN-Passwort eingegeben wird. Dieses befindet sich in den meisten Fällen auch auf der Unterseite des Routers. Anschliessend kann überprüft werden, ob die Einrichtung korrekt durchgeführt wurde:

- Hat das verbundene Gerät eine IP-Adresse im definierten Bereich 172.30.5.100 – 172.30.5.200 erhalten?
- Ist der Zugriff auf das Web-GUI des Routers unter http://172.30.5.1/ gewährleistet?

Wenn beide Fragen mit Ja beantwortet werden können, ist die Einrichtung der Entwicklungsumgebung im Heimnetzwerk abgeschlossen.

6.2 Inbetriebnahme Raspberry Pi

6.2.1 Vorbereitungen

Bevor der RPi genutzt werden kann, wird eine microSD-Karte benötigt, auf der ein Betriebssystem installiert ist. Um das OS auf der microSD-Karte zu installieren, benötigt man:

- einen Computer, mit dem die microSD als Bootmedium beschrieben werden kann
- einen Kartenleser oder Adapter, um die microSD mit dem Computer zu verbinden

Für die Installation wird die Software *Raspberry Pi Imager* empfohlen. ([05] Raspberry Pi, 2025)

Darüber hinaus ist es empfehlenswert, auf dem Entwicklungscomputer zusätzlich den RealVNC Viewer zu installiert. So ist nach der Grundkonfiguration eine grafische Verbindung zu den RPi möglich.

6.2.2 Download RPi Imager

Der Imager kann auf zwei Arten installiert werden:

- über die offizielle Webseite https://www.raspberrypi.com/software/
- direkt über den Paketmanager mit: `sudo apt install rpi-imager`

Nach der Installation wird die Anwendung gestartet, entweder über das Programmsymbol oder im Terminal mit dem Befehl `rpi-imager`

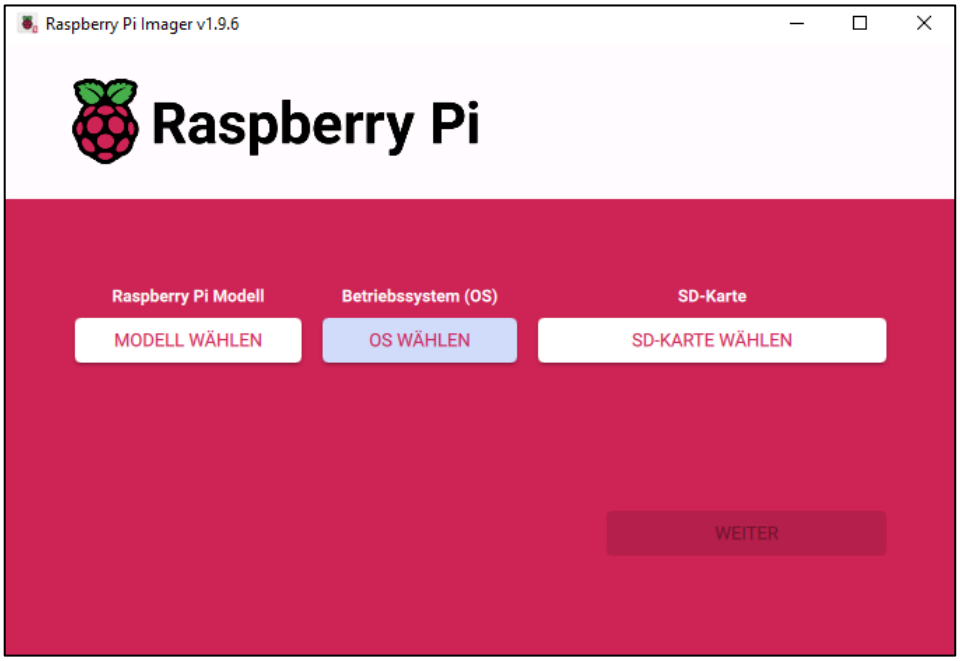

Abb. 33: Raspberry Pi Imager

6.2.3 Schreiben der microSD

Nachdem die microSD an den Computer angeschlossen wurde und der Imager gestartet ist, werden folgende Auswahlen getroffen:

MODELL WÄHLEN:

Das entsprechende Modell auswählen, hier *Raspberry Pi Zero 2 W*

OS WÄHLEN:

Das Betriebssystem auswählen, *Raspberry Pi OS (32-Bit with Desktop)*

SD-KARTE WÄHLEN:

Die angeschlossene microSD-Karte auswählen *(max. 256GB)*

ACHTUNG: Alle Daten auf der microSD werden überschrieben. Vorher sicherstellen, dass sich keine wichtigen Daten auf der Karte befinden.

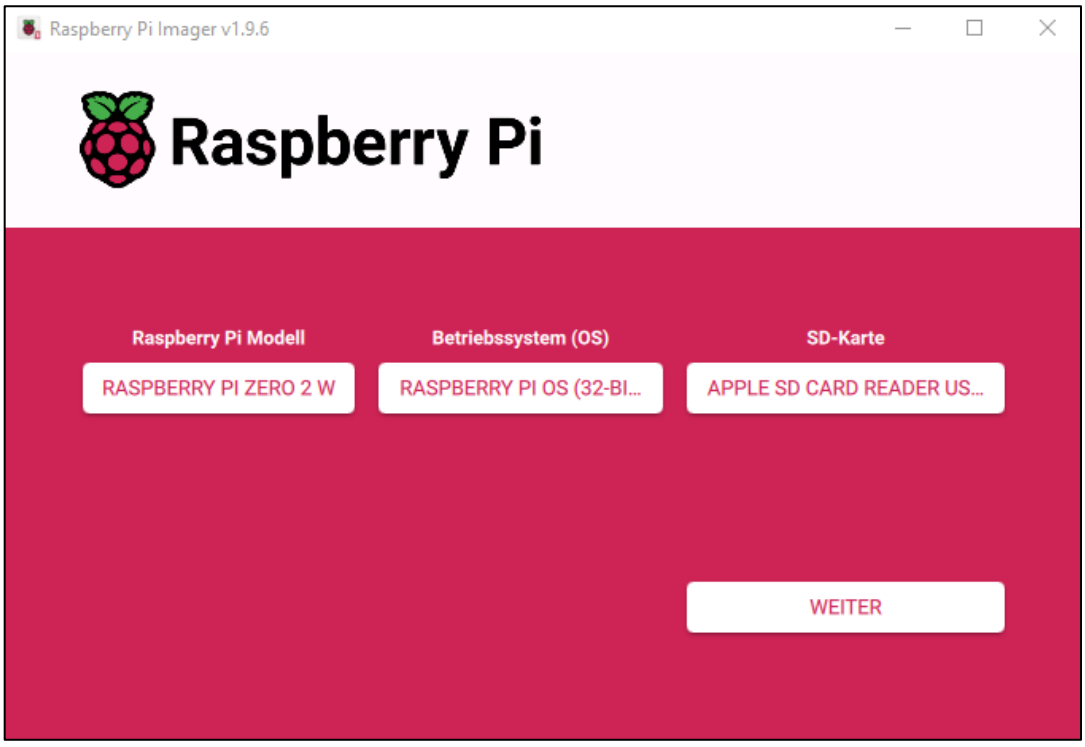

Abb. 34: SD schreiben Imager

Nachdem alle grundlegenden Auswahlen getroffen wurden, wird mit **Weiter** fortgefahren. Anschliessend erscheint ein Popup-Fenster, in dem man aufgefordert wird, die Betriebssystemanpassungen vorzunehmen.

Über die Schaltfläche **Einstellungen bearbeiten** können verschiedene Konfigurationen bereits vor dem Schreiben der microSD-Karte vorgenommen werden. Dadurch wird die Inbetriebnahme der RPi später vereinfacht.

Im Reiter *Allgemein* werden folgende Einstellungen konfiguriert:

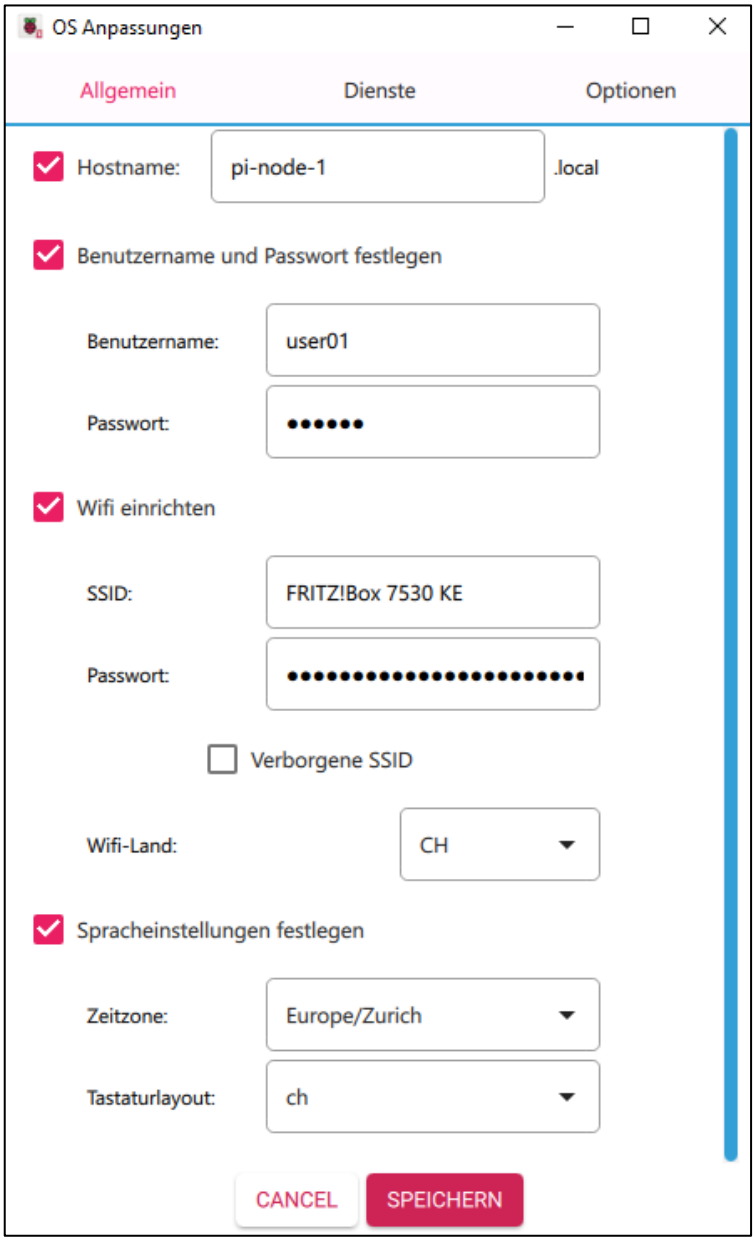

Abb. 35: OS Anpassungen Imager

Im Reiter *Dienste* können zusätzliche Funktionen aktiviert werden, die für den späteren Betrieb des RPi nützlich sind. Hier wird insbesondere der SSH-Dienst aktiviert, um den RPi ohne Monitor und Tastatur aus der Ferne steuern zu können.

Für die Authentifizierung stehen zwei Optionen zur Verfügung:

- Passwort zur Authentifizierung verwenden (Standard, einfach einzurichten)
- Authentifizierung via Public-Key

Da die Geräte in der Entwicklungsumgebung betrieben werden, wird die Variante mit Passwort genutzt.

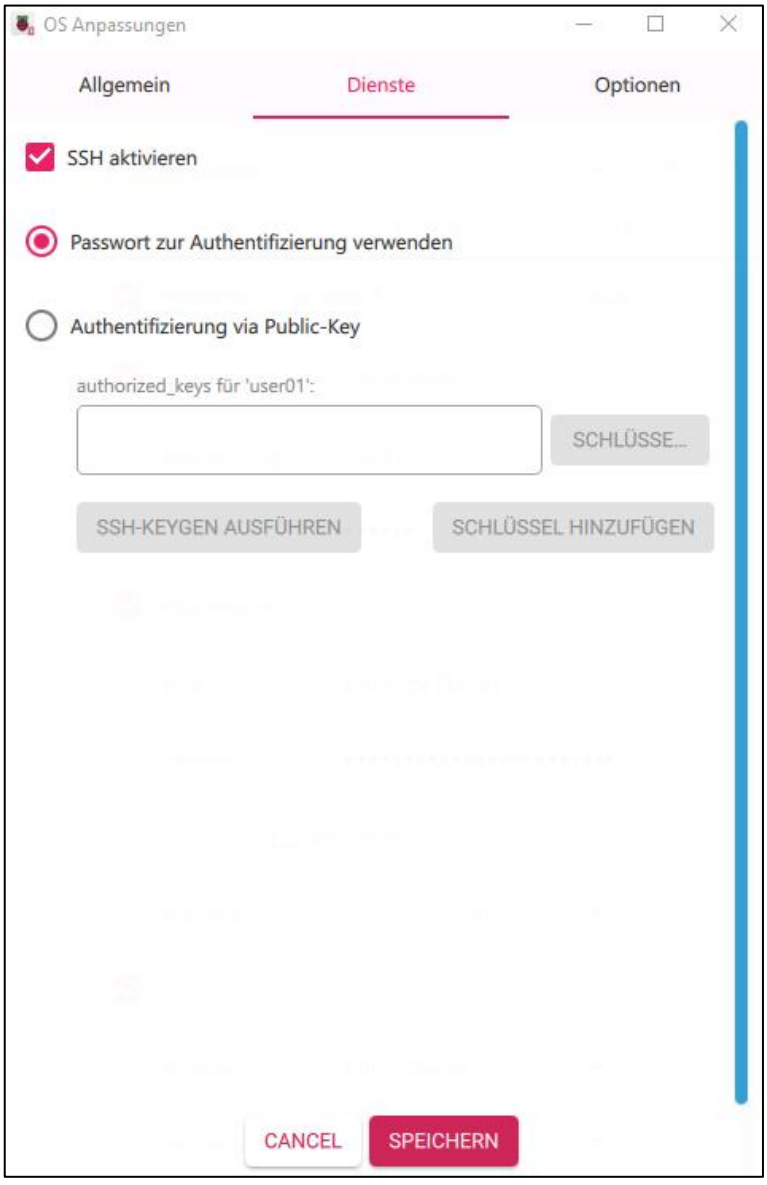

Abb. 36: OS Anpassungen Dienste

Im Reiter *Optionen* werden zusätzliche Einstellungen vorgenommen, die den Schreibvorgang erleichtern. Für die Installation wird aktiviert, dass nach Abschluss des Vorgangs ein Tonsignal abgespielt und das Medium automatisch ausgeworfen wird. Die Option Telemetry aktivieren bleibt deaktiviert.

Sind alle Anpassungen in den drei Registern vorgenommen, wird mit **Speichern** bestätigt. Im vorherigen Popup-Fenster kann nun mit **Ja** ausgewählt werden, dass die Konfigurationen übernommen werden.

Anschliessend erscheint ein weiteres Popup-Fenster mit der Warnung, dass alle vorhandenen Daten auf der microSD-Karte gelöscht werden. Dieser Hinweis muss ebenfalls mit **Ja** bestätigt werden, bevor der Schreibvorgang beginnt.

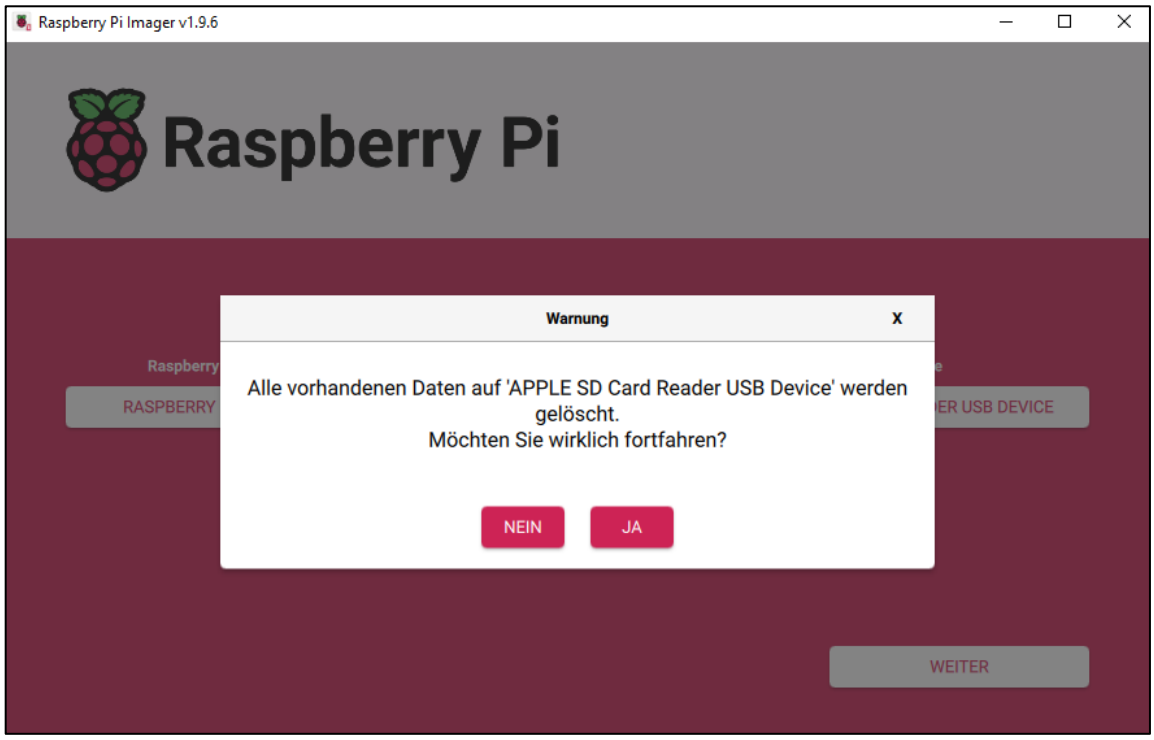

Abb. 37: Popup Warnung Imager

Nach erfolgreichem Abschluss des Schreibvorgangs erscheint eine Bestätigungsmeldung, dass das ausgewählte Betriebssystem auf die microSD-Karte geschrieben wurde. Diese kann abschliessend mit **Weiter** bestätigt werden. Anschliessend kann die Karte aus dem Lesegerät entfernt werden.

Damit ist die microSD vorbereitet und einsatzbereit. Im nächsten Schritt wird sie in den RPi eingesetzt, um die Inbetriebnahme zu starten.

6.2.4 Erster Start Raspberry Pi

Bevor auf den RPi zugegriffen werden kann, verbindet sich der Entwicklungscomputer zunächst mit dem WLAN der Entwicklungsumgebung. Nur so ist gewährleistet, dass eine direkte Netzwerkverbindung zu den RPi besteht.

Nachdem die vorbereitete microSD-Karte in den RPi eingesetzt wurde, wird dieser über das Micro-USB Netzteil mit Strom versorgt. Beim ersten Start dauert der Bootvorgang etwas länger, da das System erste Konfigurationen durchführt.

Der erfolgreiche Start ist am Blinkmuster der Status-LED erkennbar. Ein gleichmässiges, wiederholtes Blinken signalisiert, dass das Betriebssystem geladen wird. Nach kurzer Zeit ist der RPi betriebsbereit und sollte sich automatisch mit dem vorkonfigurierten WLAN verbinden.

Der erste Zugriff erfolgt per SSH vom Entwicklungscomputer mit den im Raspberry Pi Imager definierten Zugangsdaten. Hierfür wird **PowerShell** (PS) verwendet.

Der Befehl im PS lautet: `ssh user01@pi-node-1.local`

Anschliessend muss das zuvor definierte Passwort eingegeben und die Sicherheitsabfrage mit **yes** bestätigt werden. Dadurch wird der SSH-Fingerprint dauerhaft auf dem Entwicklungscomputer gespeichert und zukünftige Verbindungen können ohne erneute Bestätigung aufgebaut werden.

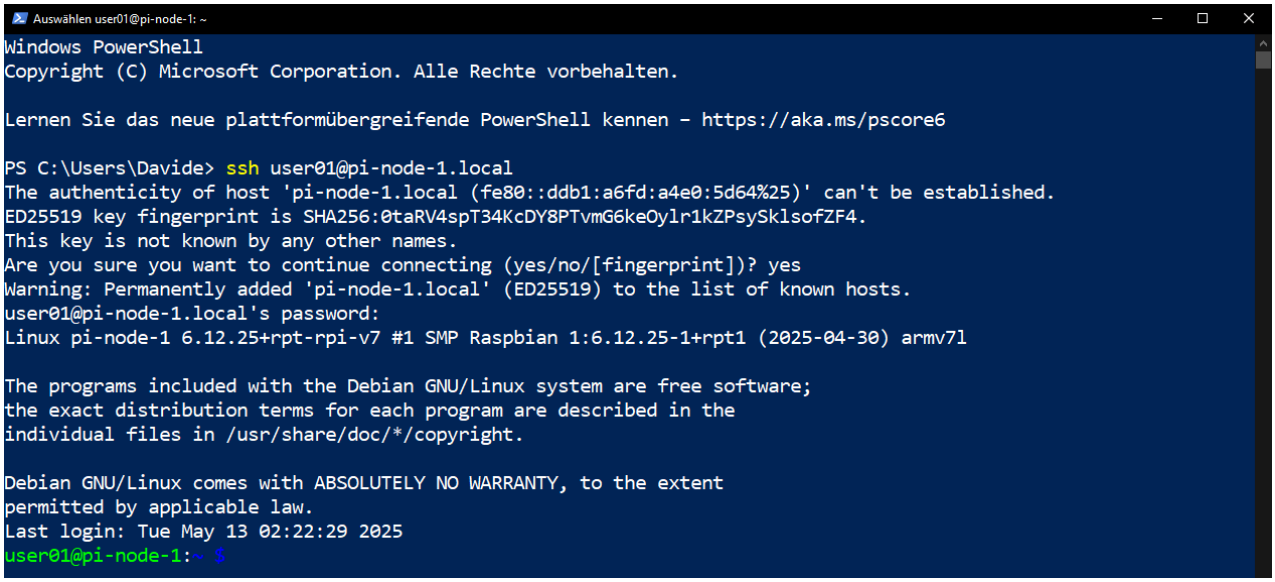

Abb. 38: SSH-Verbindung mit PowerShell

6.2.4.1 Aktivieren des VNC-Servers

Im nächsten Schritt wird der VNC-Server auf dem RPi aktiviert. Damit steht später ein grafischer Fernzugriff über den RealVNC Viewer zur Verfügung. Dies erfolgt über den Raspberry Pi Konfigurator **raspi-config**, der im Terminal mit folgendem Befehl gestartet wird:

- `sudo raspi-config`

Im Menü wird unter **3 Interface Options** → **I3 VNC** der VNC-Server aktiviert. Auf die Nachfrage, ob der Server aktiviert werden soll, wird mit **yes** bestätigt. Nach einer kurzen Wartezeit von rund *30 Sekunden* erscheint eine Meldung, dass der VNC-Server erfolgreich aktiviert wurde. Ab diesem Zeitpunkt startet der VNC-Dienst bei jedem Systemstart automatisch und steht für den Fernzugriff bereit.

