Aufbau und Evaluation eines Media IP Echtzeitnetzwerkes

Bachelorarbeit

zur Erlangung des akademischen Grades Bachelor of Science

an der Fachhochschule Kiel

im Fachbereich Informatik und Elektrotechnik

im Studiengang Informationstechnologie

Autor: Anton Bracke

Abgabedatum: 14. Juli 2021

1. Betreuer: Prof. Dr. Robert Manzke
2. Betreuer: Prof. Dr. Steffen Prochnow

Zusammenfassung

Audiosignale in Echtzeit zu übertragen ist eine der Anforderungen, die man unter anderem in Funkhäusern oder Tonstudios stellt. Lange wurde für diese Übertragung extra Infrastruktur in Form von proprietären Systemen angeschafft und verbaut. In den letzten Jahren verbreiteten sich Ethernet-Netzwerke aufgrund des Fortschritts der Digitalisierung immens. Durch ihre immer größer werdenden Bandbreiten, sowie immer geringer werdenden Latenzen, erlauben sie es, dass Audiosignale mit Latenzen von unter 10 Millisekunden übertragen werden können. Für einen solchen Einsatz kann größtenteils auf handelsübliche Hardware, welche oft bereits vorhanden ist, zurückgegriffen werden. Zusätzlich werden lediglich Netzwerkgeräte, wie zum Beispiel Netzwerkswitches, mit Paket-Priorisierung für eine besonders geringe Latenz und Endgeräte mit speziellen Netzwerkkarten, welche über eine interne Hardwareuhr verfügen, für eine synchrone Übertragung vorrausgesetzt. Sämtliche weitere Eigenschaften können durch spezielle Software und Treiber, welche nach Standards wie AES67 arbeiten, erfüllt werden. Dieses hat unter anderem den großen Vorteil, dass durch die Standarisierung von Ethernet-Netzwerken und dem AES67 Standard verschiedenste Geräte unterschiedlichster Anbieter ihre Audiosignale miteinander austauschen können. Welche dieser Komponenten notwendig sind und auf welche Eigenschaften hierbei ein besonderer Fokus gelegt werden sollte, wird im Rahmen dieser Arbeit behandelt.

Abstract

Transmitting audiosignals with close to real-time speed is one of the requirements many broadcast and recording studios have. For a long time this kind of transmissions was done by using additional proprietary hardware from different manufacturers. Digitization is one of the reason ethernet networks gained an enormous amount of prominence. Based on their ever growing bandwidths and ever decreasing latencies, they can be used to transmit audio with latencies below 10 milliseconds. By using such existing networks only little effort has to be expended to establish an audio network as most of the hardware can be kept. Normally it is sufficient to replace some network switches to support packet prioritization and to attach some new enddevices which include a network card with an internal hardware clock to provide a performant basis for time synchronization. Any additional request can be realised by software and drivers which support standards like AES67. Standards similar to AES67 and the ones used by ethernet networks bring the benefit that hardware from many different suppliers can interact with each other. This paper addresses which of those components are required and should be focused on for a minimal audio over IP network.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Abkürzungsverzeichnis

AES	Audio Engineering Society
ALSA	Advanced Linux Sound Architecture
AoIP	Audio over IP
AVB	Audio Video Bridging
BBB	Beaglebone Black
CI	Continuous Integration
CPU	Central processing unit
CSS	Cascading Style Sheets
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
GPIO	General Purpose Input/Output
GPL	General Public License
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTML	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
IP	Internet Protocol
ISO	International Standards Organization
JS	JavaScript
JSON	JavaScript Object Notation
MAC	Media-Access-Control-Address / Media-Access-Code
mDNS	multicast DNS
OSI	Open Systems Interconnection
PC	Personal Computer
PCI	Peripheral Component Interconnect
PCM	Pulscodemodulation
PTP	Precision Time Protocol
QoS	Qualtity of Service
RAM	Random-Access Memory
RFC	Request for Comments
RT	Realtime
RTP	Realtime Transport Protocol
RTSP	Real-Time Streaming Protocol
SAP	Session Announcement Protocol
SD-Karte	Secure Digital Memory - Karte
SDP	Session Description Protocol
SPI	Serial Peripheral Interface
SSH	Secure Shell
UDP	User Datagram Protocol
WLAN	Wireless Local Area Network