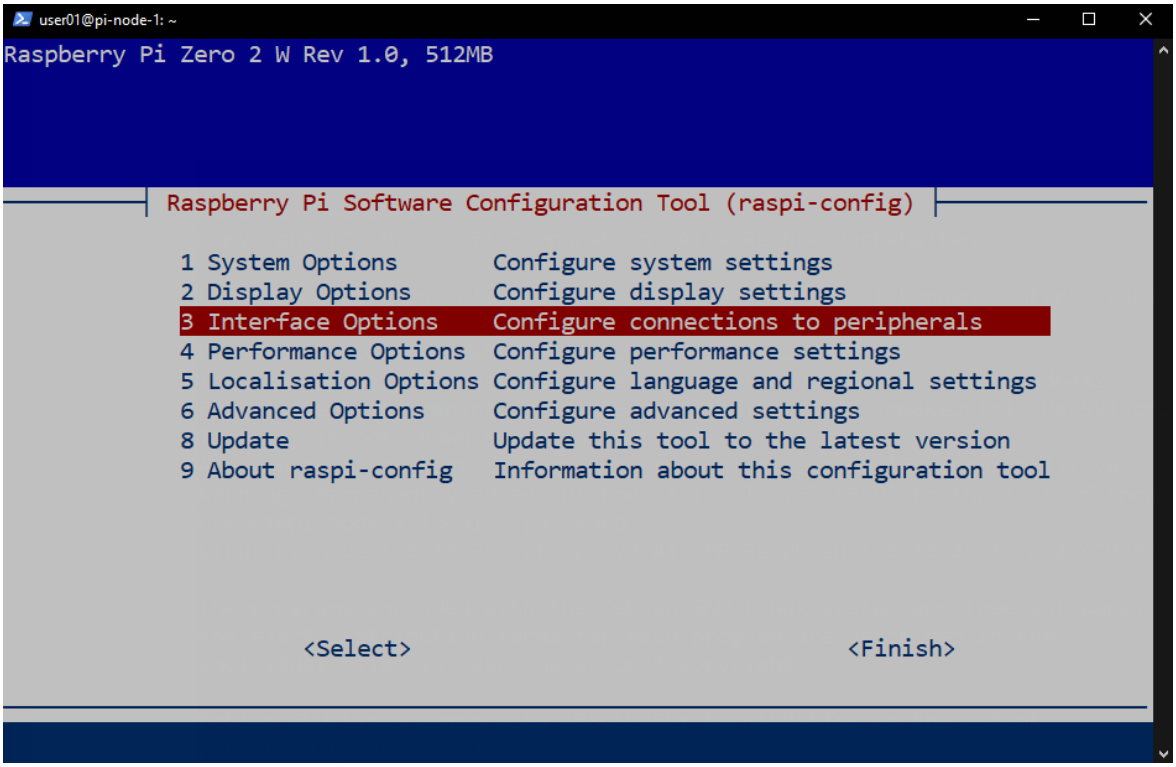

Abb. 39: Aktivierung VNC-Server

6.2.4.2 Vergabe einer festen IP-Adresse via SSH

Standardmässig erhält der RPi seine IP-Adresse per DHCP vom Router der Entwicklungsumgebung. Damit die Geräte jedoch dauerhaft unter derselben Adresse erreichbar sind, wird eine feste IPv4-Adresse konfiguriert. Dies erfolgt über den **NetworkManager**, der seit Raspberry Pi OS Bookworm standardmässig die Netzwerkkonfiguration verwaltet. Zunächst werden die vorhandenen Netzwerkverbindungen mit folgendem Befehl angezeigt:

- `nmcli connection show`

Die aktive WLAN-Verbindung trägt in dieser Umgebung den Namen **preconfigured**. Um eine feste IP-Adresse zu vergeben, wird dieser Name im folgenden Befehl verwendet:

- `sudo nmcli connection modify preconfigured ipv4.method manual ipv4.addresses 172.30.5.10/24 ipv4.gateway 172.30.5.1 ipv4.dns 172.30.5.1`

Damit werden die Netzwerkeinstellungen wie folgt gesetzt:

- IP-Adresse: 172.30.5.10
- Subnetzmaske: /24 (255.255.255.0)
- Gateway: 172.30.5.1
- DNS: 172.30.5.1

Die Änderungen werden mit folgendem Befehl aktiv:

- `sudo nmcli connection up preconfigured`

Anschliessend kann mit dem Befehl `ip a` überprüft werden, ob die neue Adresse korrekt übernommen wurde. Um sicherzustellen, dass die Konfiguration auch nach einem Neustart erhalten bleibt, empfiehlt es sich, den RPi neu zu starten mit `sudo reboot`.

```
user01@pi-node-1:~ $ nmcli connection show
NAME          UUID                                  TYPE      DEVICE
preconfigured  e24076ce-0822-45b3-9695-eaee46c06dbb  wifi      wlan0
lo             d0412bfa-f3d7-4db1-8615-22cf26b574d4  loopback  lo
user01@pi-node-1:~ $
user01@pi-node-1:~ $ sudo nmcli connection modify preconfigured ipv4.method manual ipv4.addresses 172.30.5.10/24
 ipv4.gateway 172.30.5.1 ipv4.dns 172.30.5.1
user01@pi-node-1:~ $ sudo nmcli connection up preconfigured
Connection successfully activated (D-Bus active path: /org/freedesktop/NetworkManager/ActiveConnection/3)
user01@pi-node-1:~ $
user01@pi-node-1:~ $
user01@pi-node-1:~ $
user01@pi-node-1:~ $ sudo reboot

Broadcast message from root@pi-node-1 on pts/1 (Sun 2025-09-07 18:30:00 CEST):

The system will reboot now!

user01@pi-node-1:~ $ Connection to pi-node-1.local closed by remote host.
```

Abb. 40: Vergabe feste IP-Adresse

6.2.4.3 System aktualisieren

Nachdem die feste IP-Adresse erfolgreich gesetzt und der RPi neu gestartet wurde, erfolgt die erneute Verbindung per SSH nun nicht mehr über den Hostnamen, sondern direkt über die vergebene IP-Adresse.

- `ssh user01@172.30.5.10`

Im nächsten Schritt wird das Betriebssystem auf den aktuellen Stand gebracht. Dies erfolgt mit dem folgenden Befehl:

- `sudo apt update && sudo apt upgrade -y`

Damit werden die Paketlisten aktualisiert und alle installierten Pakete auf die neueste Version gebracht. Dieser Vorgang kann je nach Geschwindigkeit der Internetverbindung und Grösse der Updates einige Minuten dauern.

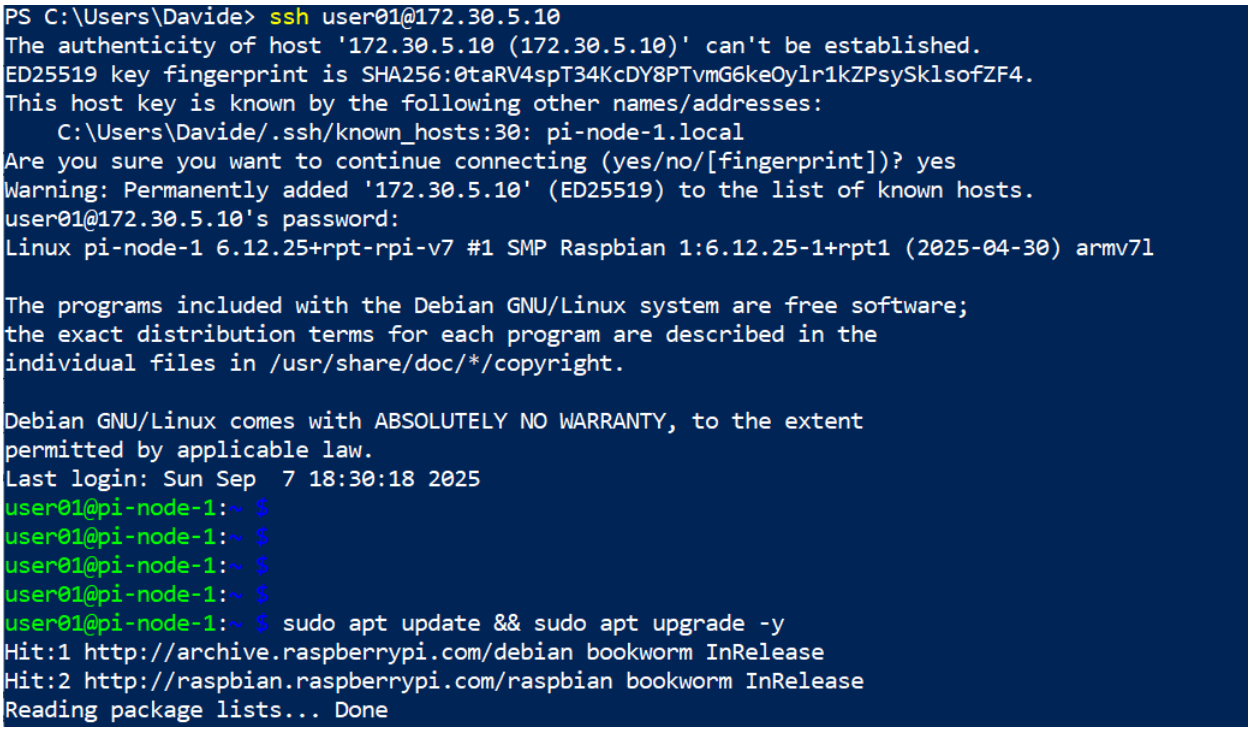

Abb. 41: RPi via SSH aktualisieren

Falls während eines Updates die Meldung erscheint, dass die Konfigurationsdatei **initramfs.conf** angepasst wurde, kann diese mit **Y** bestätigt werden. Dadurch bleibt das System auf dem aktuellen Stand und erhält die empfohlenen Standardwerte.

6.2.4.4 VNC-Verbindung testen

Nach erfolgreicher Aktualisierung kann der zuvor aktivierte VNC-Server getestet werden. Dazu wird auf dem Entwicklungscomputer der **RealVNC Viewer** gestartet und im oberen Suchfeld die feste IP-Adresse des RPi eingegeben und mit *Enter* bestätigt.

Abb. 42: Eingabe IP im RealVNC Viewer

Daraufhin erscheint eine Meldung zur **Identitätsprüfung**, die mit **Fortsetzen** bestätigt wird. Anschliessend öffnet sich das Authentifizierungsfenster, in dem Benutzername und Passwort eingegeben werden. Optional kann das Passwort gespeichert werden, sodass es bei erneuten Verbindungen nicht jedes Mal eingegeben werden muss.

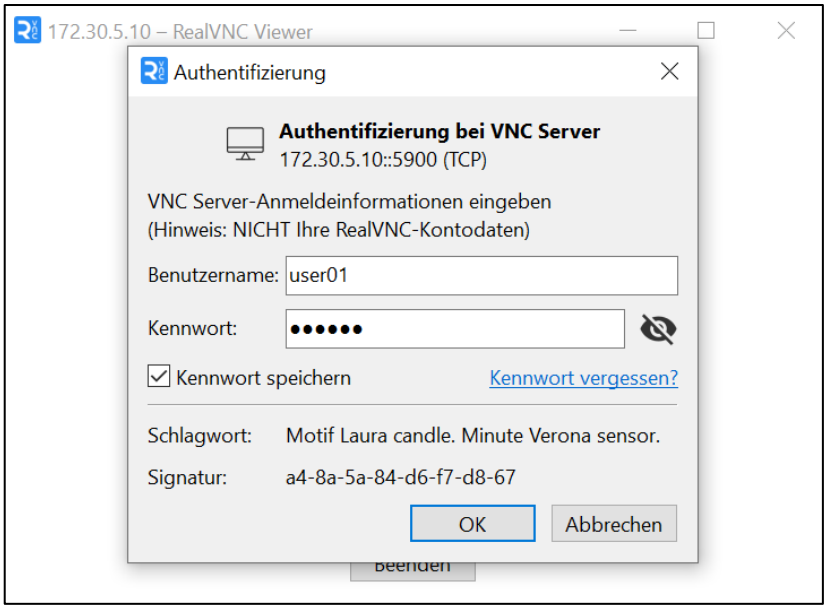

Abb. 43: Authentifizierung bei VNC-Server

Wenn die Anmeldung erfolgreich ist, öffnet sich die grafische Benutzeroberfläche des RPi im Viewer.

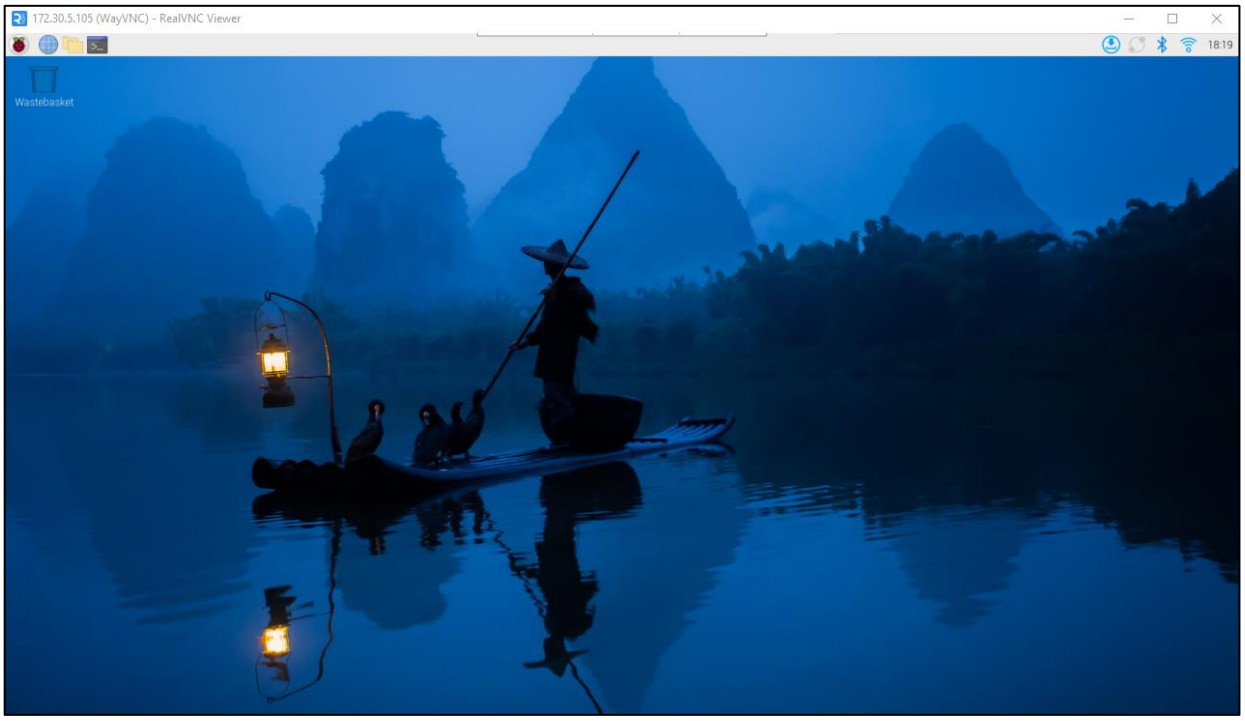

Abb. 44: GUI Raspberry Pi

Damit ist die Grundkonfiguration erfolgreich abgeschlossen und der Raspberry Pi ist sowohl per SSH als auch per VNC zuverlässig erreichbar. Auf dieser Grundlage können nun die weiteren Softwarekomponenten installiert und die Integration in das geplante Test- und Mesh-Netzwerk vorbereitet werden.

Die beschriebenen Schritte sind identisch und werden in gleicher Weise auch für die beiden weiteren RPi durchgeführt, wobei sich lediglich die Hostnamen und IP-Adressen unterscheiden:

- **pi-node-2** mit der IP-Adresse **172.30.5.20**
- **pi-node-3** mit der IP-Adresse **172.30.5.30**

6.2.5 Portfreigabe auf dem Router (optional)

Um bei Bedarf auch aus dem Heimnetzwerk auf die Geräte der Entwicklungsumgebung zugreifen zu können, wird auf der Fritz!Box eine Portfreigabe eingerichtet. Dadurch lassen sich die RPi Zero 2 WH nicht nur innerhalb des Subnetzes 172.30.5.0/24, sondern auch direkt aus dem Heimnetz 192.168.1.0/24 erreichen.

Im Folgenden wird die Freigabe für **SSH-Port 22** auf dem *pi-node-1* beschrieben:

1. Als Erstes folgt die Anmeldung an der Fritz!Box unter http://172.30.5.1
2. Im Menü **Internet** → **Freigaben** → **Portfreigaben** den Button *Gerät für Freigaben hinzufügen* auswählen

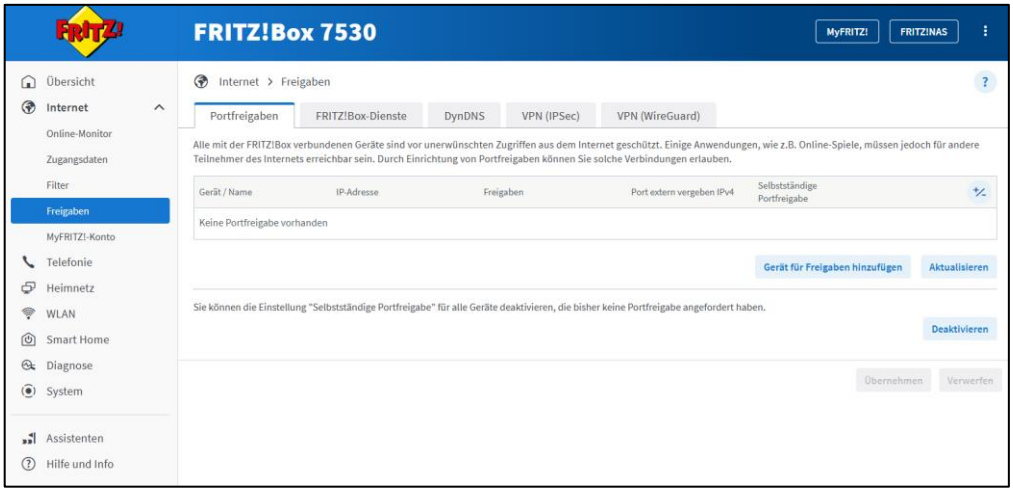

Abb. 45: Übersicht Portfreigaben

3. Das Gerät **pi-node-1** mit der internen IP-Adresse 172.30.5.10 auswählen. Anschliessend im unteren Bereich rechts auf *Neue Freigabe* klicken

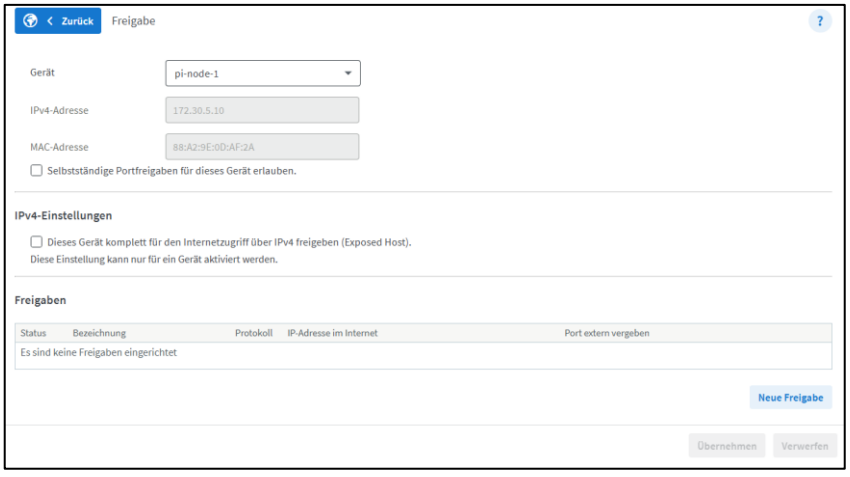

Abb. 46: Detailansicht pi-node-1

4. In der Maske **Freigabe anlegen** wird die Portfreigabe erstellt

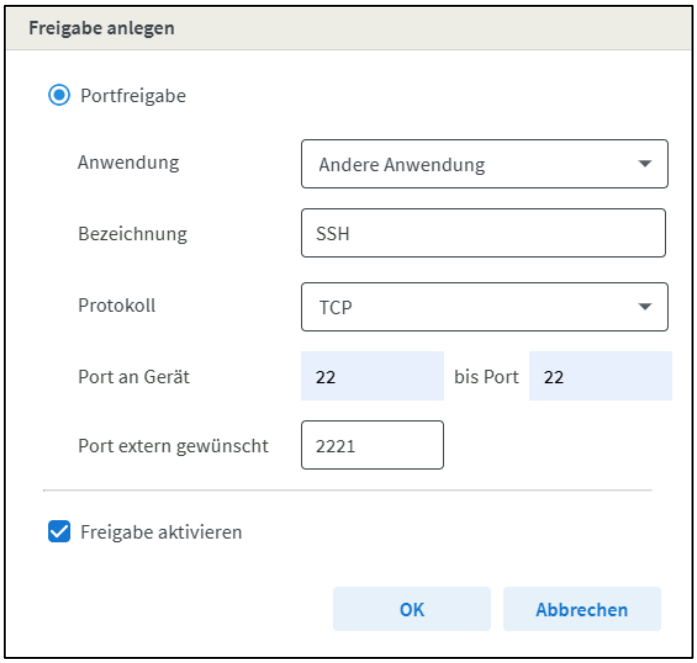

Abb. 47: Portfreigabe für SSH auf pi-node-1

5. Nach dem Speichern erscheint die eingerichtete Portfreigabe in der Übersichtstabelle. Dort ist erkennbar, dass *pi-node-1* über die Fritz!Box unter dem externen Port **2221** per SSH erreichbar ist.

Abb. 48: Übersicht mit allen Portfreigaben

Test der Portfreigabe

Damit die Portfreigabe genutzt werden kann, muss zunächst die IP-Adresse der Fritz!Box im Heimnetzwerk bekannt sein, diese erhält der Router per DHCP. Die Adresse kann entweder direkt in der Benutzeroberfläche des Heimrouters oder über einen Netzwerk-Scan ermittelt werden.

Anschliessend kann die Verbindung auf dem Entwicklungscomputer aus dem Heimnetzwerk getestet werden. Für das Beispiel *pi-node-1* lautet der Befehl:

- `ssh user01@192.168.1.152 -p 2221`

Damit wird die Verbindung zur Fritz!Box aufgebaut, die die Anfrage an *pi-node-1* auf **172.30.5.10:22** in der Testumgebung weiterleitet.

Um eine Übersicht zu schaffen, sind in der folgenden Tabelle alle drei RPi mit ihren internen IP-Adressen sowie den zugehörigen externen Ports für SSH und VNC aufgeführt. Somit ist ersichtlich, wie die Verbindung aus dem Heimnetzwerk hergestellt werden kann.

Node	IP-Adresse	Dienst	Interner Port	Externer Port	Verbindung via PS
pi-node-1	172.30.5.10	SSH	22	2221	ssh user01@192.168.1.152 -p 2221
pi-node-2	172.30.5.20	SSH	22	2222	ssh user02@192.168.1.152 -p 2222
pi-node-3	172.30.5.30	SSH	22	2223	ssh user03@192.168.1.152 -p 2223
pi-node-1	172.30.5.10	VNC	5900	5901	RealVNC → 192.168.1.152:5901
pi-node-2	172.30.5.20	VNC	5900	5902	RealVNC → 192.168.1.152:5902
pi-node-3	172.30.5.30	VNC	5900	5903	RealVNC → 192.168.1.152:5903

Tab. 18: PowerShell Befehle für Zugriff

Diese Konfiguration ist **optional** und dient ausschliesslich dem vereinfachten Management. Für den produktiven Betrieb bleibt die Testumgebung weiterhin isoliert.

6.3 Inbetriebnahme Waveshare WM8960

Für die Sprachaufnahme und -wiedergabe wird der Waveshare WM8960 Hi-Fi Stereo Sound HAT eingesetzt. Dieses Modul stellt die Audio-Schnittstelle bereit und wird über die GPIO-Pins des RPi Zero 2 WH angebunden.

Hinweis vor der Inbetriebnahme: Mit den Befehlen `aplay -l` und `arecord -l` lassen sich die aktuell erkannten Audio-Geräte anzeigen. Auf einem RPi ohne Sound-HAT wird bei `aplay -l` üblicherweise nur der HDMI-Ausgang angezeigt und bei `arecord -l` erscheint in diesem Fall kein Eintrag, da ohne HAT kein Aufnahmegerät vorhanden ist. Nach erfolgreicher Montage und Integration des WM8960 sollten an dieser Stelle sowohl Wiedergabe- als auch Aufnahmegeräte sichtbar sein.

6.3.1 Vorbereitung und Montage

Vor der Montage des WM8960 muss der RPi vollständig **stromlos** sein, das Netzteil ist abgezogen und der RPi ausgeschaltet. Nur so lässt sich sicherstellen, dass es beim Aufstecken des HAT auf die GPIO-Pins nicht zu Spannungsspitzen oder Kurzschlüssen kommt, die die Hardware beschädigen könnten.

Der WM8960 wird vorsichtig auf den 40-Pin Header des RPi gesteckt und mit den mitgelieferten Schrauben fixiert. Dadurch wird das Modul mechanisch stabilisiert und ein versehentliches Lösen beim Transport verhindert. Die Lautsprecher sollten bereits vor dem Einschalten an den Steckklemmen des HAT angeschlossen werden.

Nach der vollständigen mechanischen Montage kann das Netzteil wieder verbunden und der RPi gestartet werden.

Abb. 49: RPi mit aufgestecktem HAT und Lautsprecher

6.3.2 *Konfiguration über Device Tree Overlay*

Da der Linux-Kernel das WM8960-Modul bereits nativ unterstützt, ist keine separate Treiberinstallation erforderlich, wie sie auf der Waveshare-Webseite[32] beschrieben ist.

Die Aktivierung des WM8960 erfolgt über das Device Tree Overlay[33]. Dazu wird die Konfigurationsdatei **/boot/firmware/config.txt** angepasst. Folgende Zeilen müssen bearbeitet werden:

- dtparam=i2c_arm=on # einkommentieren
- dtparam=i2s=on # einkommentieren
- dtparam=audio=off # von 'on' auf 'off' ändern
- dtoverlay=wm8960-soundcard # diese Zeile neu hinzufügen

Eine komplette Übersicht der **config.txt** Datei ist im **Anhang B** zu finden.

Nach der Anpassung wird die Datei gespeichert (CTRL + X) und das System mit `sudo reboot` neu gestartet, damit das Overlay geladen wird.

6.3.3 *Überprüfung der Aktivierung*

Die erfolgreiche Initialisierung des Sound-HAT lässt sich über Kernelausgaben prüfen:

- Eine Überprüfung mit `aplay -l && arecord -l` zeigt das der WM8960 HAT nun als Aus- und Eingabegerät erkannt wird
- Mit `dmesg | grep -i wm8960` werden die Treiber-Meldungen angezeigt
- In der ALSA-Übersicht `cat /proc/asound/cards` erscheint die Karte als 1 [wm8960soundcard].

```
user01@pi-node-1:~ $ aplay -l && arecord -l
**** List of PLAYBACK Hardware Devices ****
card 0: vc4hdmi [vc4-hdmi], device 0: MAI PCM i2s-hifi-0 [MAI PCM i2s-hifi-0]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
card 1: wm8960soundcard [wm8960-soundcard], device 0: bcm2835-i2s-wm8960-hifi wm8960-hifi-0 [bcm2835-i2s-wm8960-hifi wm8960-hifi-0]
  Subdevices: 0/1
  Subdevice #0: subdevice #0
**** List of CAPTURE Hardware Devices ****
card 1: wm8960soundcard [wm8960-soundcard], device 0: bcm2835-i2s-wm8960-hifi wm8960-hifi-0 [bcm2835-i2s-wm8960-hifi wm8960-hifi-0]
  Subdevices: 0/1
  Subdevice #0: subdevice #0
user01@pi-node-1:~ $
```

Abb. 50: Ausgabe vom Befehl *aplay -l && arecord -l*

[32] https://www.waveshare.com/wiki/WM8960_Audio_HAT

[33] https://www.raspberrypi.com/documentation/computers/configuration.html#device-trees-overlays-and-parameters

6.3.4 Funktionstest Audio

Zunächst wird ein Wiedergabetest durchgeführt:

- `speaker-test -D hw:1,0 -c 2 -twav`

Die Testtöne werden abwechselnd auf dem linken und rechten Kanal ausgegeben.

Sollte kein Ton hörbar sein, müssen die Lautstärkeregler im **alsamixer** geprüft werden. Um diesen zu öffnen kann der Befehl `sudo alsamixer -c 1` genutzt werden. Nach der Eingabe öffnet sich ein Fenster, in dem man die verschiedenen Einstellungen für die Soundkarte vornehmen kann.

Folgende Regler bzw. Items müssen überprüft und ggf angepasst werden:

- Left Output Mixer PCM [Off] → wird mit der Taste **m** von „Mute“ auf „Ein“ umgeschaltet
- Right Output Mixer PCM [Off] → mit **m** auf „Ein“ schalten

Nach diesen Anpassungen ist der Testton auf beiden Lautsprechern hörbar.

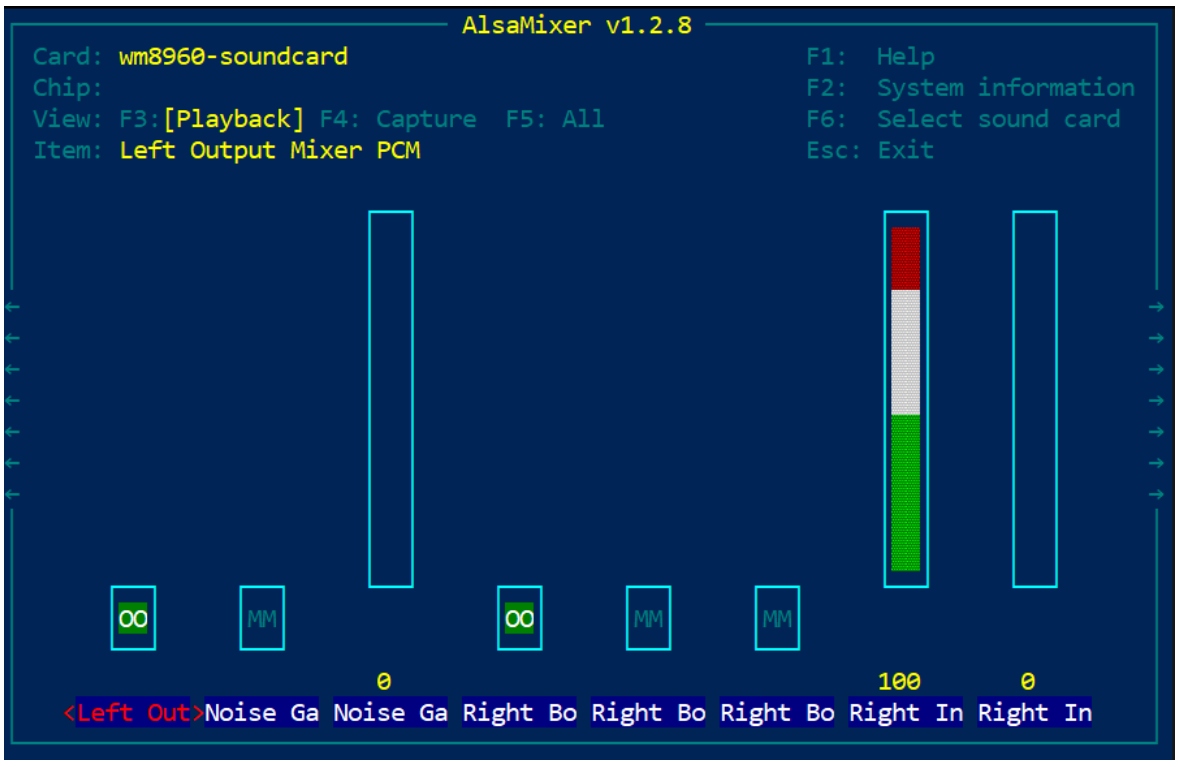

Abb. 51: alsamixer Einstellungen Output

Anschliessend wird der Mikrofon-Eingang getestet. Hierzu wird eine 5-sekündige Aufnahme der eigenen Stimme erzeugt, um diese dann wieder abspielen zu können.

Damit dies funktioniert, müssen zuvor im alsamixer weitere Einstellungen vorgenommen werden.

Folgende Items müssen umkonfiguriert werden:

- Playback [dB gain: -4.50, -4.50] → entspricht ca. Lautstärke 84
- Left Input Mixer Boost [Off] → mit **m** auf „Ein“ schalten
- Right Input Mixer Boost [Off] → mit **m** auf „Ein“ schalten

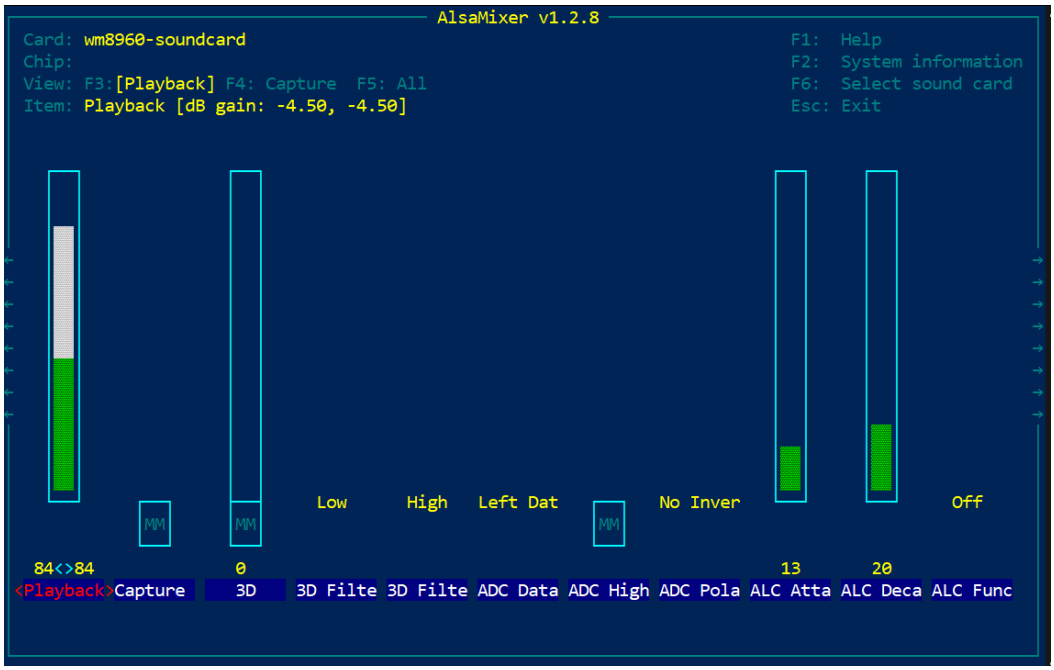

Abb. 52: alsamixer Einstellungen Input

Die Sprachaufnahme wird mit folgendem Befehl gestartet:

- `arecord -D hw:1,0 -f cd -c2 -d 5 test.wav`

Mit folgendem Befehl kann die gespeicherte Aufnahme wiedergegeben werden:

- `aplay test.wav` oder mit `aplay -D hw:1,0 test.wav`

Damit ist die Audiofunktionalität der Wiedergabe und Aufnahme bestätigt.

6.3.5 Integration ins Gesamtsystem

Der WM8960 wird nach der Konfiguration automatisch als Standardgerät erkannt.

Eine manuelle Anpassung über ein separates Konfigurationsfile ist nicht notwendig, da der Test mit `speaker-test -c 2` bereits ohne weitere Anpassungen die Tonausgabe über den Audio-HAT bestätigt.

Unter Raspberry Pi OS werden die Lautstärkeeinstellungen in der Regel dank dem **alsa-restore.service** automatisch gespeichert. Ob dieser aktiv ist, kann mit dem Befehl `systemctl status alsa-restore.service` überprüft werden.

Falls der Service nicht aktiv sein sollte, kann optional mit `sudo alsactl store` die aktuelle Konfiguration dauerhaft in **/var/lib/alsa/asound.state** gesichert werden.

Damit ist der WM8960 Audio-HAT vollständig in das System integriert. Er stellt sowohl Ein- als auch Ausgabegeräte für die Sprachkommunikation bereit und kann in den nachfolgenden Schritten für die PTT-Steuerung verwendet werden.

6.4 Integration Push-to-Talk Funktion

Um die Bedienung des Systems möglichst einfach und intuitiv zu gestalten, wird eine PTT-Funktion implementiert. Die Grundidee besteht darin, dass das Mikrofon standardmässig deaktiviert ist und erst durch das Drücken des physischen Tasters auf dem Sound HAT aktiviert wird. Sobald der Taster losgelassen wird, wird das Mikrofon unmittelbar wieder stummgeschaltet. Dieses Verhalten entspricht dem klassischen Funkbetrieb und verhindert die Übertragung unbeabsichtigter Umgebungsgeräusche.

6.4.1 Hardwareseitige Umsetzung

Der Waveshare WM8960 Audio HAT verfügt über einen integrierten Taster, der sich für die PTT-Funktion eignet. Alternativ könnte ein externer Taster über einen freien GPIO-Pin angeschlossen werden. Jedoch wird der integrierte Button verwendet, um den zusätzlichen Verkabelungsaufwand zu minimieren und eine kompakte Bauweise zu gewährleisten. ([14] Waveshare, 2025)

6.4.2 Softwareseitige Umsetzung

Die Steuerung der PTT-Funktion erfolgt über ein kleines Python-Skript. Dieses überwacht den GPIO-Pin des Tasters und schaltet bei einem Tastendruck das Mikrofon über ALSA frei. Beim Loslassen des Tasters wird das Mikrofon wieder deaktiviert. Auf diese Weise ist sichergestellt, dass nur während des Knopfdrucks eine Sprachübermittlung stattfinden kann.

6.4.3 Vorgehen für die Integration

Neues Verzeichnis erstellen mit dem Befehl:

- **`sudo mkdir -p /etc/walkietalkie`**

PTT-Konfigurationsdatei anlegen:

- **`sudo nano /etc/walkietalkie/ptt.env`**

Der Inhalt der Datei ist folgender:

```
GPIO_PIN=17                          # BCM-Nummer des Tasters
ALSA_CARD_INDEX=1                    # Kartennummer des WM8960
ALSA_CAPTURE_CONTROL=Capture         # Schalter für Mikrofon
DEBOUNCE_MS=30                       # Entprellzeit in Millisekunden
```

Python-Skript für PTT erstellen:

- `sudo nano /usr/local/bin/ptt.py`

Der komplette Code ist im **Anhang B** aufgeführt. Das Skript liest den GPIO-Pin, setzt beim Start das Mikrofon auf OFF und schaltet bei Tastendruck auf ON sowie beim Loslassen wieder auf OFF.

Nach dem Speichern muss das Skript ausführbar gemacht werden:

- `sudo chmod +x /usr/local/bin/ptt.py`

Systemd Service erstellen:

- `sudo nano /etc/systemd/system/ptt.service`

Der Inhalt[34] der Datei ist folgender:

```
[Unit]
Description=Push-to-Talk GPIO Service
After=multi-user.target

[Service]
ExecStartPre=/usr/bin/amixer -c 1 sset Capture nocap
ExecStart=/usr/local/bin/ptt.py
Restart=always
User=root

[Install]
WantedBy=multi-user.target
```

[34] https://www.youtube.com/watch?v=DUGZC-tNm2w

Service aktivieren und starten:

- `sudo systemctl daemon-reload`
- `sudo systemctl enable --now ptt.service`

Zur Kontrolle kann der Status des Service geprüft werden:

- `sudo systemctl status ptt.service`

```
user01@pi-node-1:~ $ sudo systemctl status ptt.service
● ptt.service - Push-to-Talk GPIO Service
     Loaded: loaded (/etc/systemd/system/ptt.service; enabled; preset: enabled)
     Active: active (running) since Wed 2025-09-24 15:09:09 CEST; 37min ago
    Process: 958 ExecStartPre=/usr/bin/amixer -c 1 sset Capture nocap (code=exited, status=0/SUCCESS)
   Main PID: 965 (python3)
      Tasks: 4 (limit: 368)
        CPU: 20.325s
     CGroup: /system.slice/ptt.service
             └─965 python3 /usr/local/bin/ptt.py

Sep 24 15:10:19 pi-node-1 ptt.py[1497]:   Capture channels: Front Left - Front Right
Sep 24 15:10:19 pi-node-1 ptt.py[1497]:   Limits: Capture 0 - 63
Sep 24 15:10:19 pi-node-1 ptt.py[1497]:   Front Left: Capture 63 [100%] [30.00dB] [on]
Sep 24 15:10:19 pi-node-1 ptt.py[1497]:   Front Right: Capture 63 [100%] [30.00dB] [on]
Sep 24 15:10:21 pi-node-1 ptt.py[1498]: Simple mixer control 'Capture',0
Sep 24 15:10:21 pi-node-1 ptt.py[1498]:   Capabilities: cvolume cswitch
Sep 24 15:10:21 pi-node-1 ptt.py[1498]:   Capture channels: Front Left - Front Right
Sep 24 15:10:21 pi-node-1 ptt.py[1498]:   Limits: Capture 0 - 63
Sep 24 15:10:21 pi-node-1 ptt.py[1498]:   Front Left: Capture 63 [100%] [30.00dB] [off]
Sep 24 15:10:21 pi-node-1 ptt.py[1498]:   Front Right: Capture 63 [100%] [30.00dB] [off]
user01@pi-node-1:~ $
```

Abb. 53: Statusabfrage des ptt.service

PTT-Funktion testen:

Zum Schluss kann mit den Befehlen `arecord` und `aplay` erneut eine Aufnahme gemacht werden. Hierbei ist zu beachten, dass der PTT-Button während der Aufnahme gedrückt und gehalten werden muss, da ansonsten keine Sprache aufgenommen wird.

Fehlersuche und Test

- `watch -n 0.3 "amixer -c 1 sget Capture | grep -E '\[(on|off)\]'"`

Mit diesem Befehl ist ersichtlich, ob das Mikrofon durch Drücken des PTT-Button von OFF auf ON schaltet. Das Verhalten kann auch optional auf dem GUI beobachtet werden.

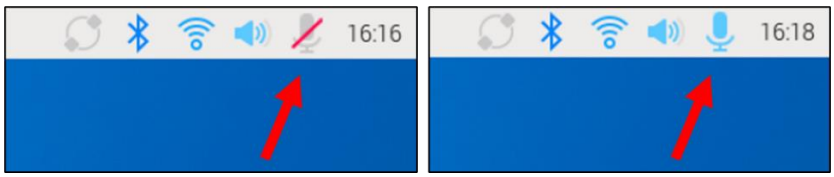

Abb. 54: Mikrofon von OFF auf ON

6.5 Installation und Konfiguration Mumble

Für die Sprachübertragung wird die Open Source Software Mumble eingesetzt. Jeder RPi betreibt sowohl den Mumble-Server als auch den Mumble-Client, sodass eine redundante Kommunikation gewährleistet ist.

6.5.1 Installation der Software

Zunächst erfolgt die Installation von Mumble-Client und Server:

- `sudo apt install -y mumble mumble-server`

Damit ist die Grundinstallation für Client und Server bereits abgeschlossen.

6.5.2 Konfiguration des Mumble-Servers

Nach der Installation von Mumble kann der Befehl `sudo dpkg-reconfigure mumble-server` verwendet werden, um eine erste Grundkonfiguration vorzunehmen. ([06] Hoerlis, 2023)

Folgende Einstellungen werden via dpkg festgelegt:

- Autostart mumble-server on server boot? Mit **<Yes>** bestätigen
- Allow mumble-server to use higher priority? Mit **<Yes>** bestätigen
- Passwort für den SuperUser Account anlegen: ****** und mit **<Ok>** bestätigen

Nach dieser Erstkonfiguration muss der Serverdienst neu gestartet werden. Dies kann mit dem Befehl `sudo systemctl restart mumble-server.service` durchgeführt werden.

Die Konfigurationsdatei des Servers befindet sich unter **/etc/mumble-server.ini**. Sie kann mit dem Befehl `sudo nano /etc/mumble-server.ini` geöffnet und bearbeitet werden. Für den Betrieb auf dem RPi werden folgende Anpassungen im Konfigurationsfile empfohlen:

- **welcometext:** Anzeige einer kurzen Information beim Verbindungsaufbau, wie z.B. Verbunden mit dem Mumble Server *walkietalkie-node-1*
- **users:** Begrenzung der maximalen Benutzerzahl auf zehn Teilnehmer

Nach den Anpassungen wird der *mumble-server.service* wieder neu starten. Damit ist die Grundkonfiguration des Servers abgeschlossen.

6.5.3 Konfiguration des Mumble-Clients

Der Mumble-Client wird offiziell nur als GUI-Variante angeboten, weshalb für die Bedienung der Zugriff über den VNC-Server genutzt wird. Nach der Installation befindet sich der Client im Menü unter **Internet** → **Mumble**.

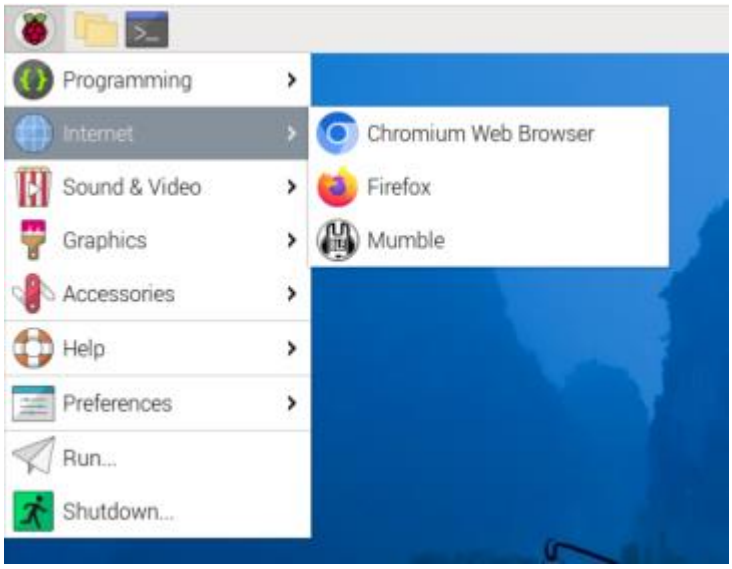

Abb. 55: Erster Start des Mumble-Clients

Beim ersten Start öffnet sich der **Audio Tuning Wizard**. Dieser Assistent führt durch die Erstkonfiguration der Audioeinstellungen und sollte unbedingt durchgeführt werden.

Dieser wird mit *Next* > gestartet.

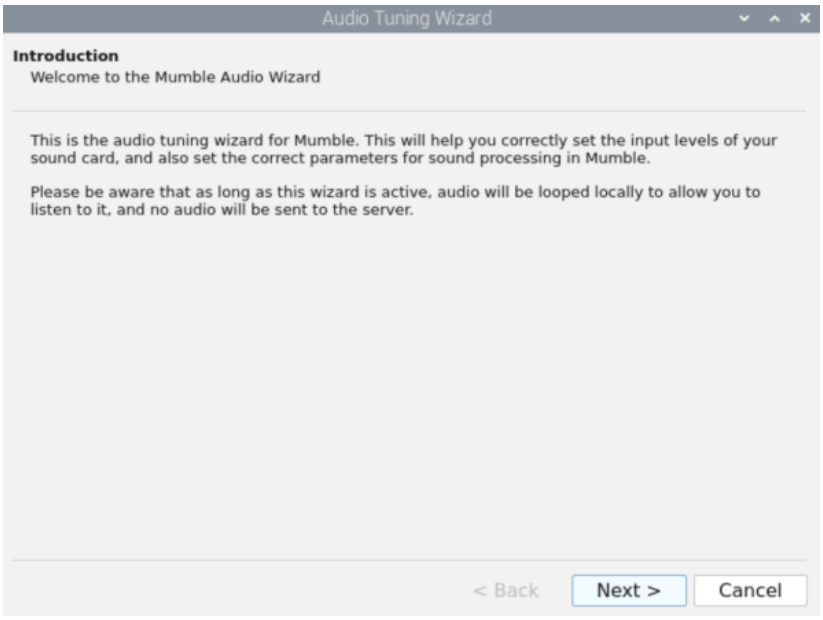

Abb. 56: Audio Tuning Wizard

Zunächst erfolgt die Auswahl des Audiogeräts. Es wird als System **PulseAudio** und jeweils **Default Input** sowie **Default Output** gewählt. Die Option *Enable positional audio* wird abgewählt, da diese Funktion nicht benötigt wird. Anschliessend mit *Next* > bestätigen.

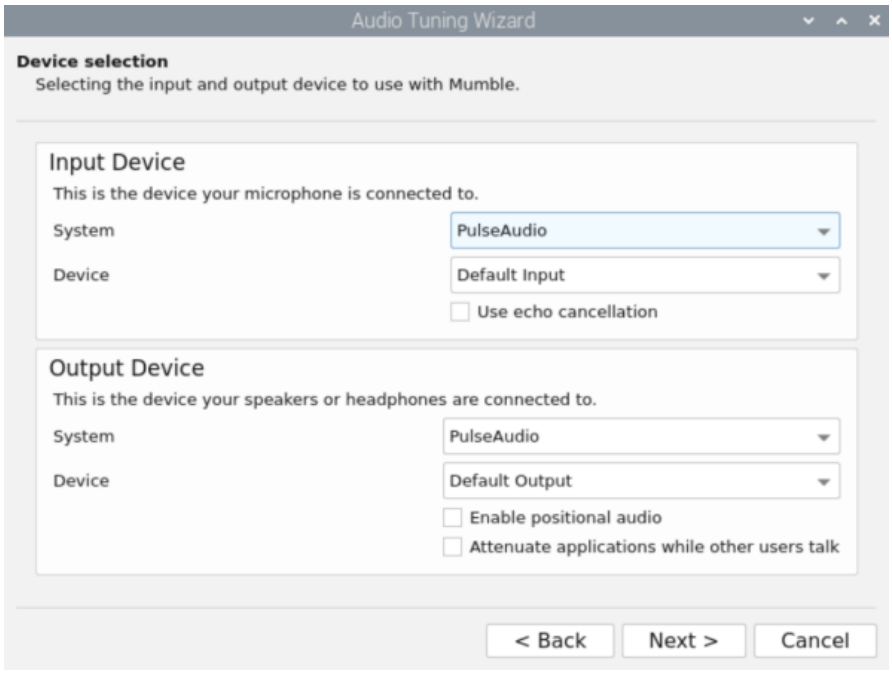

Abb. 57: Device Auswahl mit dem Audio Tuning Wizard

Im nächsten Schritt folgt das Device Tuning. Dabei wird die Puffergrösse bestimmt, also wie viele Audio-Daten zwischengespeichert werden, bevor sie abgespielt werden:

- Kleiner Puffer = geringere Latenz
- Zu kleiner Puffer = Knacken, Aussetzer oder Verzerrungen

Nun sollte über den Lautsprecher des WM8960 eine Teststimme hörbar sein. Falls nicht, müssen die Einstellungen im vorherigen Schritt angepasst werden. Ist die Stimme hörbar, können die Default-Einstellungen von 50 ms übernommen werden. Danach mit *Next* > bestätigen.

Anschliessend folgt das Volume Tuning. Hier wird die Empfindlichkeit des Mikrofons eingestellt. Es werden die Standardeinstellungen übernommen. Danach mit *Next* > bestätigen.

Im nächsten Schritt wird die Voice Activity Detection konfiguriert. Diese ist jedoch nicht relevant, da bereits eine systemweite PTT-Funktion integriert ist.

Falls gewünscht, kann hier die Mumble-interne PTT-Option gewählt und mit einer Taste aktiviert werden. Dies würde ein angepasstes oder neues Python-Skript erfordern, welches die GPIO-PTT des Sound-HATs auf eine Tastatureingabe abbildet.

Es wird die Einstellung **Raw amplitude from input** gewählt und die Empfindlichkeit auf Minimum gesetzt, da keine Sprachdetektion benötigt wird. Mit *Next* > bestätigen.

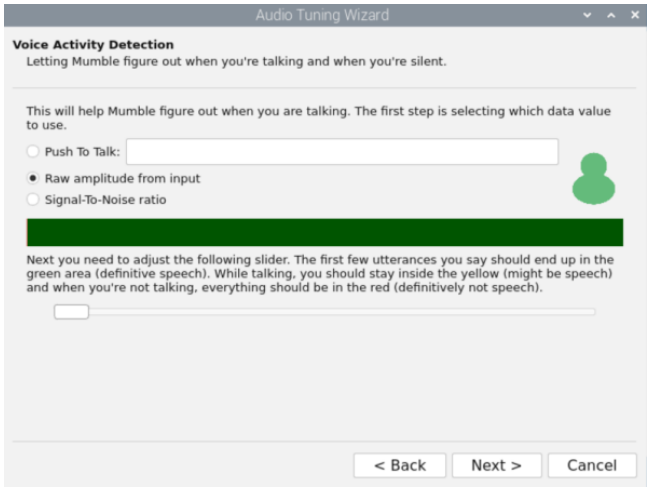

Abb. 58: Voice Activity Detection mit dem Audio Tuning Wizard

Es folgen die Qualitäts- und Benachrichtigungseinstellungen.