1 Einleitung

1.1 Motivation

In Zeiten von *Internet of Things* (IoT), bei denen sämtliche Geräte von der Heizung über das Küchenlicht bis hin zum Kühlschrank mit Hilfe von Smartphone Anwendungen, den sogenannten Apps, zentral gesteuert werden können, ist ein Bestandteil essentiell: Das zu Grunde liegende Netzwerk. Erst durch ein vorhandenes Netzwerk, welches die einzelnen Endgeräte miteinander verbindet, entsteht die Möglichkeit, dass die Geräte untereinander kommunizieren und aus der Ferne von einem Anwender gesteuert werden können. Dabei sind die Anforderungen an die Bandbreite und auch die Latenz des Netzwerkes stetig gestiegen.

Durch diese geringe Latenz und große Verbreitung von Computernetzwerken ist auch das Interesse gestiegen, die Übertragung von Audio Signalen, wie sie zum Beispiel in Tonstudios erforderlich ist, über ein bereits existierendes Netzwerk mit echtzeitähnlichen Geschwindigkeiten zu realisieren. Durch die Wiederverwendung dieser Infrastruktur können zum einem Kosten gespart werden, aber auch die Schnittstellen zwischen Geräten unterschiedlicher Hersteller lassen sich so einfacher realisieren, da durch die bereits genormte Netzwerkhardware nur noch eine kompatible Software entwickelt und verwendet werden muss.

Einer dieser Standards zum Streamen von Audio mit hoher Qualität und gleichzeitig geringer Latenz über ein Netzwerk ist AES67. Der AES67 Standard wurde seitdem er 2013 von der *Audio Engineering Society* veröffentlicht wurde, bereits von verschiedenen wichtigen Anbietern in die eigenen Geräte integriert und durch die Interoperabilität mit vorhandenen Protokollen wie Livewire, Q-LAN, Ravenna und Dante ist auch der professionelle Einsatz von AES67 bereits möglich. [3]

Da der Standard mittlerweile in der professionellen Anwenderwelt als etabliert gilt, entstand das Interesse ein minimales Netzwerkes, über welches Audio mit dem AES67 Standard übertragen wird, zu betreiben. Dabei soll die Leistung des Netzwerkes ermittelt werden.

1.2 Ziel dieser Arbeit

Diese Arbeit beschäftigt sich mit diversen aktuell verfügbaren Hardware sowie Software Komponenten, welche zum Betrieb eines Audionetzwerkes verwendet werden können und versucht dabei einen Einblick zu geben, auf welche Eigenschaften bei einer solchen Anwendung Rücksicht genommen werden sollte und auf welche Aspekte man zu Gunsten der

Kosten für Aufbau und Betrieb möglicherweise einen geringeren Fokus legen kann.

1.3 Aufbau der Arbeit

Diese Arbeit besteht aus mehreren Teilen. Sie beginnt mit einigen Grundlagen Rund um den Aufbau eines Audionetzwerkes. Dabei wird neben dem grundsätzlichen Netzwerk mit seinen Schichten auch unter anderem auf die Netzwerkprotokolle PTP, Ravenna, AVB und den AES67 Standard eingegangen. Zusätzlich werden das Betriebssystem Linux, der Kernel des Linux-Betriebssystems, die Funktion von Treibern bei Linux und die Advanced Linux Sound Architecture, sowie die Themen Git und Docker behandelt.

Im Kapitel Media-IP-Netzwerk werden verschiedene Komponenten vorgestellt, welche für den Aufbau eines minimalen Audionetzwerkes verwendet werden können. Mit einer Auswahl dieser Komponenten wird daraufhin ein mögliches Netzwerk konzipiert.

Die mit der Inbetriebnahme verbundenen Schritte zur Einrichtung eines Audionetzwerkes und die dafür notwendigen Software und Treiber Installationen werden im Kapitel 4 aufgeführt.

In drei verschiedenen Tests gegliedert, werden in dem Kapitel Evaluierung, die zuvor gewählten Komponenten, in unterschiedlichen Szenearien getestet. Zunächst wird der Basisversuchsaufbau zusammen mit den verbunden Testmethoden und Instrumenten erläutert. Um einige Eigenschaften, welche von einem Audionetzwerk verlangt werden, gezielt zu testen, wird der Basisaufbau jeweils nur minimal angepasst. Der erste Test prüft einen Point-to-Point Aufbau mit einer Soundkarte.

Ein weiterer Test untersucht einen Point-to-Point Aufbau, welcher mit Hilfe von zwei GPIO Pins den Start der Ausgabe eines Audiosignals anzeigt und so ohne Einflüsse durch eine Soundkarte Ergebnisse liefert.

Um den Empfang mit mehreren Endgeräten zu untersuchen wird zum Schluss der bisher vorhandenen Aufbau vom einem Sender und einem Empfänger, um einen weiteren Empfänger ergänzt.

Für jeden Test wird jeweils der Versuchsaufbaus festgelegt. Anschließend wird der Versuch durchgeführt. Mit Hilfe der gesammelten Messergebnisse, wird dann eine Auswertung gemacht und mögliche Verbesserungen, sowie Probleme in einer Diskussion besprochen.

Zuletzt werden die Ergebnisse der Arbeit zusammengefasst und auf Grundlage dieser eine Empfehlung für den Aufbau eines Media-IP-Echtzeitnetzwerkes gegeben. Der folgende Ausblick zeigt schlussendlich, wie die Arbeit und Erkenntnisse dieser Thesis weiter genutzt werden können.

2 Grundlagen

2.1 Computer

Ein *Personal Computer* (PC) besteht unter anderem aus einer Hauptplatine, Arbeitsspeicher (*Random Access Memory*, RAM) und einem Hauptprozessor (*Central Processing Unit*, CPU). Einige der gängigen Prozessorarchitekturen sind *x86* und *ARM*.

2.2 Linux

Mit Linux bezeichtnet man ein Betriebssystem, welches auf dem Linux-Kernel basiert. Der Linux-Kernel wurde 1991 vom finnischem Studenten *Linus Torvalds* erstmals veröffentlicht und wird seither von Firmen, Non-Profit Organisationen und vielen Freiwilligen weiter entwickelt. Diese Zusammenarbeit am Kernel ist möglich, da der gesamte Quellecode quelloffen unter der *GNU General Public License* zur Verfügung steht und somit frei genutzt und angepasst werden kann. Um den Linux Kernel zu nutzen wird dieser meist mit zusätzlicher Software aus dem GNU Projekt verpackt und als sogenannte Distribution bereitgestellt. Zu den bekanntesten Distributionen zählen Debian, Ubuntu und Fedora. Neben den von Computern weit bekannten x86 und x86-64 Architekturen unterstütz Linux auch viele andere Architekturen, weshalb über die Jahre auch zusätzlich zu den typischen Computern viele andere Plattformen ein Betriebssystem verwenden, welches auf dem Linux-Kernel basiert. Mit 72,72% [32] Marktanteil im Mai 2021 ist Android das nach der Internetnutzung meist genutzte mobile Betriebsystem und basiert ebenfalls auf dem Linux-Kernel. [12] Zudem ist Linux unter anderem in Form der OpenWRT [24] oder FritzOS [20] Betriebssysteme auf Routern vertreten. Mit der wachsenden Bedeutung von Cloud Systemen gewinnt auch Linux weiter an Bedeutung. Im Juni 2021 werden laut W3Techs über 90% [36] der Webseiten mit den Softwaren Nginx, Apache, Cloudflare Server und LiteSpeed betrieben, welche auf einem Linux Betriebsystem installiert werden. [35] [6]

2.2.1 Kernel

Bis auf wenige Ausnahmen ist der Linux-Kernel in der Sprache C geschrieben und wird dabei als monolithischer Kernel bezeichtnet, da er als eine große Software betrieben wird, die sowohl sämtliche Funktionen für die Verwaltung von Speicher und Prozessen, aber auch Treiber und zusätzlich Funktionen für die Verwendung von Hardware beinhaltet. Im Zusammenhang mit einem Linux-Betriebssystem kann ein drei Schichtenmodell verwendet werden. Dabei bildet die unterste Schicht die Hardware des Systems, darüber befindet sich der Kernel und alle Module die im *Kernelspace* laufen und die oberste Schicht beschreibt den *Userspace*, der sämtliche Programme des Benutzers beherbergt, die durch den Kernel verwaltet werden. [19]

Seit Version 2.6 arbeitet der Linux Kernel größenteils präemptiv. Dabei wird jeder Prozess bis auf wenige Ausnahmen nach einer gewissen Zeit unterbrochen, sodass auch anderen Prozessen die Möglichkeit gegeben wird, zeitnahe ihre Aufgaben zu erledigen. Um einen vollständig präemptiven Kernel zu erhalten, kann eine sogenannte *RT-Patch* Version des Kernels verwendet werden. [11]