Quality settings: Die Einstellung **Balanced** bietet ein gutes Verhältnis aus Sprachqualität, Latenz und Bandbreite und wird daher verwendet. Alternativ könnte auch High gewählt werden, was jedoch nicht erforderlich ist.

Notification settings: Hier wird **Disable Text-To-Speech** gewählt, sodass nur kurze Hinweistöne wie z. B. User joined/left ausgegeben werden. Eine Vorlesefunktion wäre im Projektkontext störend.

Zum Abschluss erscheint die Meldung, dass die Konfiguration abgeschlossen ist. Optional kann hier das Senden anonymer Statistiken an das Mumble-Projekt aktiviert werden.

Mit *Finish* wird der Assistent beendet.

Direkt danach öffnet sich das Zertifikatsmanagement. Hier wird empfohlen, ein **automatisches Zertifikat** zu erstellen. Mit *Next* > die Standardauswahl bestätigen und das Abschlussfenster mit *Finish* schliessen.

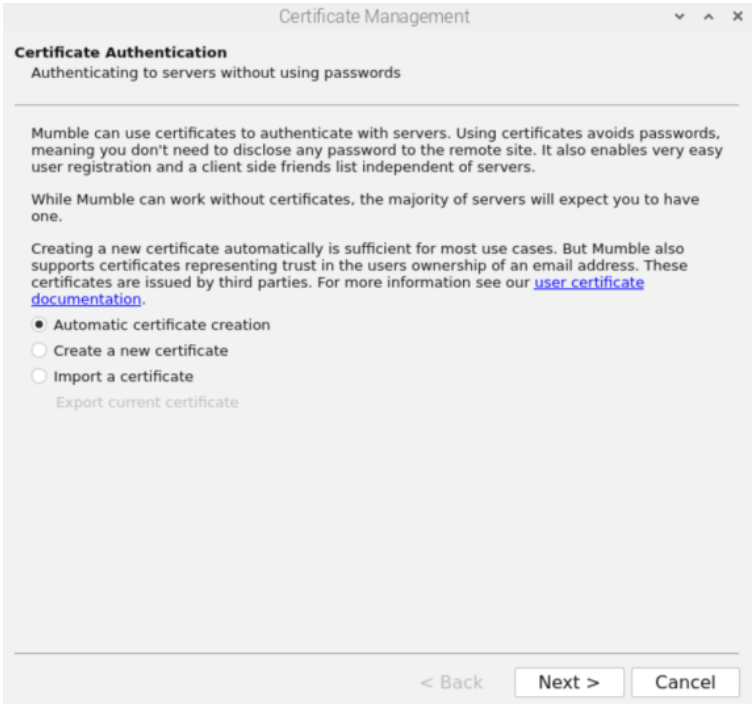

Abb. 59: Zertifikatsmanagement beim Mumble Client

Nun öffnet sich die Übersicht der verfügbaren Server. Der zuvor konfigurierte lokale Server sollte darin sichtbar sein. Mit Auswahl des Servers und dem Klick auf *Connect* erfolgt die Verbindung. Vorher ist noch der Benutzername anzugeben.

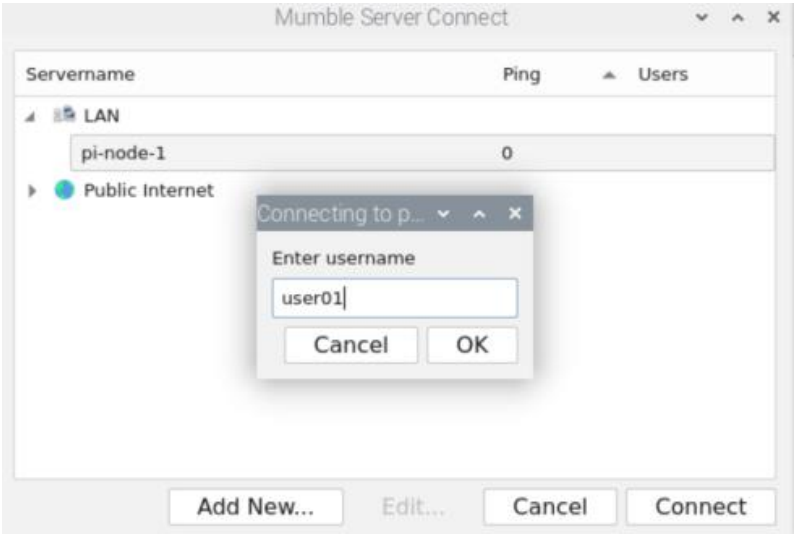

Abb. 60: Verbindung zum Server mit Mumble Client

Beim ersten Verbindungsaufbau erscheint eine Warnung zum unbekannten Serverzertifikat. Diese kann mit *Yes* bestätigt werden. Dadurch wird das Zertifikat lokal gespeichert und beim nächsten Verbindungsaufbau nicht erneut abgefragt.

Abb. 61: Warnmeldung Zertifikat beim Mumble Client

Nach erfolgreicher Verbindung ertönt ein akustisches Signal, welches den Verbindungsaufbau bestätigt. Damit ist die Grundkonfiguration des Clients abgeschlossen.

Optional kann man sich mit dem SuperUser-Account des Servers verbinden, um erweiterte Einstellungen vorzunehmen. Dies ist jedoch nicht notwendig.

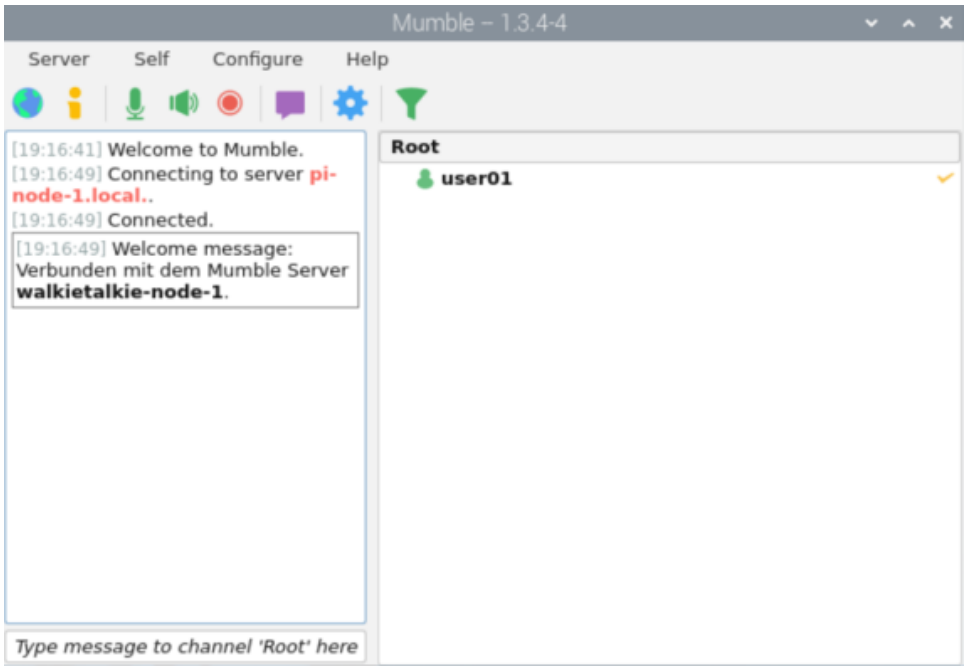

Abb. 62: Mumble Client verbunden mit Server

6.6 Integration Autostart und Fallback

Bevor mit der Autostart und Fallback Funktion gestartet werden kann, müssen alle Schritte aus den vorgängigen Kapiteln auf die restlichen RPi Knoten durchgeführt werden.

Nachdem dies abgeschlossen ist, müssen sich die Mumble-Clients auf allen drei Servern jeweils mindestens einmal verbinden. Dies ist wichtig, damit die Zertifikate der einzelnen Server lokal im Client gespeichert werden. So wird sichergestellt, dass beim automatischen Verbindungsaufbau keine weiteren Zertifikatswarnungen auftreten.

6.6.1 Umsetzung Autostart

Für die Umsetzung wird ein user-basierter systemd-Service eingerichtet, der beim Booten ein Startskript aufruft. Dieses Skript startet den Mumble-Client mit den nötigen Session-Variablen, damit das GUI-Fenster im VNC sichtbar ist.

Hinweis: Die nächsten Schritte werden auf dem *pi-node-1* für den *user01* umgesetzt.

6.6.1.1 Autostart Shell-Skript

Das Autostart-Skript wird im Home Verzeichnis des Users erstellt. Dies wird mit folgendem Befehl durchgeführt:

- `sudo nano /home/user01/mumble-autostart.sh`

Der Inhalt der Datei ist folgender:

```
#!/bin/bash
set -euo pipefail

# GUI/Session Variablen (wichtig fuer VNC/Qt)
export DISPLAY=:0
export XDG_RUNTIME_DIR=/run/user/$(id -u)
export DBUS_SESSION_BUS_ADDRESS=unix:path=${XDG_RUNTIME_DIR}/bus

# Kurze Basis-Wartezeit
sleep 5
```

```
# Optional: warten bis Netzwerk wirklich steht (Port erreichbar)
for i in {1..30}; do
  if nc -z 172.30.5.10 64738 2>/dev/null; then
    echo "Primary erreichbar."
    break
  fi
  echo "Warte auf Primary..."
  sleep 1
done

# Mumble GUI direkt auf pi-node-1 starten
exec /usr/bin/mumble "mumble://user01@pi-node-1.local:64738"
```

Nach dem Abspeichern der Datei muss diese noch ausführbar gemacht werden mit:

- **`sudo chmod +x /home/user01/mumble-autostart.sh`**

6.6.1.2 systemd-Service

Hierfür wird als erstens ein neues Verzeichnis erstellt:

- **`mkdir -p /home/user01/.config/systemd/user`**

Danach das systemd File mit:

- **`sudo nano /home/user01/.config/systemd/user/mumble-client.service`**

Der Inhalt der Datei ist folgender:

```
[Unit]
Description=Mumble Client Autostart (User)
After=graphical-session.target network-online.target
Wants=network-online.target
```

```
[Service]

Type=simple

ExecStart=/home/user01/mumble-autostart.sh

Restart=always

RestartSec=5


[Install]

WantedBy=default.target
```

6.6.1.3 Service aktivieren

Zum Schluss wird der neue Service aktiviert, hierzu werden folgende Befehle ausgeführt:

Systemd-Daemon neu laden

- `systemctl --user daemon-reload`

Service aktivieren und starten:

- `systemctl --user enable --now mumble-client.service`

User-Services[35] ohne Login aktivieren (**wichtig**)

- `sudo loginctl enable-linger user01`

[35] https://wiki.archlinux.org/title/Systemd/User

6.6.1.4 Autostart Test

Nachdem der Autostart auf allen RPi implementiert wurde, können alle mit `sudo reboot` neu gestartet werden. Nach dem Boot sollte sich jeder Mumble-Client automatisch mit dem Server *pi-node-1* verbinden. Dies ist auch im VNC-Fenster sichtbar.

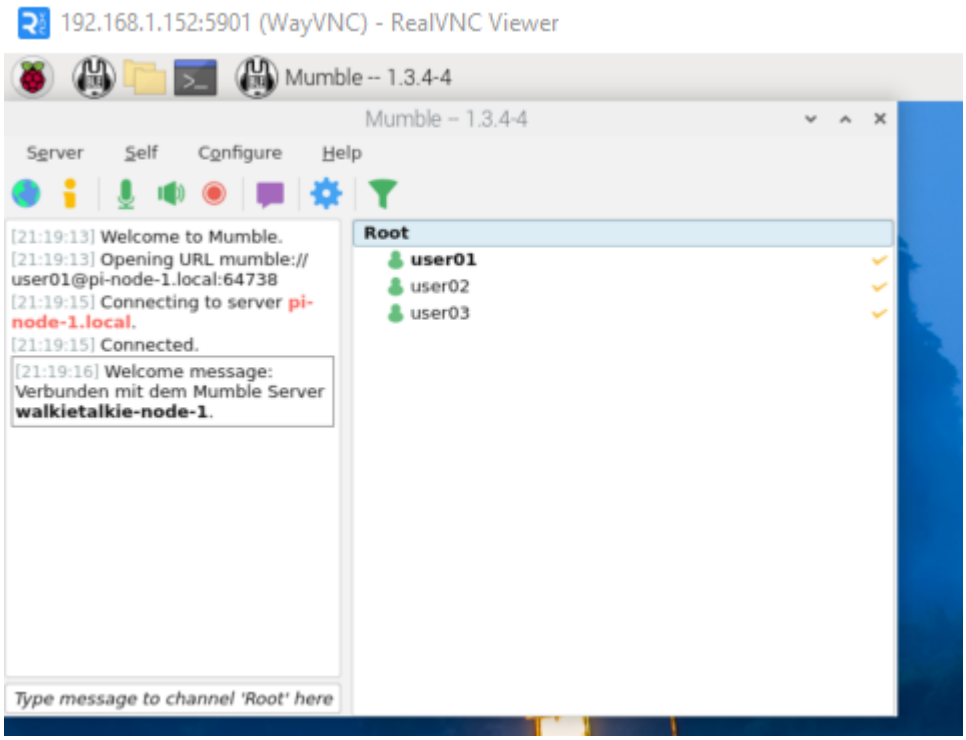

Abb. 63: Auto-Connect der Clients zum pi-node-1 Mumble Server

Die Autostart-Konfiguration ist damit abgeschlossen. Alle drei RPi verbinden sich nach dem Boot automatisch mit dem Server auf *pi-node-1*.

Hinweis:

Wenn keine Fallbacklösung gewünscht ist, kann das nächste Kapitel übersprungen werden. Die Lösung mit der festen Autoverbindung zu pi-node-1 ist zwar die zuverlässigere, erfüllt jedoch nicht vollständig die Anforderungen gemäss Pflichtenheft.

6.6.2 Umsetzung Fallback

Die folgende Umsetzung wird zunächst nur auf pi-node-1 realisiert und anschliessend analog auf pi-node-2 und pi-node-3 übertragen. Dabei ist in allen Terminalbefehlen sowie in den Konfigurationsdateien der Benutzername **user01** entsprechend durch **user02** bzw. **user03** zu ersetzen.

Zusätzlich ist in den jeweiligen **mumble_fallback.py** Dateien der Benutzername anzupassen, damit sich die Clients eindeutig voneinander unterscheiden lassen.

6.6.2.1 Anpassung mumble-autostart.sh

Der Autostart ruft nun nicht mehr direkt den Mumble-Client auf, sondern startet das Python-Fallback-Skript.

Deshalb muss der Inhalt des **mumble-autostart.sh** wie folgt angepasst werden:

```
#!/bin/bash
set -euo pipefail

# Session-Variablen setzen (wichtig für GUI über VNC)
export DISPLAY=:0
export XDG_RUNTIME_DIR=/run/user/$(id -u)
export DBUS_SESSION_BUS_ADDRESS=unix:path=${XDG_RUNTIME_DIR}/bus

# Kurze Pause, bis Netzwerk und Audio bereit sind
sleep 10

# Python-Fallback starten
exec /home/user01/mumble_fallback.py
```

6.6.2.2 Fallback-Skript

Das folgende Python-Skript startet den Mumble-Client, überwacht die Erreichbarkeit des aktuellen Servers und schaltet bei Ausfall automatisch auf den nächsten Server um.

Das Skript wird im Home Verzeichnis angelegt. Das File wird mit dem Befehl **`sudo nano /home/user01/mumble_fallback.py`** erstellt.

Der Inhalt der Datei sieht wie folgt aus:

```
#!/usr/bin/env python3
import os, socket, subprocess, time, shutil

USERNAME = "user01"
SERVERS = [
    ("pi-node-1.local", 64738),
    ("pi-node-2.local", 64738),
    ("pi-node-3.local", 64738),
]

def is_up(host, port):
    try:
        with socket.create_connection((host, port), timeout=1.0):
            return True
    except OSError:
        return False

if __name__ == "__main__":
    main()
```

Das vollständige Skript befindet sich im **Anhang B**.

Nach dem Abspeichern muss das Python-Skript mit folgendem Befehl ausführbar gemacht werden:

- **`sudo chmod +x /home/user01/mumble_fallback.py`**

6.6.2.3 Fallback Test

Bevor getestet werden kann werden die RPi einmal mit `sudo reboot` neu gestartet, damit der Autostart greift und sich die Clients automatisch wieder mit dem Server *pi-node-1* verbinden.

Danach kann auf allen drei RPi die VNC-Verbindung aufgebaut werden, um das Verhalten des Testes live zu beobachten.

Nun wird auf dem Mumble-Server *pi-node-1* ein Ausfall simuliert, indem der Server-Service mit folgendem Befehl gestoppt wird:

- `sudo systemctl stop mumble-server.service`

Erwartetes Verhalten: Die Mumble-Clients erkennen den Ausfall, beenden die laufende Sitzung und verbinden sich automatisch mit *pi-node-2*, welches der nächste Server ist.

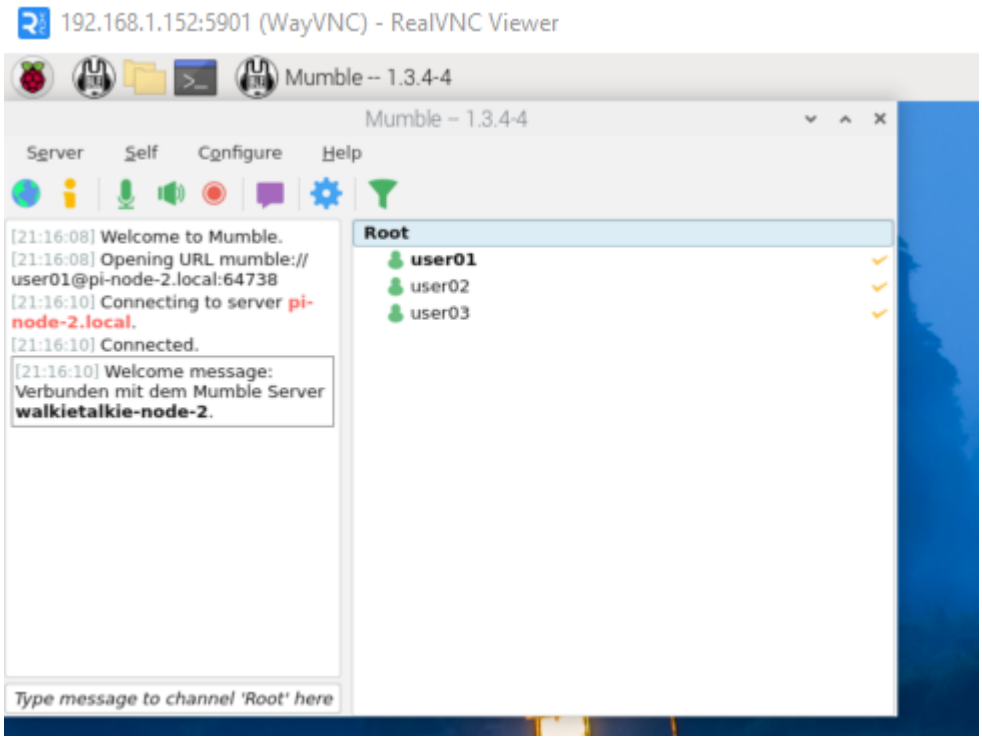

Abb. 64: Fallback von Node 1 zum Node 2

Danach den Server auf *pi-node-1* wieder starten:

- `sudo systemctl start mumble-server.service`

Nach kurzer Zeit wechseln die Clients automatisch zurück auf *pi-node-1*. Damit ist die Fallback-Funktion für alle Benutzer erfolgreich implementiert.

6.7 Installation und Integration Mesh-Netzwerk

Ein Hauptziel ist es, die Sprachkommunikation nicht nur über einen klassischen WLAN-Router, sondern auch über das Mesh-Netzwerk zu ermöglichen.

Bereits in der Konzeption wurde definiert, anstelle des integrierten WLAN-Chips des RPi, externe WLAN-Adapter zu verwenden. Die Adapter wurden im Vorfeld auf ihre Linux-Kompatibilität getestet (Plug & Play) und erfüllen die Anforderungen für den Einsatz im Ad-Hoc- bzw. Mesh-Modus.

Hinweis: Da ein zusätzlicher WLAN-Adapter verwendet wird, bleiben die Geräte über das Entwicklungsnetz weiterhin per SSH und VNC erreichbar.

6.7.1 Voraussetzungscheck für den WLAN-Chip

Zunächst wird der WLAN-Adapter über ein OTG-Kabel am RPi angeschlossen. Mit den folgenden Befehlen lässt sich prüfen, ob er korrekt erkannt wird:

- `lsusb` # zeigt den Adapter in der USB-Geräteliste als *Device 002*
- `iwconfig` # zeigt das zusätzliches Interface *wlan1*

Wenn der Adapter als **wlan1** sichtbar ist, erfolgt der nächste Test, ob der Treiber den Ad-Hoc oder Mesh-Modus unterstützt:

- `iw phy phy1 info | grep -A 10 "Supported interface modes"`

In der Ausgabe muss einer der folgenden Einträge erscheinen:

- *** IBSS** # unterstützt Ad-Hoc
- *** mesh point** # unterstützt 802.11s Mesh

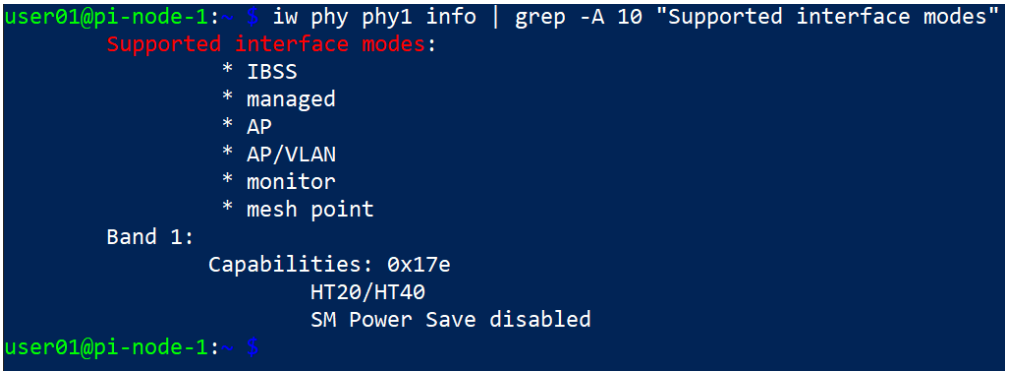

Abb. 65: Ausgabe *supported interface modes* des WLAN-Adapters phy1

6.7.2 Installation der Mesh-Komponenten

Für den Aufbau des Mesh-Netzwerks wird das Protokoll batman-adv eingesetzt. Dieses ist als Kernelmodul bereits in RPi OS enthalten und muss lediglich aktiviert werden. Für die Verwaltung und Diagnose wird zusätzlich das Tool **batctl** installiert:

- `sudo apt install -y batctl`

6.7.3 WLAN-Interface in NetworkManager deaktivieren

Damit das Mesh-Skript volle Kontrolle über den Adapter auf wlan1 hat, wird dieses Interface vom NetworkManager ausgeschlossen. Hierzu wird eine neue Datei angelegt:

- `sudo nano /etc/NetworkManager/conf.d/disable-wlan1.conf`

Inhalt der Datei:

```
[keyfile]
unmanaged-devices=interface-name:wlan1
```

Danach speichern und den NetworkManager neu starten:

- `sudo systemctl restart NetworkManager`

Mit dem Befehl `nmcli device status` ist nun ersichtlich, dass **wlan1** als *unmanaged* angezeigt wird.[36]

```
user01@pi-node-1:~ $ nmcli device status
DEVICE         TYPE      STATE                   CONNECTION
wlan0          wifi      connected               preconfigured
lo             loopback  connected (externally)  lo
p2p-dev-wlan0  wifi-p2p  disconnected            --
bat0           batadv    unmanaged               --
wlan1          wifi      unmanaged               --
user01@pi-node-1:~ $
```

Abb. 66: Ausgabe des Befehls *nmcli device status*

[36] https://github.com/binnes/WiFiMeshRaspberryPi/blob/master/part1/PIMESH.md

6.7.4 Systemd-Service mesh.service

Für die automatische Initialisierung wird ein Systemd-Service erstellt:

- **`sudo nano /etc/systemd/system/mesh.service`**

Inhalt der Datei:

```
[Unit]
Description=Mesh Netzwerk (batman-adv) konfigurieren
After=network.target
Wants=network.target

[Service]
Type=oneshot
ExecStart=/usr/local/bin/mesh-setup.sh
RemainAfterExit=yes

[Install]
WantedBy=multi-user.target
```

6.7.5 Autostart Skript für Mesh

Das Skript konfiguriert den Adapter im Ad-Hoc-Modus und bindet ihn in batman-adv ein. Als Erstens wird die Datei erstellt:

- **`sudo nano /usr/local/bin/mesh-setup.sh`**

Inahlt der Datei:

```
#!/bin/bash


# WLAN-Interface in Ad-hoc Modus versetzen
ip link set wlan1 down
iwconfig wlan1 mode ad-hoc
iwconfig wlan1 essid WalkieMesh
iwconfig wlan1 channel 1
ip link set wlan1 up


# batman-adv aktivieren
modprobe batman-adv
batctl if add wlan1
ip link set up dev bat0


# statische IP-Adresse setzen
# Hinweis: neuer Adressbereich!
ip addr add 10.30.5.10/24 dev bat0
```

Danach das Skript mit folgendem Befehl ausführbar machen:

- **`sudo chmod +x /usr/local/bin/mesh-setup.sh`**

Achtung: Hier wird ein neuer IP-Adressbereich vergeben. Da es sich um ein zusätzliches Interface handelt, muss jeder Knoten eine eindeutige Adresse besitzen, um IP-Konflikte zu vermeiden.

Zum Schluss wird der Service aktiviert:

- `sudo systemctl enable mesh.service`
- `sudo systemctl start mesh.service`

Mit `sudo systemctl status mesh.service` kann der Status des Service überprüft werden.

6.7.6 Anpassung von mumble_fallback.py

Um eine stabile Verbindung im Mesh zu gewährleisten, sollten auf allen drei RPi die Mesh-IP-Adressen auf bat0 im Fallback-Skript verwendet werden und keine *.local* Namen.

Die Serverliste wird zu Beginn der Datei neu festgelegt:

```
SERVERS = [           # Reihenfolge = Prioritaet (oben = bevorzugt)
    ("10.30.5.10", 64738),  # pi-node-1
    ("10.30.5.20", 64738),  # pi-node-2
    ("10.30.5.30", 64738),  # pi-node-3
]
```

Nach der Anpassung die Datei speichern und das Gerät mit `sudo reboot` neu starten.

Nach dem Neustart müssen auf allen drei Clients die Zertifikatsmeldungen der jeweils neuen Server-IP einmal bestätigt werden. Es wird empfohlen dies per VNC zu machen.

Vorgehen:

1. *pi-node-1*: Wenn alle drei Clients verbunden sind, die Zertifikatsmeldung für *pi-node-1* bestätigen. Anschliessend den Server mit `sudo systemctl stop mumble-server.service` stoppen
2. *pi-node-2*: Sobald die Clients dorthin verbinden, die Zertifikatsmeldung für pi-node-2 bestätigen. Dann auch hier den Server-Service stoppen
3. *pi-node-3*: Die verbleibenden Zertifikatsmeldungen bestätigen

Zum Abschluss alle gestoppten Server mit `sudo systemctl start mumble-server.service` wieder starten.

6.7.7 Lokaler Zeitserver installieren

Bevor die RPi Knoten in den definitiven Mesh-Betrieb überführt werden, wird ein zentraler Zeitserver eingerichtet. Dadurch wird sichergestellt, dass die Systemzeit auf allen drei Knoten identisch ist. Zeitabweichungen können insbesondere bei Zertifikaten, Logfiles oder der Fehlersuche zu Problemen führen.

Für die Synchronisation wird die Software **Chrony** verwendet. Die Installation erfolgt auf allen drei Knoten mit dem Befehl `sudo apt install chrony -y`.

6.7.7.1 Konfiguration auf pi-node-1 als Zeitserver

Auf *pi-node-1* wird die Konfiguration von Chrony angepasst[37]. Das Konfigurationsfile befindet sich unter **/etc/chrony/chrony.conf**. Dort werden zunächst alle drei externen Zeitserverquellen auskommentiert. Anschliessend werden am Ende der Datei die folgenden Zeilen ergänzt:

- local stratum 10 # Lokalen Stratum aktivieren
- allow 10.30.5.0/24 # Mesh-Subnetz erlauben

Nach dem Speichern wird Chrony mit `sudo systemctl enable --now chrony` aktiviert und mit `sudo systemctl restart chrony` neu gestartet.

Zur Überprüfung kann mit folgendem Befehl der aktuelle Status des Zeitservers abgefragt werden:

- `chronyc tracking`

Hinweis: Die angegeben Zeit in Chrony wird immer in UTC angezeigt.

```
user01@pi-node-1:~ $ chronyc tracking
Reference ID    : 7F7F0101 ()
Stratum         : 10
Ref time (UTC)  : Fri Oct 03 10:21:13 2025
System time     : 0.000000033 seconds fast of NTP time
Last offset     : +0.000000000 seconds
RMS offset      : 0.000000000 seconds
Frequency       : 6.647 ppm fast
Residual freq   : +0.000 ppm
Skew            : 0.000 ppm
Root delay      : 0.000000000 seconds
Root dispersion : 0.000000000 seconds
Update interval : 0.0 seconds
Leap status     : Normal
user01@pi-node-1:~ $
```

Abb. 67: Anzeige der Zeitquelle des lokalen Servers

[37] https://www.youtube.com/watch?v=jxHP0Vycdaw

6.7.7.2 Zeitkonfiguration auf pi-node-2 und pi-node-3

Auf den beiden weiteren Knoten werden in der Konfigurationsdatei **/etc/chrony/chrony.conf** ebenfalls alle drei externen Zeitquellen auskommentiert. Anschliessend wird die IP-Adresse von *pi-node-1* als einzige Zeitquelle eingetragen.

Der Eintrag `server 10.30.5.10 iburst` wird an folgender Stelle in der Datei ergänzt:

```
user03@pi-node-3:~ $ cat /etc/chrony/chrony.conf
# Welcome to the chrony configuration file. See chrony.conf(5) for more
# information about usable directives.

# Include configuration files found in /etc/chrony/conf.d.
confdir /etc/chrony/conf.d

# Use Debian vendor zone.
# pool 2.debian.pool.ntp.org iburst

# Use time sources from DHCP.
# sourcedir /run/chrony-dhcp

# Use NTP sources found in /etc/chrony/sources.d.
# sourcedir /etc/chrony/sources.d

#Zeitserver im Mesh verwenden
server 10.30.5.10 iburst
```

Abb. 68: Eintrag des Zeitservers im der chrony.conf

Nach der Anpassung wird Chrony mit `sudo systemctl restart chrony` neu gestartet. Die Synchronisation lässt sich mit folgendem Befehl überprüfen: `chronyc sources -v`.

```
user02@pi-node-2:~ $ chronyc sources -v

  .-- Source mode  '^' = server, '=' = peer, '#' = local clock.
 / .- Source state '*' = current best, '+' = combined, '-' = not combined,
| /             'x' = may be in error, '~' = too variable, '?' = unusable.
||                                                 .- xxxx [ yyyy ] +/- zzzz
||      Reachability register (octal) -.           |  xxxx = adjusted offset,
||      Log2(Polling interval) --.      |          |  yyyy = measured offset,
||                                \     |          |  zzzz = estimated error.
||                                 |    |           \
MS Name/IP address         Stratum Poll Reach LastRx Last sample
===============================================================================
^* 10.30.5.10                    10   7   377    12    -12us[-5483ns] +/-  365us
user02@pi-node-2:~ $
```

Abb. 69: Anzeige der Zeitquellen nach der Installation von Chrony

Damit verfügen alle Knoten über eine konsistente Zeitbasis, auch ohne Verbindung zu externen NTP-Servern. Es wird empfohlen die RPi neu zu starten.

6.7.8 Mesh Netzwerk Tests

Nach dem Neustart sollte `iwconfig` den **Ad-Hoc** Modus anzeigen.

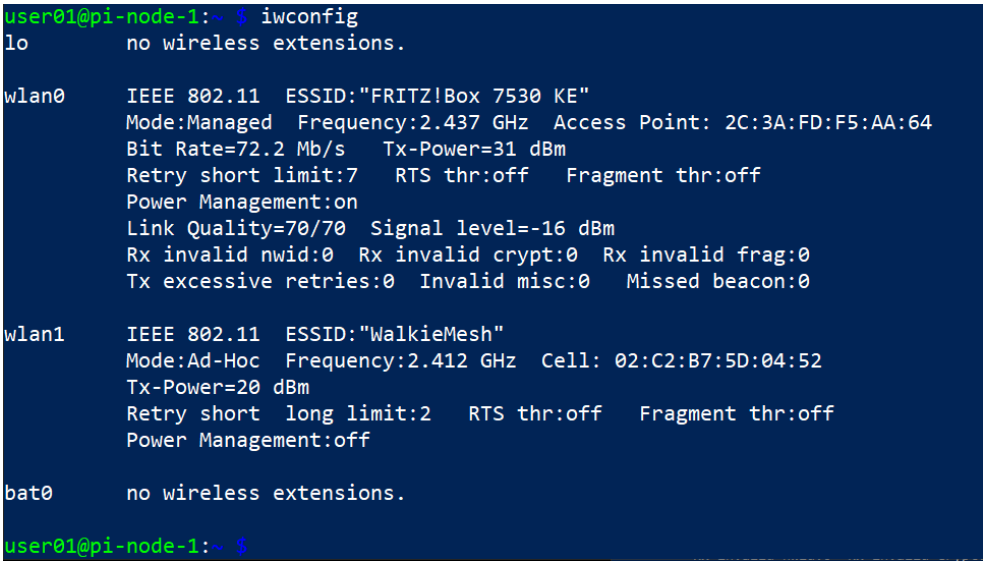

Abb. 70: iwconfig nach reboot

Anschliessend können mit `batctl` erste Diagnosen durchgeführt werden.

Hinweis: die `batctl` Befehle müssen immer mit `sudo` ausgeführt werden.

- **Nachbarn anzeigen:**

  `sudo batctl n`

  → listet alle erreichbaren Mesh-Nachbarn mit Linkqualität
- **Ping über das Mesh:**

  `sudo batctl ping 10.30.5.20` oder die MAC-Adresse

  → testet, ob der Knoten *pi-node-2* erreichbar ist
- **Traceroute über das Mesh:**

  `sudo batctl tr 10.30.5.30` oder die MAC-Adresse

  → zeigt den Pfad zum Ziel an, inkl. eventueller Zwischenknoten

Standardmässig zeigt batctl nur MAC-Adressen an, was unübersichtlich sein kann. Um diese durch Hostnamen zu ersetzen, wird auf jedem Knoten eine Datei `sudo nano /etc/bat-hosts` angelegt. Dort werden die MAC-Adressen der Interfaces der einzelnen Knoten mit den gewünschten Hostnamen verknüpft.

Inhalt der Datei:

```
1c:0c:44:00:37:f0   pi-node-1

08:da:35:df:f4:41   pi-node-2

6c:fd:b9:b5:cd:ff   pi-node-3
```

Wird nun `sudo batctl n` ausgeführt, erscheinen anstelle der reinen MAC-Adressen die vergebenen Hostnamen.

```
user01@pi-node-1:~ $ sudo batctl n
[B.A.T.M.A.N. adv 2024.2, MainIF/MAC: wlan1/1c:0c:44:00:37:f0 (bat0/46:d9:b7:df:4a:04 BATMAN_IV)]
IF             Neighbor              last-seen
        wlan1             pi-node-2    0.480s
        wlan1             pi-node-3    0.690s
user01@pi-node-1:~ $
user01@pi-node-1:~ $
user01@pi-node-1:~ $ sudo batctl ping pi-node-2
PING pi-node-2 (6c:fd:b9:b5:cd:ff) 20(48) bytes of data
20 bytes from pi-node-2 icmp_seq=1 ttl=50 time=0.73 ms
20 bytes from pi-node-2 icmp_seq=2 ttl=50 time=0.47 ms
20 bytes from pi-node-2 icmp_seq=3 ttl=50 time=0.90 ms
^C--- pi-node-2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss
rtt min/avg/max/mdev = 0.469/0.703/0.904/0.179 ms
user01@pi-node-1:~ $
user01@pi-node-1:~ $ sudo batctl tr pi-node-3
traceroute to pi-node-3 (08:da:35:df:f4:41), 50 hops max, 20 byte packets
 1: pi-node-3 (08:da:35:df:f4:41)  0.654 ms  0.662 ms  0.439 ms
user01@pi-node-1:~ $
```

Abb. 71: Diagnosebefehle mit *batctl* im Mesh

Nachdem die Tests erfolgreich durchgeführt wurden, kann beim zuvor eingesetzten Router das WLAN abgeschaltet werden. Die RPi sollten weiterhin über das Mesh verbunden bleiben und die Sprachübertragung funktionieren. Wenn das der Fall ist, kann das WLAN auf dem Router wieder aktiviert werden.

Abschluss

Mit der Einrichtung des Mesh-Netzwerks ist der entscheidende Schritt in Richtung einer dezentralen Kommunikationslösung umgesetzt. Die Knoten können nun eigenständig Verbindungen aufbauen und Datenpakete weiterleiten, ohne auf eine zentrale Infrastruktur angewiesen zu sein. Damit ist die Grundlage geschaffen, um die Mumble-Kommunikation unter realen Bedingungen zu testen.

7 Testkonzeption

Die folgende Testkonzeption beschreibt die Zielsetzung, Vorgehensweise und den Umfang der Tests für das entwickelte Walkie-Talkie-System im Mesh-Netzwerk. Es dient als Grundlage für die Durchführung der Tests und stellt sicher, dass die Anforderungen überprüft werden.

Das Konzept umfasst sowohl funktionale Szenarien wie Verbindungsaufbau, Sprachübertragung und Fallback als auch Belastungstests wie Dauerbetrieb, Reichweite und Energieverbrauch.

7.1 Testziele

Globale, messbare Testziele über alle Testfälle hinweg:

Nr.	Beschreibung	Messgrösse	Priorität*
01	Stabilität des Mesh-Link-Aufbaus	**batctl** Linkqualität, Dauer ≥ 5 Min	M
02	Automatischer Verbindungsaufbau aller Clients zum Hauptserver (pi-node-1)	Zeit bis Verbindung ≤ 2 Min	M
03	Sprachübertragung über 1 Hop verständlich	MOS ≥ 3.5	M
04	Robustheit bei Server- oder Knoten-Ausfall	Umschaltzeit ≤ 20 s, stabiler Reconnect	1
05	Reichweite im Feld nachgewiesen	Verbindung bis mind. 80 m stabil	3
06	Dauerbetrieb gewährleistet	Gespräch ≥ 30 Min ohne Abbruch	M
07	Energieversorgung praxistauglich	Laufzeit ≥ 4 h mit Powerbank	2

Tab. 19: Testziele

*** Priorität:** *M = Muss, 1 = hoch, 2 = mittel, 3 = tief*

7.2 Teststrategie und Teststufen

Da bereits in der Umsetzungsphase Vorabprüfungen auf Modul- und LAN-Ebene durchgeführt wurden, werden die Tests nun gezielt auf das Systemniveau ausgerichtet.

Strategie:

- Black-Box-orientierte Systemtests mit definierten Szenarien
- Fokus auf Robustheit, Praxistauglichkeit und Benutzerfreundlichkeit
- Keine Wiederholung redundanter LAN-Tests

Teststufen:

- *Integrationstests:* Zusammenspiel von Mesh, Mumble und PTT
- *Systemtests:* Gesamtsystem im Mesh, einschliesslich Fallback und Dauerbetrieb
- *Akzeptanztests:* Feldtests mit realitätsnaher Nutzung in Bezug auf Reichweite, Bewegung und Energie

Nr.	Teststrategien	Beschreibung
01	Vorgehen	Black-Box Tests im Mesh mit praxisnahen Szenarien
02	Teststufen	Integrationstest, Systemtest, Akzeptanztest
03	Vermeidung Redundanz	LAN-Tests nicht wiederholt, Fokus Mesh
04	Einschränkungen	Reichweite durch verfügbare Antennen limitiert, dokumentiert

Tab. 20: Teststrategie

7.3 Testabdeckung

7.3.1 Übersicht Testfälle

Folgende Testfälle wurden durchgeführt:

Nr.	Testobjekt	Bezeichnung	Datum	Mängel-klasse
T-01	Mesh	Mesh-Link Aufbau	04.10.2025	0
T-02	Mesh / Mumble	Startup Mumble-Clients auf Hauptserver	04.10.2025	2
T-03	Audiohardware	PTT-Funktion auf allen Knoten	04.10.2025	1
T-04	Sprachübertragung	Übertragung über 1 Hop	04.10.2025	0
T-05	Mesh / Hardware	Verhalten bei Antennenausfall	04.10.2025	1
T-06	Mumble	Fallback bei Server-Ausfall	04.10.2025	0
T-07	Mesh / Mumble	Fallback bei Stromausfall pi-node-1	04.10.2025	1
T-08	Mesh	Reichweite im Feld	05.10.2025	2
T-09	Mesh	Verbindung bei Bewegung	05.10.2025	2
T-10	Mesh / Hardware	Dauerbetrieb und Energielaufzeit	05.10.2025	1
T-11	Mesh / Hardware	Durchsatzmessung über 1 Hop	04.10.2025	2

Tab. 21: Übersicht Testfälle

7.3.2 Beurteilung Testziele und Abdeckung

Die gewählten Testfälle decken die im Pflichtenheft definierten Muss-Anforderungen vollständig ab. Insbesondere werden Stabilität, Sprachübertragung und Fallback-Mechanismen überprüft. Die Überprüfung der Verschlüsselung ist bewusst kein Bestandteil der Tests.

7.4 Testrahmen

7.4.1 Testvoraussetzungen

Nr.	Voraussetzungen	Beschreibung
01	Tester	Davide Bossi und Familie
02	Vorkenntnisse	Linux, Funkkommunikation, Netzwerk
03	Infrastruktur	3 x RPi Zero 2 WH inkl. WM8960-HATs, Power-banks, Laptop mit OBS-Studio SW

Tab. 22: Testvoraussetzungen

7.4.2 Mängelklassifizierung

Zur Bewertung der Testergebnisse wird eine Mängelklassifizierung eingesetzt, die den Schweregrad möglicher Abweichungen beschreibt. Mithilfe dieser Klassifizierung können Mängel nachvollziehbar dokumentiert und priorisiert werden. Dabei werden folgende Stufen unterschieden:

Klasse	Beschreibung	
0	**mängelfrei**	Ergebnis entspricht den Anforderungen
1	**belangloser Mangel**	System nutzbar, kleine Abweichung
2	**leichter Mangel**	System nutzbar, Brauchbarkeit leicht beeinträchtigt
3	**schwerer Mangel**	System eingeschränkt nutzbar
4	**kritischer Mangel**	Funktion nicht gegeben, System nicht abnahmefähig

Tab. 23: Mängelklassifizierung

7.4.3 Start- und Abbruchbedingungen

Für die Durchführung der Tests müssen definierte Start- und Abbruchbedingungen eingehalten werden. Diese stellen sicher, dass die Ergebnisse reproduzierbar und vergleichbar bleiben.

Startbedingungen:

- Alle RPi Knoten müssen betriebsbereit sein
- Das Mesh-Netzwerk muss vollständig konfiguriert und aktiv sein
- Die Audiohardware muss installiert und funktionstüchtig sein

Abbruchbedingungen:

- Auftreten eines Hardwaredefekts, der den Testbetrieb verhindert
- Akzeptanzkriterien können aufgrund von gravierenden Problemen nicht erfüllt werden

7.5 Testumgebung

7.5.1 Aufbau der Testumgebung

Um die Funktionsfähigkeit des Systems realitätsnah zu überprüfen, wird eine mobile Testumgebung eingerichtet, die unabhängig von externer Netzwerkinfrastruktur betrieben werden kann. Dieses Setup ermöglicht eine Validierung der Funktionalität im praktischen Einsatzszenario und stellt sicher, dass das System auch ohne Heimnetz zuverlässig betrieben werden kann.

Komponenten der Testumgebung:

Bildausgabe via HDMI-Port

Die RPi werden über einen Mini-HDMI zu HDMI-Adapter und ein HDMI-Kabel mit einem HDMI-Video Capture Adapter verbunden. Das Videosignal wird am Laptop mithilfe der Software OBS Studio angezeigt.

Eingabegeräte via Bluetooth

Maus und Tastatur werden per Bluetooth direkt mit dem RPi vom pi-node-1 gekoppelt. Dadurch ist eine vollständige Interaktion mit der grafischen Oberfläche ohne zusätzliche Netzwerkinfrastruktur möglich.

Fernzugriff via Mesh

Auf pi-node-1 ist ein SSH-Zugriff über das Mesh-Netzwerk eingerichtet. So sind die Systemsteuerung und Fehleranalyse im Feld möglich, auch wenn kein WLAN verfügbar ist.

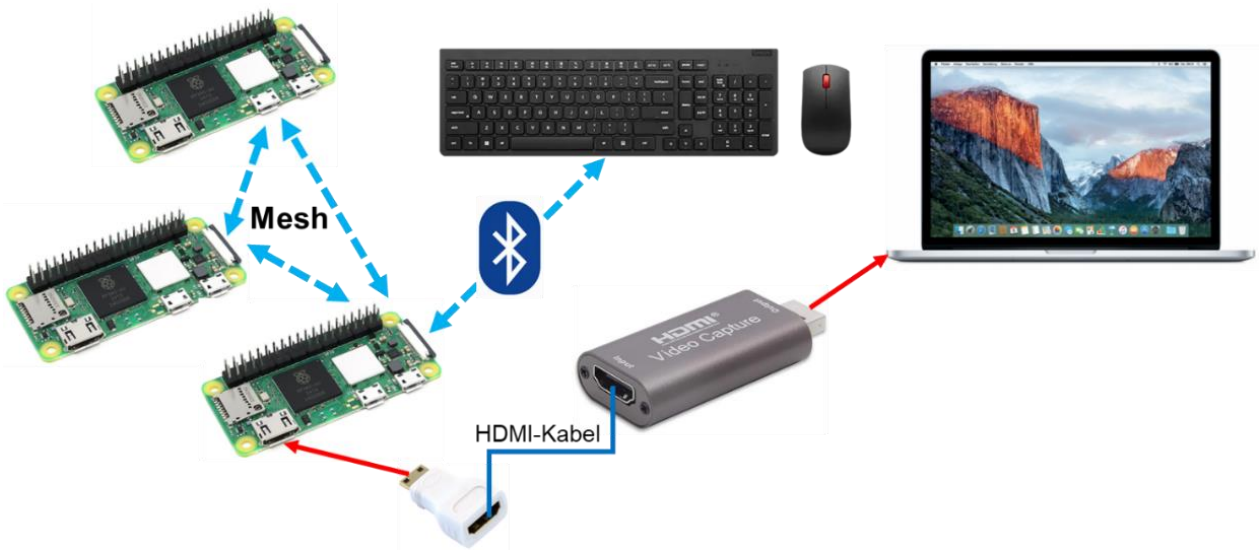

Abb. 72: Aufbau der Testumgebung

7.5.2 Testumgebung in der Wohnung

Die meisten Tests werden in der Wohnung durchgeführt. Die Idee dahinter ist, dass *pi-node-1* als zentraler Mesh-Knoten dient und Node 2 sowie Node 3 so positioniert werden, dass sie weit genug von Node 1 entfernt sind. In der Wohnung wird dies so simuliert, dass Node 2 und 3 nur über Node 1 miteinander kommunizieren können. Dazu werden Node 2 und 3 durch eine Gebäudeecke voneinander getrennt, sodass sie gezwungen sind, den Umweg über Node 1 zu nehmen.

Wichtig dabei ist zu erwähnen, dass Node 2 und Node 3 eine Sichtverbindung zu Node 1 haben, jedoch niemals direkte Sichtverbindung zueinander.

Zusätzlich sind während sämtlicher Tests der Entwickler-PC sowie Maus und Tastatur mit *pi-node-1* verbunden. Dies erleichtert sowohl die Konfiguration als auch die Überwachung des Mesh-Netzes.

Das Ganze stellt sich wie folgt dar:

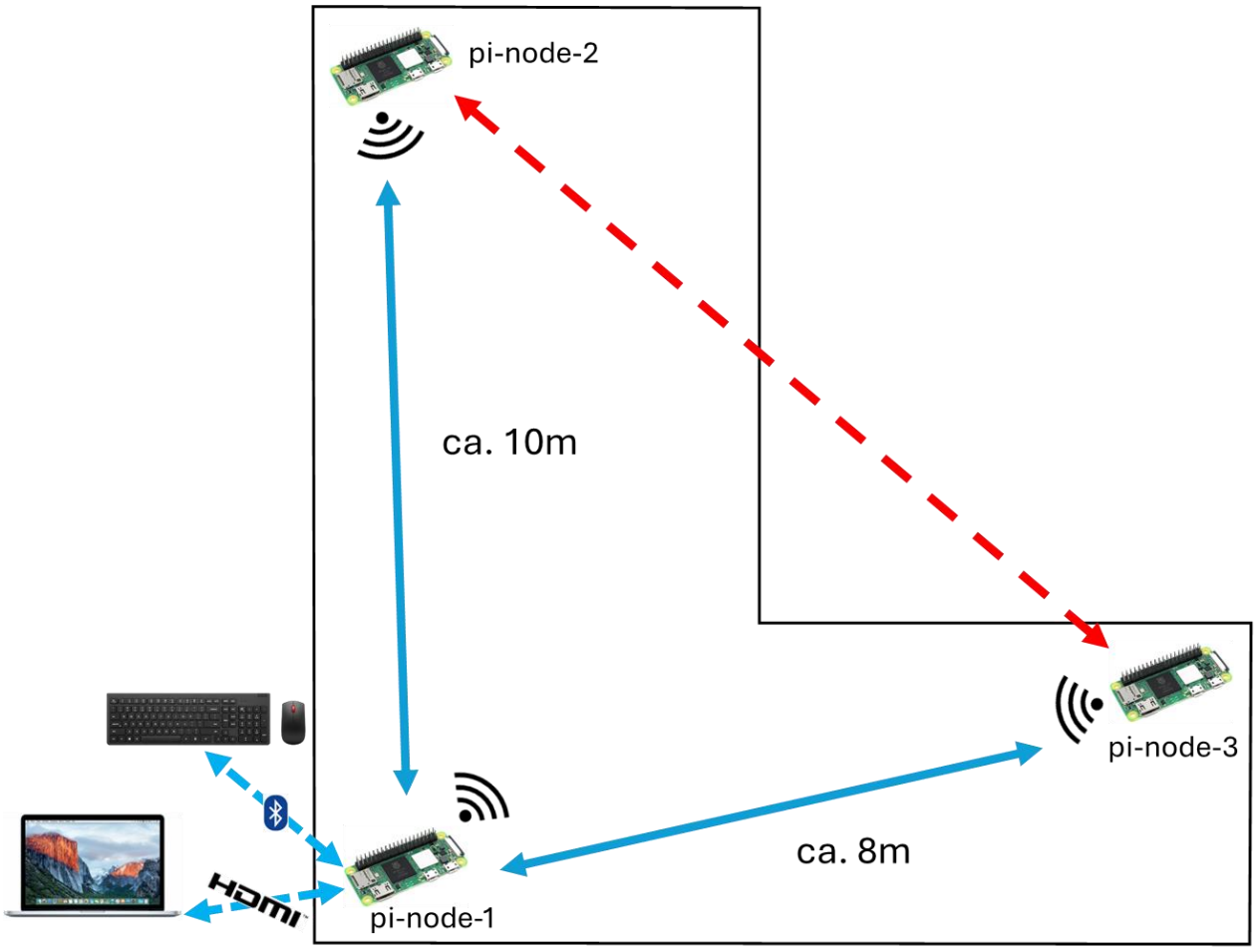

Abb. 73: Testumgebung in der Wohnung

7.5.3 Inbetriebnahme der Software OBS

Nachdem das Setup wie im vorherigen Kapitel beschrieben umgesetzt wurde, kann nun mit OBS-Studio das Videosignal des RPi erfasst werden. Nach der Installation wird die Software gestartet und folgende Einstellungen vorgenommen.

Die bestehende Szene kann wie im Beispiel zu „Pi“ umbenannt werden, indem man mit der rechten Maustaste auf die Szene klickt und den Menüpunkt **Umbenennen** auswählt. Anschliessend wird in dieser Szene eine neue Quelle hinzugefügt. Hierzu klickt man auf das **„+“** Symbol unter Quellen und wählt **Videoaufnahmegerät** aus.

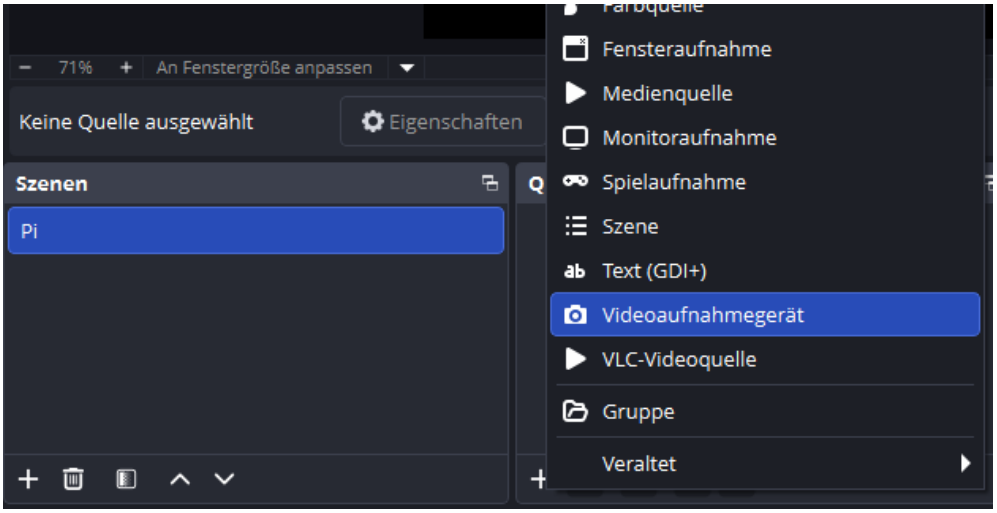

Abb. 74: Neue Quelle in OBS hinzufügen

Die neu erstellte Quelle kann auf „pi-node-1“ benannt werden. Nach der Eingabe des Namens wird mit **OK** bestätigt.

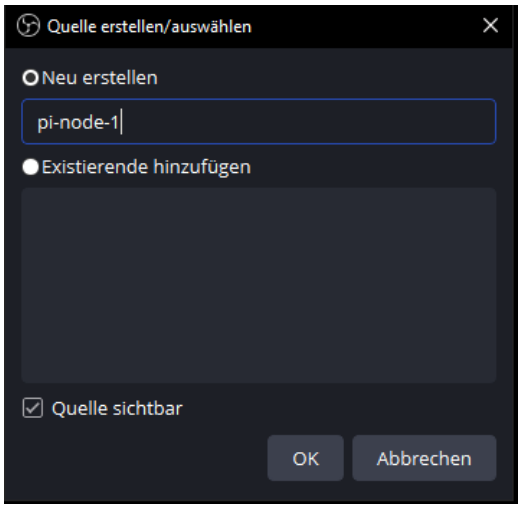

Abb. 75: Neue Quelle in OBS benennen

Anschliessend werden die Eigenschaften der Quelle „pi-node-1“ konfiguriert. In diesem Schritt wird das Eingabegerät ausgewählt. Da das Signal über die USB-Capture-Karte eingespeist wird, ist in der Dropdown-Liste das Gerät **USB Video** auszuwählen.

In der Vorschau sollte daraufhin das Videosignal des RPi sichtbar sein. Die Auswahl wird mit **OK** bestätigt.

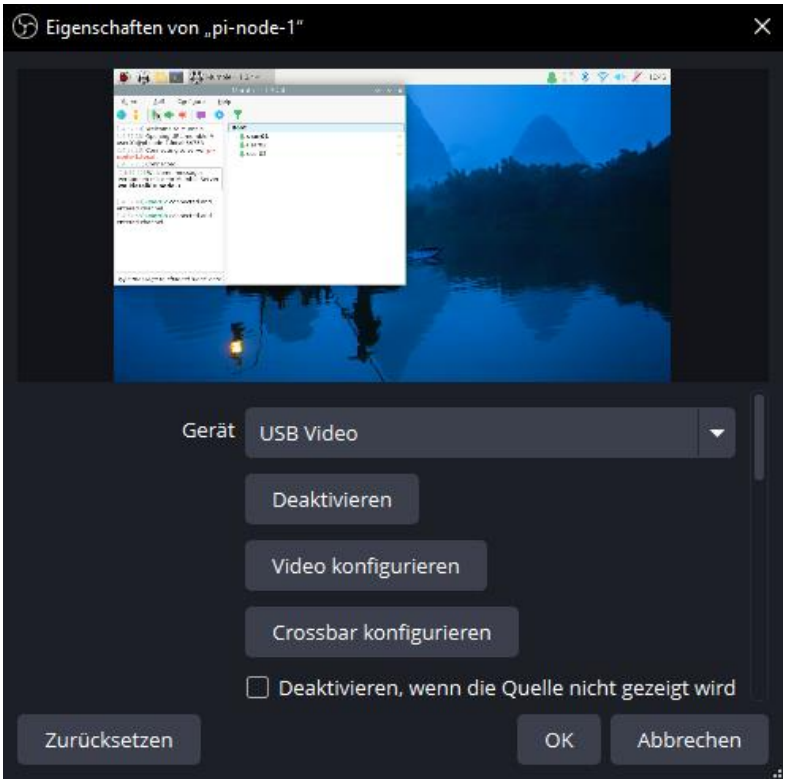

Abb. 76: Auswahl des Eingabegeräts in OBS

Es empfiehlt sich, die Eingabegeräte wie Bluetooth-Maus und -Tastatur bereits vor dem Einsatz im Feld über VNC mit dem RPi zu koppeln. Sobald das System in den reinen Capture-Betrieb übergeht, ist eine Kopplung über CLI-Befehle deutlich komplexer.

Damit sind die Vorbereitungen für die weiteren Testschritte abgeschlossen und die eigentliche Testdurchführung kann beginnen.

8 Durchführung der Testfälle

8.1 Testfälle

8.1.1 Testfall T-01

ID / Bezeichnung	T-01	Mesh-Link Aufbau
Beschreibung	Überprüfung, ob sich alle RPi Zero 2 WH korrekt im Mesh-Netzwerk verbinden und gegenseitig erkennen. Dabei wird getestet, ob die Kommunikation zwischen den Knoten stabil funktioniert und ob pi-node-2 über pi-node-1 mit pi-node-3 kommuniziert. Der Test wird vom *pi-node-1* ausgeführt.	
Testvoraussetzung	• Alle drei Raspberry Pi-Nodes sind gestartet und im Mesh-Modus aktiv • Entwickler-PC, Maus und Tastatur sind mit pi-node-1 verbunden • Das System ist vollständig gestartet, alle Dienste sind aktiv	
Testschritte	• Auf *pi-node-1* den Befehl `sudo batctl n` ausführen • Danach ssh auf pi-node-2 • Gesamtnetz mit `sudo batctl o` prüfen, dies zeigt alle Mesh-Knoten an • Anschliessend vom Node 2 `sudo batctl tr pi-node-3` ausführen • `watch -n 10 'sudo batctl n'` für mind. 10 Min. laufen lassen und die Verbindung beobachten	
Erwartetes Ergebnis	• Alle Nodes erscheinen in der Nachbarschaftstabelle • Alle drei Nodes erscheinen in der Originator-Liste (batctl o) inkl Hop anzeige • Pfad zwischen Node 2 und 3 über Node 1 wird angezeigt • Verbindung bleibt über mindestens 10 Minuten stabil und „last-seen“ ≤ 2 s bei allen Nodes	
Testdatum	04.10.2025	
Tester	Davide Bossi	
Mängelklasse	0	
Mangelbeschreibung	keine	
Bemerkungen	Der Test wurde gem. dem Kapitel *Testumgebung in der Wohnung* durchgeführt. Linkqualität stabil, Pfad blieb über 10 Minuten unverändert. Die Kommunikation untereinander war immer vorhanden.	

Tab. 24: Testfall T-01

8.1.2 Testfall T-02

ID / Bezeichnung	T-02	Startup der Mumble-Clients auf Hauptserver
Beschreibung	Überprüfung, ob alle Mumble-Clients nach dem Systemstart automatisch eine Verbindung mit dem Hauptserver **pi-node-1** herstellen, ohne dass eine manuelle Benutzerinteraktion erforderlich ist. Ziel ist sicherzustellen, dass die Sprachkommunikation unmittelbar nach dem Einschalten der Geräte betriebsbereit ist.	
Testvoraussetzung	• Alle Nodes korrekt konfiguriert • Autostart-Dienste sind aktiv • Der Test Mesh-Link Aufbau hat erfolgreich stattgefunden	
Testschritte	• Gleichzeitiger Neustart aller Nodes • Überprüfung des Mumble-Status via GUI • Beobachtung der Verbindungszeiten nach dem Systemstart	
Erwartetes Ergebnis	• Alle Mumble-Clients verbinden sich selbständig mit pi-node-1 • Die Verbindung erfolgt innerhalb von ≤ 2 Minuten nach dem Start • Kein manuelles Eingreifen nötig	
Testdatum	04.10.2025	
Tester	Davide Bossi	
Mängelklasse	2	
Mangelbeschreibung	Teilweise kam es vor, dass sich nach dem Reboot eines Nodes der Mumble-Client nicht automatisch mit dem Server auf **pi-node-1** verbunden hat. Dieses Verhalten trat ungefähr bei jedem zehnten Bootvorgang auf. Die Lösung bestand in einem erneuten Reboot des betroffenen Nodes bzw. teilweise auch in einem Neustart von **pi-node-1**.	
Bemerkungen	Der Test wurde gem. dem Kapitel *Testumgebung in der Wohnung* durchgeführt. Alle Clients waren nach 90 Sekunden verbunden.	

Tab. 25: Testfall T-02

8.1.3 Testfall T-03

ID / Bezeichnung	T-03	PTT-Funktion auf allen Nodes
Beschreibung	Überprüfung der PTT-Funktion auf allen RPi Zero 2 WH Nodes. Es wird getestet, ob das Mikrofon standardmässig deaktiviert ist und sich nur während gedrückter PTT-Taste aktiviert. Nach Loslassen der Taste muss das Mikrofon unmittelbar wieder stummgeschaltet werden.	
Testvoraussetzung	• Die Audio-Hardware ist korrekt installiert und initialisiert • Der Dienst ptt.service ist aktiv	
Testschritte	• Mikrofon-Grundzustand mit `amixer -c 1 sget Capture` prüfen • Während der Button gedrückt ist, erneut `amixer -c 1 sget Capture` ausführen • Nach Loslassen erneut prüfen • PTT-Taste gedrückt halten, kurze Testphrase sprechen und auf einem anderen Node den Sprachempfang kontrollieren • Den Test mindestens fünfmal pro Node wiederholen, um Stabilität und Reaktionszeit zu überprüfen	
Erwartetes Ergebnis	• Mikrofon ist standardmässig deaktiviert • Aktivierung erfolgt nur während gedrückter Taste • Deaktivierung unmittelbar nach Loslassen • Keine Verzögerungen oder Fehlfunktionen • Sprachübertragung klar und ohne Störgeräusche	
Testdatum	04.10.2025	
Tester	Davide Bossi	
Mängelklasse	1	
Mangelbeschreibung	Es kam teilweise vor, dass das Mikrofon nach einem Reboot oder nach einem Reconnect infolge eines Fallbacks **nicht stummgeschaltet** war, obwohl es in der GUI als stumm angezeigt wurde. Dieses Verhalten konnte jeweils durch einen kurzen Druck auf den PTT-Button behoben werden.	
Bemerkungen	Der Test wurde gem. dem Kapitel *Testumgebung in der Wohnung* durchgeführt. Die Reaktionszeit für das Aktivieren des Mikrofons betrug etwa 0,5 Sekunden. Diese Zeitspanne ist so gering, dass sie nicht als Mangel betrachtet wird.	

Tab. 26: Testfall T-03

8.1.4 Testfall T-04

ID / Bezeichnung	T-04	Sprachübertragung über 1 Hop
Beschreibung	Überprüfung der Sprachkommunikation zwischen *pi-node-2* und *pi-node-3* über den Hauptknoten **pi-node-1**. Ziel ist es sicherzustellen, dass die Sprachübertragung im Mesh-Netzwerk stabil, verständlich und mit minimalem Paketverlust erfolgt.	
Testvoraussetzung	• Das Mesh-Netzwerk ist aktiv und stabil • Der Mumble-Serverdienst auf pi-node-1 ist aktiv • Alle Mumble-Clients sind mit pi-node-1 verbunden • Die PTT-Funktion ist funktionsfähig	
Testschritte	• SSH-Verbindung auf pi-node-2 herstellen • Pfadprüfung mit `sudo batctl tr pi-node-3` ausführen. Der Pfad muss über pi-node-1 verlaufen • Auf pi-node-2 den PTT-Button gedrückt halten und etwa 30 Sekunden sprechen, danach auf pi-node-3 den Empfang prüfen • Sprachqualität auf pi-node-3 und pi-node-2 subjektiv beurteilen	
Erwartetes Ergebnis	• Pfad von pi-node-2 zu pi-node-3 verläuft über pi-node-1 • Sprachübertragung stabil, ohne hörbare Aussetzer oder Verzerrungen • Mean Opinion Score (MOS) ≥ 3.5	
Testdatum	04.10.2025	
Tester	Davide Bossi	
Mängelklasse	0	
Mangelbeschreibung	keine	
Bemerkungen	Der Test wurde gem. dem Kapitel *Testumgebung in der Wohnung* durchgeführt. Während der Übertragung wurde eine leichte Latenz festgestellt, die jedoch nicht als Mangel betrachtet wurde. Die Verständlichkeit blieb insgesamt gut.	

Tab. 27: Testfall T-04

8.1.5 Testfall T-05

ID / Bezeichnung	T-05	Verhalten bei Antennenausfall eines Nodes
Beschreibung	Überprüfung des Systemverhaltens bei einem simulierten Antennenausfall auf pi-node-2. Ziel ist es zu prüfen, ob die Verbindung zwischen den verbleibenden Knoten unterbrochen wird und ob sich der Knoten nach der Wiederherstellung der Antenne selbständig reorganisiert.	
Testvoraussetzung	• Das Mesh-Netzwerk ist aktiv und stabil • Alle Mumble-Clients sind mit pi-node-1 verbunden • Die Sprachübertragung über 1 Hop wurde zuvor erfolgreich getestet	
Testschritte	• Ausgangszustand auf pi-node-1 mit `sudo batctl o` prüfen, alle drei Nodes müssen sichtbar sein • Antennenausfall simulieren, indem auf pi-node-2 die Antenne physisch ausgesteckt wird • Reaktion auf pi-node-1 mit `sudo batctl o` beobachten • Antenne nach ca. 60 Sekunden wieder anschliessen • Erneut mit `sudo batctl o` auf pi-node-1 prüfen, ob sich der Knoten selbständig wieder verbindet	
Erwartetes Ergebnis	• Nach Ausstecken der Antenne verschwindet pi-node-2 aus der Originator-Liste • Die Verbindung zwischen pi-node-1 und pi-node-3 bleibt bestehen • Nach dem Wiederverbinden der Antenne verbindet sich pi-node-2 automatisch erneut mit dem Mesh-Netz.	
Testdatum	04.10.2025	
Tester	Davide Bossi	
Mängelklasse	1	
Mangelbeschreibung	Der Antennenausfall wurde von den anderen Nodes erst nach etwa 10 Sekunden erkannt. Nach dem Wiederverbinden der Antenne hat sich pi-node-2 nicht selbständig wieder mit dem Server auf pi-node-1 verbunden. Der Node musste neu gebootet werden, um wieder Teil des Mesh-Netzes zu sein.	
Bemerkungen	Der Test wurde gem. dem Kapitel *Testumgebung in der Wohnung* durchgeführt. Der festgestellte Mangel wird als marginal bewertet, da im realen Betrieb nicht davon auszugehen ist, dass eine Antenne physisch beschädigt oder entfernt wird.	

Tab. 28: Testfall T-05

8.1.6 Testfall T-06

ID / Bezeichnung	T-06	Fallback bei Server-Ausfall
Beschreibung	Überprüfung des Systemverhaltens beim Ausfall des Servers auf pi-node-1. Ziel ist zu verifizieren, ob alle Clients automatisch einen alternativen Server erkennen und sich selbständig neu verbinden. Der Test dient der Bewertung der Redundanz- und Fallback-Mechanismen im Mesh-Netzwerk.	
Testvoraussetzung	• Das Mesh-Netzwerk ist aktiv und stabil • Alle Mumble-Clients sind mit dem Hauptserver pi-node-1 verbunden • Die Fallback-Logik ist auf allen Nodes aktiv	
Testschritte	• Einen Server-Ausfall simulieren, indem der Mumble-Server-Dienst über das Terminal mit `sudo systemctl stop mumble-server.service` auf pi-node-1 gestoppt wird • Im GUI von pi-node-1 die Reaktion des Clients beobachten • Verifizieren, ob sich die Clients nach kurzer Zeit automatisch mit dem alternativen Server pi-node-2 verbinden • Den Mumble-Server mit `sudo systemctl start mumble-server.service` auf pi-node-1 wieder starten und beobachten, ob sich die Clients automatisch erneut mit pi-node-1 verbinden	
Erwartetes Ergebnis	• Nach dem Stoppen des Hauptservers erkennen die Clients den Ausfall, beenden und starten den Mumble-Client neu und versuchen sich automatisch mit pi-node-2 zu verbinden • Innerhalb von ≤ 15 Sekunden verbinden sich alle Clients mit dem Server auf pi-node-2 • Nach dem Neustart des Hauptservers verbindet sich das System selbständig wieder mit pi-node-1	
Testdatum	04.10.2025	
Tester	Davide Bossi	
Mängelklasse	0	
Mangelbeschreibung	keine	
Bemerkungen	Der Test wurde gem. dem Kapitel *Testumgebung in der Wohnung* durchgeführt. Die Raspberry Pi-Geräte hatten für diesen Test alle eine direkte Sichtverbindung im Umkreis von etwa fünf Metern.	

Tab. 29: Testfall T-06

8.1.7 Testfall T-07

ID / Bezeichnung	T-07	Fallback bei Stromausfall von pi-node-1
Beschreibung	Überprüfung des Systemverhaltens beim vollständigen Stromausfall des Hauptknotens pi-node-1. Ziel ist es festzustellen, ob sich das Mesh-Netzwerk stabil neu organisiert und ob die Mumble-Clients auf pi-node-2 und pi-node-3 automatisch auf einen alternativen Server umschalten.	
Testvoraussetzung	• Das Mesh-Netzwerk ist aktiv und stabil • Alle Mumble-Clients sind mit dem Hauptserver pi-node-1 verbunden • Der Test T-06 wurde erfolgreich abgeschlossen	
Testschritte	• Ausgangszustand prüfen: sudo batctl o auf pi-node-2 und pi-node-3 ausführen, alle drei Nodes müssen sichtbar sein • Den Strom von pi-node-1 trennen, indem die Powerbank ausgesteckt wird • Auf pi-node-2 und pi-node-3 die Reaktion beobachten und über `sudo batctl o` prüfen, wann pi-node-1 aus der Mesh-Liste verschwindet • Die Verbindung zwischen pi-node-2 und pi-node-3 nach dem Switchover mit Sprache überprüfen • Nach etwa 2 Minuten den Strom auf pi-node-1 wieder einschalten • Prüfen, ob sich pi-node-1 automatisch wieder in das Mesh integriert und die Clients erneut mit ihm verbinden	
Erwartetes Ergebnis	• Nach dem Stromausfall wird pi-node-1 innerhalb von ca. 10 Sekunden aus der Originator-Liste entfernt • Die Clients auf pi-node-2 und pi-node-3 erkennen den Ausfall und verbinden sich innerhalb von 20 Sekunden automatisch mit pi-node-2 als Ersatzserver • Nach dem Wiedereinschalten von pi-node-1 erfolgt automatisch die Rückverbindung der Clients zum Hauptserver • Das Mesh-Netzwerk reorganisiert sich selbständig, ohne manuelles Eingreifen	
Testdatum	04.10.2025	
Tester	Davide Bossi	
Mängelklasse	1	
Mangelbeschreibung	Nach dem Wiedereinschalten von pi-node-1 wurde der Knoten von den anderen Nodes erst nach etwa 30 Sekunden wieder erkannt. Die automatische Rückverbindung der Clients erfolgte zuverlässig, jedoch mit einer kurzen Verzögerung.	
Bemerkungen	Der Test wurde gem. dem Kapitel *Testumgebung in der Wohnung* durchgeführt. Die Raspberry Pi-Geräte hatten für diesen Test alle eine direkte Sichtverbindung im Umkreis von etwa fünf Metern.	

Tab. 30: Testfall T-07

8.1.8 Testfall T-08

ID / Bezeichnung	T-08	Reichweite im Feld
Beschreibung	Überprüfung der maximalen Funkreichweite des Mesh-Netzwerks im Aussenbereich. Ziel ist es, die Sprachverständlichkeit und Stabilität der Verbindung bei zunehmender Distanz zwischen den Nodes zu bewerten und die maximale Distanz für eine zuverlässige Kommunikation zu bestimmen. Der aktive Server muss sich während des gesamten Tests in der Mitte der drei Nodes befinden, damit eine gleichmässige Verbindung gewährleistet ist.	
Testvoraussetzung	• Alle drei Nodes sind einsatzbereit • Das Mesh-Netzwerk ist aktiv und stabil • Alle Mumble-Clients sind mit pi-node-1 verbunden • PTT-Funktion und Audioübertragung wurden zuvor erfolgreich getestet • Testumgebung: offenes Aussenfeld mit Sichtverbindung	
Testschritte	• Ausgangspunkt: Alle drei Nodes befinden sich in ca. 2 m Abstand zueinander, danach wird ein Sprachtest durchgeführt als Referenzqualität • Distanzschritte: pi-node-2 und pi-node-3 in Intervallen von 10 m weiter entfernen • Nach jedem Distanzschritt: - Sprachübertragung von pi-node-2 nach pi-node-3 über pi-node-1 testen - Sprachverständlichkeit nach MOS-Skala subjektiv bewerten • Beobachten, ab welcher Distanz erste Aussetzer oder Unterbrechungen auftreten • Ergebnisse und Distanzen dokumentieren	
Erwartetes Ergebnis	• Die Sprachübertragung bleibt bis zu einer Distanz von ca. **80 m** über einen Hop stabil und verständlich	
Testdatum	05.10.2025	
Tester	Davide Bossi	
Mängelklasse	2	
Mangelbeschreibung	Die maximale Reichweite betrug nach dem Test etwa 60 m. Die angestrebten 80 m wurden nicht erreicht. Die Wetterbedingungen waren nicht optimal, was die Signalstabilität möglicherweise beeinträchtigte.	
Bemerkungen	Der Test wurde im Freien durchgeführt, um reale Bedingungen zu simulieren. Die Distanzmessung erfolgte über das Geoportal **https://map.geo.admin.ch/**. Der pi-node-1 diente als Zwischenknoten und gleichzeitig als aktiver Server.	

Tab. 31: Testfall T-08

8.1.9 Testfall T-09

ID / Bezeichnung	T-09	Verbindung bei Bewegung
Beschreibung	Überprüfung der Stabilität des Mesh-Netzwerks und der Sprachkommunikation bei Bewegung der Nodes. Ziel ist zu ermitteln, ob während der Bewegung aller drei Nodes die Sprachqualität spürbar beeinträchtigt wird. Der aktive Server befindet sich in der Mitte der drei Nodes, um eine gleichmässige Funkverteilung sicherzustellen.	
Testvoraussetzung	• Das Mesh-Netzwerk ist aktiv und stabil • Alle Mumble-Clients sind mit dem Hauptserver pi-node-1 verbunden • Mobile Stromversorgung ist für alle Nodes vorhanden	
Testschritte	• Während der Bewegung durchgehend Sprachtest mit PTT-Funktion durchführen • Distanz und Bewegungsdauer dokumentieren	
Erwartetes Ergebnis	• Das Mesh-Netzwerk bleibt während der Bewegung stabil • Sprachübertragung ohne dauerhafte Unterbrechungen • Prüfen, wie lange eine automatische Wiederverbindung dauert, falls ein Abbruch erfolgt	
Testdatum	05.10.2025	
Tester	Davide Bossi	
Mängelklasse	2	
Mangelbeschreibung	Die Verbindung zwischen den Geräten blieb grundsätzlich stabil. Entscheidend war, dass pi-node-1 stets in Sichtweite blieb und als zentraler Mesh-Knoten fungierte. Die maximale Distanz zwischen pi-node-2 und pi-node-3 betrug während der Bewegung etwa 40 m, was als durchschnittlich zu bewerten ist. Wenn alle Nodes direkt im Mesh erreichbar waren (ohne Hop), lag der stabile Bewegungsbereich bei rund 25 m. Bei schneller Bewegung blieb die Verbindung bis ca. 20 m zuverlässig. Nach einem Verbindungsabbruch dauerte die automatische Wiederverbindung zum Server auf pi-node-1 teilweise bis zu **4 Minuten**.	
Bemerkungen	Der Test wurde im Freien durchgeführt und simulierte reale Einsatzbedingungen, bei denen sich die Geräte während des Betriebs bewegen. Pi-node-1 befand sich in zentraler Position, um die Verbindungsstabilität zwischen den mobilen Nodes zu gewährleisten.	

Tab. 32: Testfall T-09

8.1.10 Testfall T-10

ID / Bezeichnung	T-10	Dauerbetrieb und Energielaufzeit
Beschreibung	Überprüfung der Stabilität und Zuverlässigkeit des gesamten Systems im Dauerbetrieb sowie der maximalen Energielaufzeit der einzelnen Nodes bei Batteriebetrieb. Ziel ist sicherzustellen, dass die Geräte über längere Zeit stabil laufen und die Powerbanks eine ausreichende Betriebsdauer gewährleisten.	
Testvoraussetzung	• Alle drei Nodes sind vollständig konfiguriert und betriebsbereit • Das Mesh-Netzwerk ist aktiv und stabil • Alle Mumble-Clients sind mit dem Hauptserver pi-node-1 verbunden • Stromversorgung ausschliesslich über vollständig geladene Powerbanks	
Testschritte	• Alle Nodes gleichzeitig einschalten und im Normalbetrieb laufen lassen • Sprachübertragungen alle 2 Stunden 1 Minute Sprechzeit pro Node durchführen • Über die gesamte Testdauer sporadisch die Systemressourcen wie CPU- und RAM-Auslastung mit `htop` analysieren • Laufzeit jeder Powerbank dokumentieren	
Erwartetes Ergebnis	• Alle Nodes laufen im Dauerbetrieb stabil über mindestens 12 Stunden ohne Unterbrechung • CPU- und RAM-Auslastung bleiben im Normalbereich • Powerbanks ermöglichen eine durchschnittliche Laufzeit von mindestens 12 Stunden pro Node	
Testdatum	05.10.2025	
Tester	Davide Bossi	
Mängelklasse	1	
Mangelbeschreibung	Die Geräte blieben grösstenteils stabil verbunden. Vereinzelt kam es vor, dass pi-node-2 oder pi-node-3 die Verbindung zu pi-node-1 verloren. Die Ursache konnte nicht eindeutig ermittelt werden. Die Wiederverbindung zum Server dauerte, ähnlich wie im Test T-09, teilweise bis zu 4 Minuten. Die CPU- und RAM-Auslastung blieb durchgehend unter 50 %, was positiv bewertet wurde. Die Powerbanks überzeugten mit einer durchschnittlichen Laufzeit von bis zu 24 Stunden pro Node.	
Bemerkungen	Der Test fand unter realen Betriebsbedingungen statt und wurde im Rahmen der Testfälle T-08 und T-09 durchgeführt. Die Ergebnisse zeigen eine insgesamt gute Langzeitstabilität des Systems.	

Tab. 33: Testfall T-10

8.1.11 Testfall T-11

ID / Bezeichnung	T-11	Durchsatzmessung über 1 Hop
Beschreibung	Messung der effektiven Datenübertragungsrate im Mesh-Netzwerk zwischen pi-node-2 und pi-node-3 über den Zwischenknoten pi-node-1. Ziel ist es, den realen Durchsatz bei einer 1-Hop Verbindung zu ermitteln und die Leistungsfähigkeit des batman-adv-Protokolls zu bewerten.	
Testvoraussetzung	• Das Mesh-Netzwerk ist aktiv und stabil • Mumble-Server auf pi-node-1 ist aktiv • Alle drei Nodes sind eingeschaltet und verbunden • Der Test T-04 wurde erfolgreich durchgeführt • Keine zusätzliche Netzwerklast während der Messung	
Testschritte	• SSH-Verbindung zu pi-node-2 herstellen • Pfadprüfung mit `sudo batctl tr pi-node-3` → Verbindung läuft über pi-node-1 • Durchsatzmessung mit dem Befehl: `sudo batctl tp pi-node-3` • Messung fünfmal wiederholen, um einen stabilen Mittelwert zu erhalten • Ergebnisse dokumentieren und Mittelwert berechnen	
Erwartetes Ergebnis	• Der durchschnittliche Durchsatz liegt bei mind. *150 kbit/s*, abhängig von Signalstärke und Abstand • Der Wert bleibt während der Messungen konstant ±10 % • Keine Verbindungsabbrüche während der Messung	
Testdatum	04.10.2025	
Tester	Davide Bossi	
Mängelklasse*	2	
Mangelbeschreibung	Während des Tests kam es wiederholt zu Verbindungsabbrüchen mit dem Mumble-Server, obwohl die Mesh-Verbindung selbst stabil blieb. Die Messergebnisse schwankten stark zwischen 0,1 Mbit/s (mit Hop) und 10 Mbit/s (ohne Hop), abhängig von der WLAN-Signalqualität und der Sichtverbindung zwischen den Geräten. Der Durchschnitt über mehrere Messungen betrug ca. 0,2 Mbit/s, was für das verwendete Setup als praxisgerecht bewertet wird.	
Bemerkungen	Der Test wurde gem. dem Kapitel *Testumgebung in der Wohnung* durchgeführt. Die Messung erfolgte mit dem integrierten Tool `batctl tp`, welches ICMP-ähnliche Testpakete sendet und die effektive Datenrate über das Mesh ermittelt. Trotz der schwankenden Werte blieb die Verbindung über pi-node-1 stabil.	

Tab. 34: Testfall T-11

8.2 Gesamtbewertung der Tests

Die durchgeführten Tests zeigen, dass das geplante Kommunikationssystem mit den RPi grundsätzlich stabil und funktionsfähig arbeitet. Das Mesh-Netzwerk mit batman-adv bildete in allen Tests eine zuverlässige Kommunikationsbasis. Die automatische Vernetzung und die Erkennung der einzelnen Knoten funktionieren ohne manuelles Eingreifen.

Die Sprachübertragung über einen Hop ist verständlich, hat nur eine geringe Latenz und keine hörbaren Verzerrungen. Auch die PTT-Funktion arbeitet zuverlässig. Kleinere Abweichungen, wie ein gelegentlich nicht korrekt stummgeschaltetes Mikrofon nach einem Reboot, können im Betrieb einfach behoben werden.

Im Bereich Fallback zeigte das System eine gute Reaktionsfähigkeit. Beim Ausfall des Hauptservers oder bei einem Stromunterbruch auf pi-node-1 reagieren die Clients korrekt und stellten innerhalb kurzer Zeit eine Verbindung zu einem Ersatzserver her. Die automatische Rückverbindung nach Wiederverfügbarkeit des Hauptservers funktioniert ebenfalls zuverlässig.

Die Tests im Aussenbereich bestätigten, dass eine stabile Kommunikation über eine Distanz von rund 60 Metern möglich ist. Die ursprünglich angestrebten 80 Meter wurden unter den gegebenen Bedingungen nicht erreicht. Die Verbindung bleibt jedoch auch bei Bewegung und Richtungsänderungen grundsätzlich stabil, solange eine Sichtverbindung zu pi-node-1 besteht. Die Wiederverbindung nach einem kurzzeitigen Verbindungsverlust dauert teilweise mehrere Minuten, was die Einsatzfähigkeit in dynamischen Szenarien leicht einschränkt.

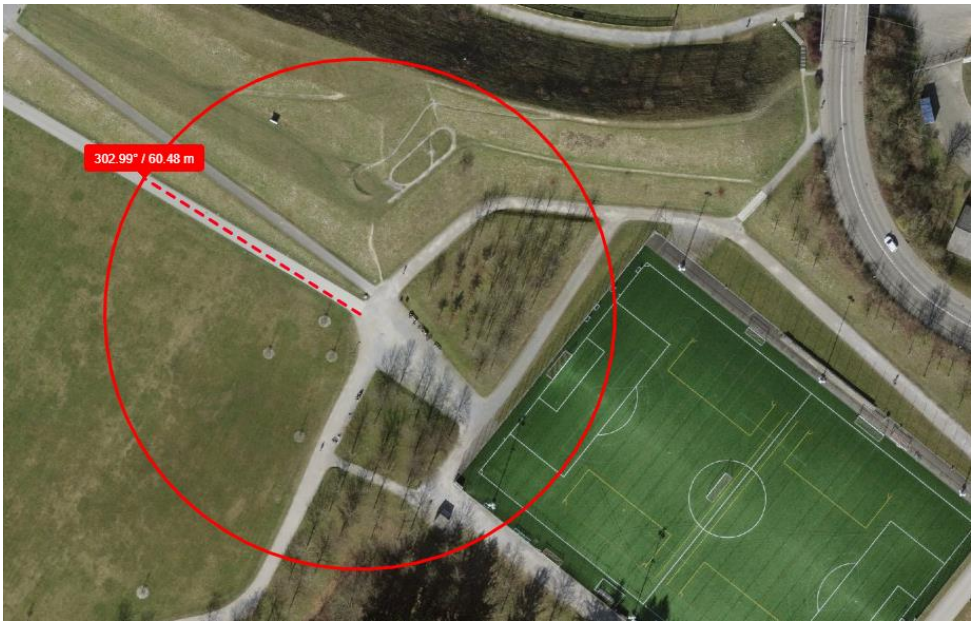

Abb. 77: Erreichte Funkstrecke mit einem Hop im Aussenbereich

Der Langzeittest im Dauerbetrieb zeigt eine gute Stabilität über mehrere Stunden hinweg. Die Systemressourcen werden nur gering beansprucht und die Powerbanks erreichen eine durchschnittliche Laufzeit von bis zu 24 Stunden. Damit ist das Ziel einer energieeffizienten Dauerkommunikation erfüllt.

Die Durchsatzmessungen ergeben eine hohe Schwankungsbreite. Der durchschnittliche Datendurchsatz über einen Hop liegt bei rund 0,2 Mbit/s, mit Spitzenwerten von bis zu 10 Mbit/s ohne Hop. Trotz dieser Unterschiede bleibt die Sprachkommunikation stabil, da VoIP-Dienste bereits ab 100 kbit/s funktionsfähig sind.

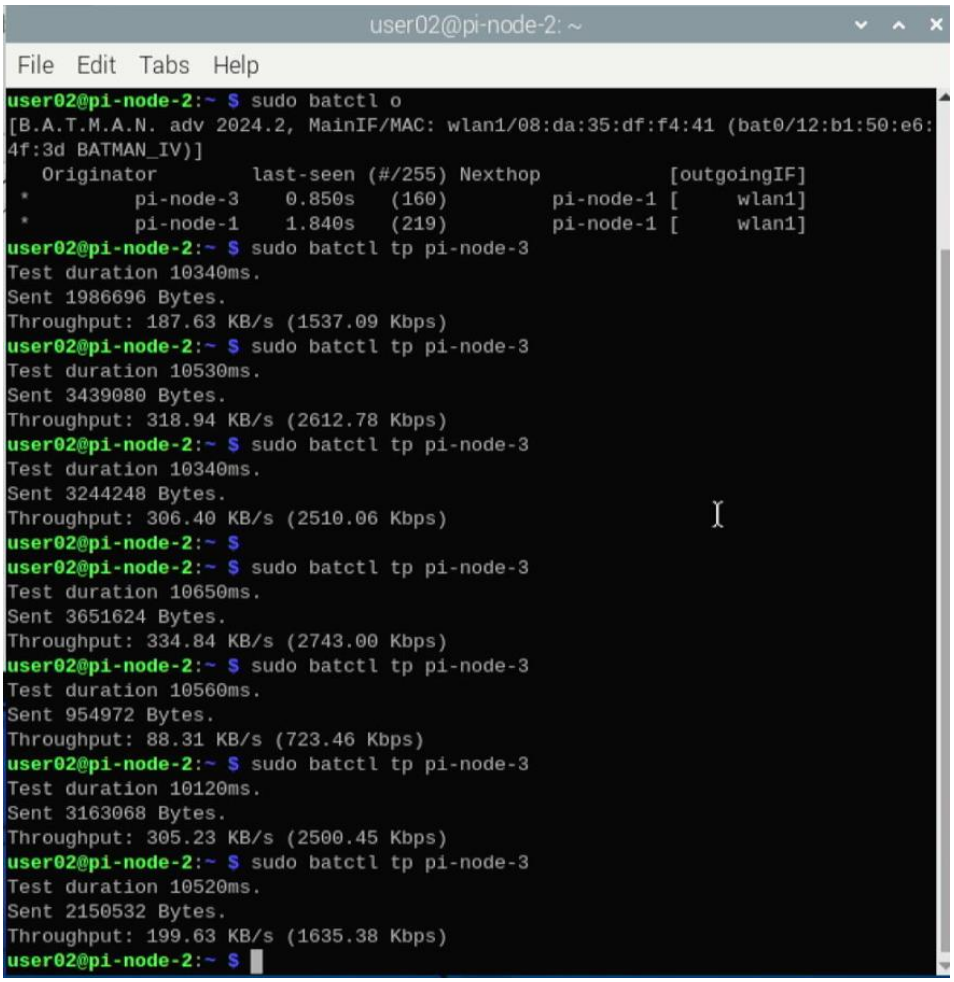

Abb. 78: Durchsatzmessung von Node 2 zu 3 über Node 1

Insgesamt kann festgehalten werden, dass das System den Anforderungen an ein mobiles, autarkes Mesh-Kommunikationsnetz weitgehend entspricht. Die grössten Herausforderungen bestehen in der Reduzierung der Wiederverbindungszeit und der Optimierung der Funkreichweite. Dennoch können die technische Funktionsfähigkeit und Praxistauglichkeit erfolgreich nachgewiesen werden.

9 Auswertung und Fazit

Dieses Kapitel fasst die wichtigsten Ergebnisse der Diplomarbeit zusammen, bewertet die erreichten Ziele im Vergleich zu den geplanten Anforderungen und zieht ein Fazit über den gesamten Projektverlauf. Dabei werden sowohl zeitliche Abweichungen als auch technische Anpassungen reflektiert.

9.1 Soll-Ist-Vergleich

9.1.1 Zeitlicher Vergleich

Der ursprüngliche Zeitplan sah eine fortlaufende Bearbeitung über mehrere Monate vor, gegliedert in die Phasen *Informieren, Planen, Entscheiden, Realisieren, Kontrollieren und Auswerten*.

Aufgrund der Geburt meiner Tochter kam es zu einer Verzögerung von rund **sechs Wochen** während der Realisierungsphase. Nach Wiederaufnahme der Arbeit konnte der Rückstand durch eine straffere Zeitplanung und den konzentrierten Einsatz an Wochenenden aufgeholt werden.

Die Hauptaufgaben wurden termingerecht abgeschlossen. Lediglich einige optionale Erweiterungen konnten aus zeitlichen Gründen nicht mehr umgesetzt werden.

Projekt-phase	Geplanter Zeitraum	Tatsächlicher Zeitraum	Abweichung	Kommentar
1. Informieren / Themenwahl	KW20 – KW25	KW20 – KW25	± 0 Wochen	Zeitplan eingehalten
2. Planen / Pflichtenheft / Zeitplan	KW28 – KW29	KW28 – KW29	± 0 Wochen	Zeitplan eingehalten
3. Entscheiden / Variantenvergleich	KW30 – KW31	KW34 – KW35	+ 4 Wochen	Verzögerung durch Geburt der Tochter, Arbeiten wurden nachgeholt
4. Realisieren / Systemaufbau	KW32 – KW36	KW36 – KW40	+ 4 Wochen	Familienbedingte Pause
5. Testen / Kontrollieren	KW37 – KW38	KW40	+ 2 Wochen	Tests auf das Wesentliche reduziert, Fokus auf Hauptfunktionen
6. Dokumentation / Auswertung	KW39 – KW41	KW41 – KW43	+ 2 Wochen	Pufferzeit durch vorherige Verzögerung aufgebraucht
Gesamtprojekt	Mai – Oktober 2025	Mai – Oktober 2025	+ ca. 2 Wochen (temporäre Verzögerung)	Leichte Verschiebung, Abschluss im geplanten Zeitraum

Tab. 35: Zeitlicher Vergleich

Trotz der Verzögerung konnte das Projekt innerhalb des ursprünglich geplanten Gesamtzeitrahmens abgeschlossen werden. Durch die priorisierte Bearbeitung der Hauptfunktionen wurde das Projektziel vollständig erreicht.

9.1.2 Zielerreichung nach Pflichtenheft

Alle im Pflichtenheft definierten Hauptziele wurden erreicht:

Pflichtenheft-Punkt	Soll-Zustand	Ist-Zustand	Bewertung
Aufbau eines mobilen Mesh-Netzwerks mit verschlüsselter Sprachübermittlung auf einem Pi	Drei funktionsfähige Knoten im Mesh	Erfolgreich umgesetzt	**erreicht**
Automatisches Routing und selbstheilende Struktur	Verwendung von batman-adv	Erfolgreich implementiert	**erreicht**
Sprachkommunikation über Mumble	Mumble-Server und Client pro Knoten	Voll funktionsfähig	**erreicht**
Cluster-Bildung mehrerer Server	Automatische Zusammenschaltung der Server	Geändert zu Fallback-Strategie	**angepasst**
TLS-Verschlüsselung	Sichere Sprachübertragung	Erfolgreich umgesetzt	**erreicht**
PTT über GPIO	Sprachsteuerung per PTT-Taste	Erfolgreich integriert	**erreicht**
Autostart aller Dienste	Vollautomatische Inbetriebnahme	Erfolgreich realisiert	**erreicht**
Mobiler Betrieb über Powerbank	Getestet im Feld	Erfolgreich getestet	**erreicht**
Optional: WLAN-Hotspot für Smartphones mit Mumble-App	Verbindung mit Mumble-App	Nicht realisiert	**nicht erreicht**

Tab. 36: Zielerreichung nach Pflichtenheft

In der ursprünglichen Idee sollten sich mehrere Mumble-Server automatisch zu einem Cluster zusammenschalten. Stattdessen wurde eine Client-Fallback-Logik umgesetzt.

Diese Lösung ermöglicht eine Redundanz und hat sich im Feldtest als stabil erwiesen. Die Abweichung wurde bewusst vorgenommen und als „angepasst“ markiert.

Alle Hauptziele wurden erreicht. Die Anforderungen an Autostart, Mobilität, Sprachqualität und Ausfallsicherheit wurden vollständig erfüllt.

9.2 Lesson Learned

9.2.1 Technisch

Rückblickend hat sich die technische Umsetzung des Projekts als anspruchsvoll, aber äusserst lehrreich erwiesen. Besonders positiv war die stabile Integration der zentralen Systemkomponenten. Die gewählte Struktur aus systemd-Units, Autostart- und Python-Skripten erwies sich als ideal für einen automatisierten Startvorgang aller Dienste, besonders für den Headless-Betrieb.

Die im Vorfeld erarbeiteten Testfälle haben sich als sehr nützlich erwiesen. Dank klarer Testkriterien konnten Fortschritte objektiv bewertet und auftretende Fehler eingrenzen werden. Dies erleichterte sowohl die Fehlersuche als auch die Dokumentation der Resultate.

Darüber hinaus bot das Projekt eine wertvolle Möglichkeit, sich in mehreren technischen Bereichen weiterzuentwickeln. So konnte ich mein Wissen im Umgang mit Linux stark vertiefen, mich intensiver mit Audio-Systemen auseinandersetzen und ein besseres Verständnis VoIP und Mesh gewinnen.

Trotz dieser positiven Erfahrungen gab es auch einige Hindernisse und Herausforderungen. Besonders viel Zeit floss in die Optimierung einzelner Komponenten, obwohl dafür im ursprünglichen Zeitplan kaum Reserven vorgesehen waren. Viele Stunden wurden in die Feinabstimmung investiert, um ein stabiles Gesamtsystem zu erreichen. Diese Detailarbeit erwies sich zwar als lernreich, führte aber zu einem spürbaren Mehraufwand.

9.2.2 Persönlich

Eine weitere Herausforderung ergab sich durch eine familiäre Unterbrechung während der Projektlaufzeit. Die Geburt meiner Tochter führte zu einer mehrwöchigen Pause, wodurch der zeitliche Ablauf zunächst ins Stocken geriet. Der anschliessende Wiedereinstieg war anspruchsvoll und erforderte eine Nachplanung und Umpriorisierung einzelner Aufgaben.

Die grösste Herausforderung bestand insgesamt darin, das Projekt mit den zahlreichen parallelen Verpflichtungen in Einklang zu bringen. Neben der Diplomarbeit galt es, einen Jobwechsel, den regulären Schulbetrieb und die militärischen Verpflichtungen im Oktober zu koordinieren. Diese Kombination aus beruflichen, privaten und schulischen Belastungen machte das Zeitmanagement zu einem zentralen Erfolgsfaktor. Der Anspruch, in allen Bereichen das Beste zu geben, war zwar kräftezehrend, führte aber auch zu einer deutlichen Steigerung der eigenen Organisations- und Priorisierungsfähigkeiten.

Trotz dieser intensiven Phase überwiegt das Positive deutlich. Die gesammelten Erfahrungen sind nicht nur für zukünftige technische Projekte, sondern auch für die berufliche Praxis von grossem Wert.

9.3 Ausblick

Für die zukünftige Weiterentwicklung gibt es mehrere Erweiterungsoptionen, mit denen sich die Funktionalität, Stabilität und Benutzerfreundlichkeit weiter verbessert lassen. Gleichzeitig lohnt es sich, einen Blick auf aktuelle Trends in der Mesh- und Netzwerktechnik zu werfen, um den Anschluss an technologische Entwicklungen zu halten.

9.3.1 Erweiterungsmöglichkeiten

Eine mögliche Erweiterung wäre die Hotspot-Integration: Ein RPi könnte als Access Point fungieren, sodass Smartphones direkt mit der Mumble-App ins Mesh eingebunden werden können. Dadurch entfiele die Konfiguration zusätzlicher Audiohardware und eines Mumble-Clients auf jedem Knoten. Voraussetzung hierfür ist jedoch, dass pro Knoten ein DHCP-Server eingerichtet wird, damit verbundene Geräte automatisch eine IP-Adresse aus dem Mesh-Adressbereich beziehen können.

Weiter ist eine Statusanzeige bei Knotenausfall denkbar: Eine visuelle Meldung könnte anzeigen, wenn ein Nachbarknoten ausgefallen ist. Die akustische Meldung durch die Mumble-App ist bereits vorhanden, aber eine unabhängige visuelle Komponente würde die Betriebssicherheit zusätzlich erhöhen.

Zur Reichweitenerweiterung könnten alternative Funkmodule integriert werden. Ein hybrides System aus WLAN-Mesh und ergänzender Funktechnik könnte die Netzabdeckung erhöhen.

Ein robusteres Gehäuse für die Hardware wäre sinnvoll, um die Feldtauglichkeit zu verbessern. Mit einem 3D-Drucker liesse sich ein passendes Gehäuse anfertigen, das Schutz vor Witterung, Erschütterungen und Staub bietet.

Zudem wäre der Einbau eines kleinen Displays sinnvoll, um visuelle Statusinformationen wie Verbindungsqualität zu anderen Knoten oder den Zustand der Verbindung zum Server anzuzeigen. Dies erhöht die Bedienbarkeit im Feld und erlaubt schnelle Diagnosen ohne Zugang zu einem externen Gerät.

9.3.2 Trends und zukünftige Entwicklungen

Beim Blick in die Zukunft zeigt sich, dass Mesh-Netzwerke weiterhin an Bedeutung gewinnen werden. Dies insbesondere in Szenarien, in denen Infrastruktur fehlt oder Ausfallsicherheit erforderlich ist.

Forschungen deuten auf folgende Entwicklungen hin:

Intelligente Netze und Edge Computing

Mesh-Knoten könnten zunehmend die eigene Rechenkapazität nutzen (Edge Computing), um lokal Daten zu verarbeiten und Netzwerkentscheidungen autonom zu treffen. Dies reduziert Latenz und entlastet zentrale Server. ([17] Yuan, 2024)

Neue Funktechnologien und Standards

Standards wie IEEE 802.11s ermöglichen einen nativen Mesh-Betrieb im WLAN und werden von Geräten zunehmend unterstützt. ([18] Wikipedia, 2025)

Energieautarke Knoten (Energy Harvesting)

Die Nutzung von Solar-, Vibrations- oder Thermoenergie zur Teilversorgung der Nodes gewinnt an Bedeutung, um dauerhaft autarken Betrieb zu ermöglichen. ([19] Anja, 2025)

In der Praxis existieren bereits ähnliche Projekte, die als Inspirationsquelle dienen können. So nutzt z. B. das offene Projekt Meshtastic LoRa-basierte Mesh-Netze für Off-Grid-Kommunikation, besonders in Gebieten ohne vorhandene Infrastruktur. ([20] Wikipedia, 2025)

Auch neuere dezentrale Kommunikations-Apps wie Bitchat zeigen, wie Peer-to-Peer-Kommunikation ohne klassische Netzwerke realisiert werden kann. ([21] Wikipedia, 2025)

9.4 Schlussfazit

Die Diplomarbeit zeigt, dass sich mit einfacher und kostengünstiger Hardware ein autarkes Kommunikationssystem realisieren lässt, das ohne zentrale Infrastruktur auskommt. Damit wurde das Hauptziel erreicht. Der entwickelte Prototyp erfüllt die wichtigsten Anforderungen aus dem Pflichtenheft und zeigt, dass eine dezentrale Sprachkommunikation auf IP-Basis sowohl technisch und praktisch umsetzbar ist.

Der entstandene Prototyp bildet eine solide Grundlage für zukünftige Erweiterungen und Weiterentwicklungen.

Insgesamt bestätigt die Arbeit, dass mit sorgfältiger Planung, technischem Verständnis und konsequenter Umsetzung komplexe Kommunikationssysteme auch auf kompakten Einplatinencomputern realisieren lassen. Das Projekt verbindet theoretisches Wissen mit praktischer Anwendung und zeigt, wie Open Source Technologien eine geeignete Basis für innovative Kommunikationslösungen bilden können.

Literaturverzeichnis – Sekundärquellen

[01] BABS, B. f. (2005). Abgerufen am 23. Mai 2025 von https://www.zsooberthurgau.ch: https://www.zsooberthurgau.ch/phocadownloadpap/Einsatzunterlagenzumherunterladen/funkgestuetzte%20telematiksysteme.pdf

[02] Mekiker, B. (23. August 2023). Abgerufen am 24. Mai 2025 von Cornell University: https://arxiv.org/pdf/2308.12328

[03] Ebbutt, G. (18. Februar 2021). Abgerufen am 26. Mai 2025 von DTC | Codan Company: https://www.dtccodan.com/assets/general-downloads/Mesh-network-Increasing-resilience-and-reducing-infrastructure-across-network-domains.pdf

[04] AVM, F. (2025). *Fritz!Box 7530 | Assets AVM.* Abgerufen am 25. Juni 2025 von https://assets.avm.de/files/docs/fritzbox/fritzbox-7530/fritzbox-7530_man_de_DE.pdf

[05] Raspberry Pi, L. (2025). *Raspberry Pi Documentation*. Von https://www.raspberrypi.com/documentation/ abgerufen

[06] Hoerlis, T. (29. 12 2023). *Youtube*. Von https://www.youtube.com/watch?v=048MNvc9suA abgerufen

[07] Aran, B. (06 2025). *GitHub*. Von https://github.com/aranbarri/pimesh abgerufen

[08] Marek, L. (09 2025). *GitHub*. Von https://github.com/open-mesh-mirror/batman-adv abgerufen

[09] Marek, L. (2025). *Open-Mesh*. Von https://www.open-mesh.org/projects/open-mesh/wiki abgerufen

[10] Chrony, W. (09 2025). *Chrony*. Von https://chrony-project.org/index.html abgerufen

[11] Schweizerische, E. (09 2025). *HERMES, Online*. Von https://www.hermes.admin.ch/de/projektmanagement/ergebnisse.html abgerufen

[12] Ossmann, M. (09 2025). *Great Scott Gadgets*. Von https://greatscottgadgets.com/hackrf/one/ abgerufen

[13] MediaTek. (2025). *MediaTek*. Von https://www.mediatek.com/products/broadband-wifi/rt5370 abgerufen

[14] Waveshare. (2025). *Waveshare*. Von https://www.waveshare.com/wiki/WM8960_Audio_HAT abgerufen

[15] Mumble, F. (2025). *Mumble Info*. Von https://forums.mumble.info/topic/1195-suggestion-multiple-server-connections/ abgerufen

[16] mum-rs, G. S. (2025). *GitHub*. Von https://github.com/mum-rs/mum abgerufen

[17] Yuan, C. (2024). *ResearchGate*. Von https://www.researchgate.net/publication/380031685_The_future_of_wireless_mesh_network_in_next-generation_communication_a_perspective_overview abgerufen

[18] Wikipedia. (2025). *Wikipedia*. Von https://en.wikipedia.org/wiki/IEEE_802.11s abgerufen

[19] Anja, V. B. (2025). *WIoT Group*. Von https://wiot-group.com/think/en/resources/mesh-networks-and-wireless-communication/ abgerufen

[20] Wikipedia, M. (2025). *Wikipedia*. Von https://en.wikipedia.org/wiki/Meshtastic abgerufen

[21] Wikipedia, B. (2025). *Wikipedia*. Von https://en.wikipedia.org/wiki/Bitchat abgerufen

[22] OpenAI, C. (2025). *ChatGPT*. Von https://chatgpt.com/ abgerufen

[23] DeepL, W. (2025). *DeepL*. Von https://www.deepl.com/de/write abgerufen

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ALSA ..... *Advanced Linux Sound Architecture*
AREDN ..... *Amateur Radio Emergency Data Network*
ARM ..... *Advanced RISC Machine*
bit ..... *binary digit*
BPMN ..... *Business Process Model and Notation*
CHF ..... *Schweizer Franken*
CLI ..... *Command Line Interface*
COTS ..... *Commercial Off-The-Shelf*
CPU ..... *Central Processing Unit*
DAC ..... *Digital-to-Analog Converter*
DAI ..... *Digital Audio Interface*
DHCP ..... *Dynamic Host Configuration Protocol*
ESP32 ..... *Espressif32*
ESSID ..... *Extended Service Set Identifier*
EU ..... *Europäische Union*
GB ..... *Gigabyte*
GHz ..... *Gigahertz*
GPIO ..... *General Purpose Input/Output*
GPS ..... *Global Positioning System*
GUI ..... *Graphical User Interface*
HAT ..... *Hardware Attached on Top*
HDMI ..... *High Definition Multimedia Interface*
I/O ..... *Input/Output*
I2C ..... *Inter-Integrated Circuit*
I2S ..... *Inter-IC Sound*
IBSS ..... *Independent Basic Service Set*
IoT ..... *Internet of Things*
IP ..... *Internet Protocol*
KI ..... *Künstliche Intelligenz*
LAN ..... *Local Area Network*
LED ..... *Light Emitting Diode*
LoRa ..... *Long Range*
MAC ..... *Media Access Control*

mAh ..... *Milliamperestunden*
MB ..... *Megabyte*
MEMS ..... *Micro-Electro-Mechanical Systems*
MKII ..... *Mark 2*
MOS ..... *Mean Opinion Score*
MTU ..... *Maximum Transmission Unit*
NTP ..... *Network Time Protocol*
OBS ..... *Open Broadcaster Software*
OLSR ..... *Optimized Link State Routing Protocol*
OS ..... *Operating System*
OSI ..... *Open Systems Interconnection*
OTG ..... *On-The-Go*
PCM ..... *Pulse Code Modulation*
PMR ..... *Private Mobile Radio*
PS ..... *PowerShell*
PTT ..... *Push-to-Talk*
RAM ..... *Random Access Memory*
RPi ..... *Raspberry Pi*
SBC ..... *Single-Board-Computer*
SD ..... *Secure Digital*
SDR ..... *Software Defined Radio*
SIP ..... *Session Initiation Protocol*
SoC ..... *System on Chip*
SRTP ..... *Secure Real-time Transport Protocol*
SSH ..... *Secure Shell*
SSID ..... *Service Set Identifier*
SW ..... *Software*
systemd ..... *system daemon*
TCP ..... *Transmission Control Protocol*
TLS ..... *Transport Layer Security*
UDP ..... *User Datagram Protocol*
USB ..... *Universal Serial Bus*
USD ..... *United States Dollar*
UTC ..... *Universal Time Coordinated*
VLC ..... *VideoLAN Client*
VNC ..... *Virtual Network Computing*

VoIP ..... *Voice over Internet Protocol*
WAN..... *Wide Area Network*
WiFi..... *Wireless Fidelity*
WLAN..... *Wireless Local Area Network*

Anhang A

Interview #1 vom 29.07.2025

Fragen an Uta:

- Wie gebe ich die Quelle eines Bildes an? Direkt in der Bildbeschriftung? (siehe Abb. 1)
  Antwort: mit Fussnote lösen und das Wort Abbildung mit Abb. abkürzen.
- Abgabe der DA per Mail an wen?
  Antwort: im Normalfall JA, jedoch bei Stefan nachfragen.

Kapitelstruktur und Anpassungshinweise von Uta:

Kapitel 1 | Informieren

- Kapitelbezeichnungen aussagekräftiger formulieren, z. B. statt „Informieren“: *Allgemeine Informationen zum Projekt und Ausgangslage*
- 1.1 Einführung: Der aktuelle Inhalt gehört nicht hierher, sondern später unter „Projektidee“. An dieser Stelle zuerst die Ausgangslage beschreiben
- 1.4 Problemstellung: Der Variantenvergleich (Tabelle) gehört erst deutlich später ins Dokument
- 1.5 Erste Projektidee / Lösungsansatz: Die vorhandene Grafik ist zu detailliert für diese Stelle, besser unter „Realisieren“ einfügen
- Ergänzend: 1.6 Grobe Projekt- oder Lösungsidee als eigenständiger Punkt

Kapitel 2 | Planung

- Zuerst Pflichtenheft mit Must-have und Nice-to-have-Kriterien
- Danach Zeitplan (SOLL) mit Meilensteinen als Tabelle
- Arbeitsjournal anschliessend integrieren
- „Technische Planung“ ggf. umbenennen in Analyse der Systemfunktionalität, mit Unterpunkten:
  - Voraussetzungen, bereitzustellende Arbeitsmittel
  - Rollen und Use Cases im System
  - Use Case Rolle 1
  - Use Case Rolle 2
  - Use Case Rolle 3
  - Funktionalität des Gesamtsystems
- Testumgebung: prüfen, ob sie an dieser Stelle bereits planbar ist

Kapitel 3 | Entscheiden

- Umbenennen in Variantenentscheid mit Begründung
- 3.1 Beschreibung der möglichen Hardware-Varianten
- 3.2 Variantenentscheid mit Begründung Hardware
- 3.3 Beschreibung der möglichen Software-Varianten
- 3.4 Variantenentscheid mit Begründung Software

Kapitel 4 | Realisieren

- Ggf. umbenennen in Technische Konzeption des Zielsystems.
- Schwerpunkt: Zusammenführung und Integration aller Einzelteile

Kapitel 5 | Kontrollieren

- Vorab neues Unterkapitel: Konzeption der Testumgebung
- Danach: Durchführung der Tests (Softwaretests, Funktionstests)

Kapitel 6 | Technische Umsetzung des mobilen Prototyps (NEU)

Kapitel 7 | Testen und Auswertung der Tests

- Zielerreichung (erreicht / nicht erreicht) dokumentieren

Kapitel 8 | Fazit (NEU)

- 8.1 Soll-Ist-Vergleich, ersetzt bisheriges „Zielerreichung nach Pflichtenheft“
  - Punkte, die nicht erreicht wurden, mit Begründung
- 8.2 Lessons Learned
- 8.3 Ausblick
  - zukünftige Arbeiten
  - Trends
  - Erweiterungsmöglichkeiten

Interview #2 vom 10.09.2025

Thema: Struktur und Feingliederung der Diplomarbeit

Rückmeldung und Empfehlungen von Uta:

Kapitel 1 | Informieren

- Struktur grundsätzlich in Ordnung
- Kapitelbezeichnungen sollten selbsterklärender und weniger generisch sein (z. B. „Allgemeine Informationen und Ausgangslage“ statt nur „Informieren“), falls dafür noch Zeit bleibt
- Dieser Punkt wurde bereits im Vorherigen Interview von Uta erwähnt

Kapitel 2 | Planung

- Aufbau grundsätzlich ebenfalls in Ordnung
- Neue Unterkapitelstruktur empfohlen:
- 2.1 Projektplanung und Rahmenbedingungen
  - 2.1.5 Risiken und Annahmen
  - 2.1.6 Risikomatrix (am besten als eigenes Unterkapitel)
  - 2.1.7 Annahmen (ebenfalls als eigenes Unterkapitel)
- 2.2 Zeitliche Projektplanung mit Feinkriterien
- 2.3 Wichtige Meilensteine chronologisch

Ab hier beginnt laut Uta die eigentliche Eigenleistung des Projekts. Deshalb empfiehlt sie, ab diesem Punkt ein neues Kapitel zu starten.

Kapitel 3 | Analyse der Systemfunktionalität (NEU)

- In diesem Kapitel sollen zwei echte Use Cases beschrieben werden:
  - Use Case 1: Aktor Funker
  - Use Case 2: Aktor Empfänger
- Ziel dieses Kapitels:
  - Zuerst eine Informationssammlung, was das System leisten soll und geplanten Funktionen
  - Anschliessend daraus konkrete Anwendungsfälle (Use Cases) ableiten:
    - Was macht der Funker mit dem System?
    - Was macht der Empfänger mit dem System?

Hinweis zu Kapitel 3:

- Die bisherigen Vergleiche technischer Varianten sind laut Uta keine Use Cases, sondern technische Vergleiche
- Deshalb sollen diese Inhalte ins neue Kapitel 4 verschoben und dort umbenannt werden

Kapitel 4 | Vergleich möglicher Varianten und technischer Komponenten

Hier können alle bisherigen Inhalte des bisherigen Kapitels 2.3.3 und Kapitel 3 (Variantenentscheid) eingefügt werden.

Empfohlene Struktur laut Uta:

- 4.1 Mögliche Hardware-Komponenten
  - 4.1.1 Einplatinencomputer mit SDR-Funkmodul (Funktionsweise)
  - 4.1.2 Einplatinencomputer mit WLAN-Mesh-Netzwerk (Funktionsweise)
    → inkl. Vergleich der Einplatinencomputer
  - 4.1.3 LoRa-WAN-basiertes Mesh-Netzwerk (Funktionsweise)
  - 4.1.4 Gegenüberstellung der technischen Komponenten mit Vor- und Nachteilen
  - 4.1.5 Variantenentscheid für die gewählte Hardware
  - 4.1.6 Begründung des Entscheids
  - 4.1.7 Vergleich der Audio-Hardware
  - 4.1.8 Variantenentscheid und Begründung der gewählten Audio-Hardware
- 4.2 Vergleich möglicher Software-Komponenten
  - 4.2.1 Vergleich Betriebssysteme
  - 4.2.2 Vergleich Mesh-Protokolle
  - 4.2.3 Vergleich VoIP-Software
  - 4.2.4 Entscheidung und Begründung der Softwarewahl
- 4.3 Wirtschaftlichkeitsbetrachtung der gewählten Komponenten

(entspricht bisherigem Kapitel 3.4 Kostenübersicht der gewählten Varianten)

Kapitel 5 | Technische Konzeption des Zielsystems

- Dieses Kapitel folgt auf die Variantenvergleiche und beschreibt die Systemarchitektur und das Zielsystem im Detail
- Struktur ist laut Uta grundsätzlich in Ordnung

Kapitel 6 | Realisierung und Umsetzung des Zielsystems

- Der Aufbau dieses Kapitels ist bereits sehr gut.
- Empfohlene Unterstruktur ist bereits vorhanden

Kapitel 7 | Planung der Testumgebung (Vorschlag von Uta)

Dieses Kapitel kann vor der Testdurchführung eingefügt werden.

Empfohlene Gliederung:

- 7.1 Heimnetzwerk
- 7.2 Feldumgebung
- 7.3 (optional) Übersicht über Testarten
- 7.4 Erstellung des Testkonzepts
- 7.5 Erstes Testkonzept – Softwaretests
- 7.6 Zweites Testkonzept – Funktionstests
- 7.7 Drittes Testkonzept – Feldtests mit Messungen
- 7.x Konzeptionelles Zusammenspiel der Komponenten

Kapitel 8 | Durchführung der Tests

Klare Trennung nach Testarten empfohlen:

- 8.1 Softwaretests durchführen
- 8.2 Funktionstests durchführen
- 8.3 Feldtests mit Messungen durchführen

Kapitel 9 | Auswertung und Fazit

Struktur ist laut Uta soweit in Ordnung.

Fokus:

- Zusammenfassung der Testergebnisse
- Bewertung der Zielerreichung
- Persönliches Fazit

Anhang B

boot/firmware/config.txt File

Angepasste und ergänzte Zeilen in **ROT** ersichtlich:

```
# WM8960 Audio HAT# For more options and information see
# https://www.raspberrypi.com/documentation/computers/config_txt.html
# Some settings may impact device functionality. See link above for details

# Uncomment some or all of these to enable the optional hardware interfaces
dtparam=i2c_arm=on
dtparam=i2s=on
#dtparam=spi=on

# Enable audio (loads snd_bcm2835)
dtparam=audio=off

# Additional overlays and parameters are documented
# /boot/firmware/overlays/README

# Automatically load overlays for detected cameras
camera_auto_detect=1

# Automatically load overlays for detected DSI displays
display_auto_detect=1

# Automatically load initramfs files, if found
auto_initramfs=1
```

```
# Enable DRM VC4 V3D driver
dtoverlay=vc4-kms-v3d
max_framebuffers=2

# Don't have the firmware create an initial video= setting in cmdline.txt.
# Use the kernel's default instead.
disable_fw_kms_setup=1

# Disable compensation for displays with overscan
disable_overscan=1

# Run as fast as firmware / board allows
arm_boost=1

[cm4]
# Enable host mode on the 2711 built-in XHCI USB controller.
# This line should be removed if the legacy DWC2 controller is required
# (e.g. for USB device mode) or if USB support is not required.
otg_mode=1

[cm5]
dtoverlay=dwc2,dr_mode=host

[all]
# WM8960 Audio HAT
dtoverlay=wm8960-soundcard
```

/usr/local/bin/ptt.py File

Python Skript zur Steuerung der PTT-Taste:

```
#!/usr/bin/env python3
import os
import RPi.GPIO as GPIO
import time
import subprocess

# Konfiguration aus env-Datei laden
env_path = "/etc/walkietalkie/ptt.env"
config = {}
with open(env_path) as f:
    for line in f:
        if "=" in line:
            k,v = line.strip().split("=",1)
            config[k] = v

PIN = int(config["GPIO_PIN"])
CARD_INDEX = config["ALSA_CARD_INDEX"]
CONTROL = config["ALSA_CAPTURE_CONTROL"]
DEBOUNCE = int(config["DEBOUNCE_MS"]) / 1000.0

GPIO.setmode(GPIO.BCM)
GPIO.setup(PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP)

last_state = GPIO.input(PIN)
```

```
try:
    while True:
        state = GPIO.input(PIN)
        if state != last_state:
            time.sleep(DEBOUNCE)
            if state == GPIO.LOW:  # Taste gedrückt
                subprocess.run(["amixer","-c",CARD_INDEX,"sset",CONTROL,"cap"])
            else:  # Taste losgelassen
                subprocess.run(["amixer","-c",CARD_INDEX,"sset",CONTROL,"nocap"])
            last_state = state
        time.sleep(0.01)
except KeyboardInterrupt:
    GPIO.cleanup()
```

/home/user01/mumble_fallback.py File

Python-Skript welches als Fallback dient:

```
#!/usr/bin/env python3
import os
import shutil
import socket
import subprocess
import time
import signal
import sys

# --- Konfiguration ------------------------------------------------------
USERNAME = "user01"   # Mumble-Benutzername anpassen
PASSWORD = ""         # optional, sonst "" lassen
SERVERS = [           # Reihenfolge = Prioritaet (oben = bevorzugt)
    ("pi-node-1.local", 64738),
    ("pi-node-2.local", 64738),
    ("pi-node-3.local", 64738),
]
CHECK_TIMEOUT = 1.0   # Sek. fuer Portcheck
RETRY_DOWN    = 3.0   # Sek. warten, wenn kein Server erreichbar
RECHECK_PRI   = 15.0  # Sek. Intervall, wenn auf Sekundaer, um Primary zu pruefen
RUN_ON_DESKTOP = True # setzt DISPLAY/XDG/DBUS fuer GUI-Client
MUMBLE_BIN = shutil.which("mumble") or "/usr/bin/mumble"
# ------------------------------------------------------------------------
```

```
def log(msg: str):
    print(msg, flush=True)


def is_up(host: str, port: int) -> bool:
    try:
        with socket.create_connection((host, port), timeout=CHECK_TIMEOUT):
            return True
    except OSError:
        return False


def pick_server():
    for host, port in SERVERS:
        if is_up(host, port):
            return host, port
    return None, None


def build_url(user: str, host: str, port: int, pw: str = "") -> str:
    # mumble://[user[:pass]@]host[:port]
    auth = user if not pw else f"{user}:{pw}"
    auth = (auth + "@") if auth else ""
    return f"mumble://{auth}{host}:{port}"


def launch_client(host: str, port: int) -> subprocess.Popen:
    env = os.environ.copy()
    if RUN_ON_DESKTOP:
        env["DISPLAY"] = env.get("DISPLAY", ":0")
        env["XDG_RUNTIME_DIR"] = env.get("XDG_RUNTIME_DIR", f"/run/user/{os.getuid()}")
        env["DBUS_SESSION_BUS_ADDRESS"] = env.get(
            "DBUS_SESSION_BUS_ADDRESS",
```

```
            f"unix:path={env['XDG_RUNTIME_DIR']}/bus"
        )
    url = build_url(USERNAME, host, port, PASSWORD)
    log(f"Starte Mumble-Client: {url}")
    return subprocess.Popen([MUMBLE_BIN, url], env=env)


def stop_client(proc: subprocess.Popen | None):
    """Versuche den laufenden Mumble-Prozess sauber zu beenden, notfalls hart killen."""
    if proc is None:
        return
    try:
        if proc.poll() is None:
            log("Beende laufenden Mumble-Client (SIGTERM)...")
            proc.terminate()
            try:
                proc.wait(timeout=5)
            except subprocess.TimeoutExpired:
                log("Prozess reagiert nicht → hartes Kill (SIGKILL).")
                proc.kill()
                proc.wait(timeout=3)
    except Exception as e:
        log(f"Fehler beim Beenden: {e}")


def main():
    log("Mumble Fallback-Launcher gestartet.")
    preferred = SERVERS[0]
    current: tuple[str, int] | None = None
    proc: subprocess.Popen | None = None
```

```
    # Sanfte Startverzoegerung, damit Netzwerk/Audio bereit sind
    time.sleep(2)

    try:
        while True:
            # Falls kein Client laeuft (neu starten oder nach Absturz)
            if proc is None or proc.poll() is not None:
                stop_client(proc)  # sicherheitshalber
                host, port = pick_server()
                if not host:
                    log("Kein Server erreichbar. Warte...")
                    time.sleep(RETRY_DOWN)
                    continue
                log(f"Verbinde zu {host}:{port}")
                current = (host, port)
                proc = launch_client(host, port)
                # kleine Pause, damit GUI/Verbindung initialisieren kann
                time.sleep(2)

            # Rueckwechsel auf Primary, falls wir gerade auf Sekundaer sind
            if current != preferred and is_up(*preferred):
                log("Primary ist wieder online → Wechsel zurueck auf Primary.")
                stop_client(proc)
                proc = None
                current = None
                continue

            # Wenn aktueller Server weg ist, sofort umschalten
            if current and not is_up(*current):
```

```
                log(f"Aktueller Server {current[0]} nicht erreichbar → Umschalten.")
                stop_client(proc)
                proc = None
                current = None
                continue

            # Schlafintervall: schneller, wenn wir auf Primary sind; gemuetlicher auf
Sekundaer
            time.sleep(3 if current == preferred else RECHECK_PRI)

    except KeyboardInterrupt:
        log("Beendet (KeyboardInterrupt).")
    finally:
        stop_client(proc)

if __name__ == "__main__":
    sys.exit(main() or 0)
```