2.2.2 ALSA

ALSA ist die Abkürzung für *Advanced Linux Sound Architecture*, welches das Basis-Soundsystem von Linux ist. [1] ALSA beinhaltete eine Sammlung von Treibern, welche als Kernelmodule zusammen mit dem ALSA Framework im Kernel enthalten sind. Somit dient ALSA Anwendungen als Schnittstelle um Sound-Hardware, meist in Form einer Soundkarte, zu nutzen und zu steuern. Es werden Ein- und Ausgabekanäle zur Verfügung gestellt, welche zur Aufnahme und Wiedergabe verwendet werden können. Zudem erlaubt ALSA die Benutzung einer Vielzahl von Funktionen, welche auf die einzelnen Kanäle angewendet werden können, bevor eine Anwendung oder eine Soundkarte diese empfängt. So lassen sich unter anderem Kanäle mixen oder durch einen Equalizer anpassen.

2.3 Netzwerk

Durch Netzwerke lassen sich viele Endgeräte, sogenannte Knoten, miteinander verbinden. Eines der größten und bekanntesten Netzwerken ist das *Internet*, welches Knoten auf der ganzen Welt miteinander verbindet, sodass ein weltweiter Austausch von Daten ermöglicht wird. [39]

Ein Endgerät, das mit einem Netzwerk kommunizieren möchte, benötigt mindestens einen Netzwerkadapter. Jeder dieser Adapter, besitzt zur Adressierung eine Media-Access-Control Adresse (MAC). Die MAC-Adresse wird durch den Hersteller eines Netzwerkadapters fest an diesen gebunden. Daher wird die MAC-Adresse auch als physische Kennung bezeichnet. [39]

Da Netzwerke meist hochgradig dynamisch aufgebaut und verschachtelt werden können, werden zusätzlich dynamische Adressen in Netzwerken benutzt. Diese Adressen des Internet-Protocols (IP) lassen sich statisch oder dynamisch über das Betriebssystem eines Endgerätes einem Netzwerkadapter zuordnen. [39]

Um die Vergabe von IP-Adressen automatisch und dynamisch durchzuführen, wird das *Dynamic Host Configuration Protocol* (DHCP) eingesetzt. Dabei fragt ein Endgerät eine neue Adresse an und ein DHCP-Server, welcher im Netzwerk auf Anfragen lauscht, vergibt

eine noch freie Adresse aus seinem Vorrat an das Endgerät. [39]

2.3.1 Hub, Switch und Router

Zur Verbindung verschiedener Knoten eines Netzwerkes, gibt es drei Geräte.

Hub Ein Hub bietet bei der Verbindung die primitivste Art des Paketaustausches. Wenn er ein Paket auf einem seiner Ports empfängt, vervielfacht er dieses und schickt es über alle Ports wieder heraus. Diese Art des Versandes ist ein sogenannter *Broadcast*. So können alle angeschlossenen Geräte alle Pakete empfangen. Ein Nachteil des Broadcasts ist, dass auch Geräte, welche nicht an dem Empfang des Paketes interessiert sind, trotzdem dieses Paket erhalten und somit die Leistung des Hubs mindern. [39]

Switch Im Gegensatz zu einem Hub, wertet ein Switch mit Hilfe der MAC-Adresse der verbunden Geräte an seinen Ports aus, an welchen der einzelnen Ports ein Paket weitergeleitet werden soll. [39]

Router Ein Router ist in der Lage mehrere Netzwerke miteinander zu verbinden. Router werden in den meisten privaten und gewerblichen Netzwerken als Verbindungsglied zum Internet verwendet. Sie erkennen anhand der Zieladresse, dass ein Paket in ein anderes Netzwerk weitergeleitet werden muss und können zusätzlich Pakete filtern (Firewall), Pakete im Namen des Routers weiterleiten und auch Pakete, welche an den Router adressiert sind, an ein hinter dem Router befindliches Endgerät umadressieren und weiterleiten. [39]

2.4 OSI-Modell

Die Übertragung eines Netzwerkes lässt sich in sieben Schichten nach dem OSI-Modell (vgl. Abbildung 1) einteilen.

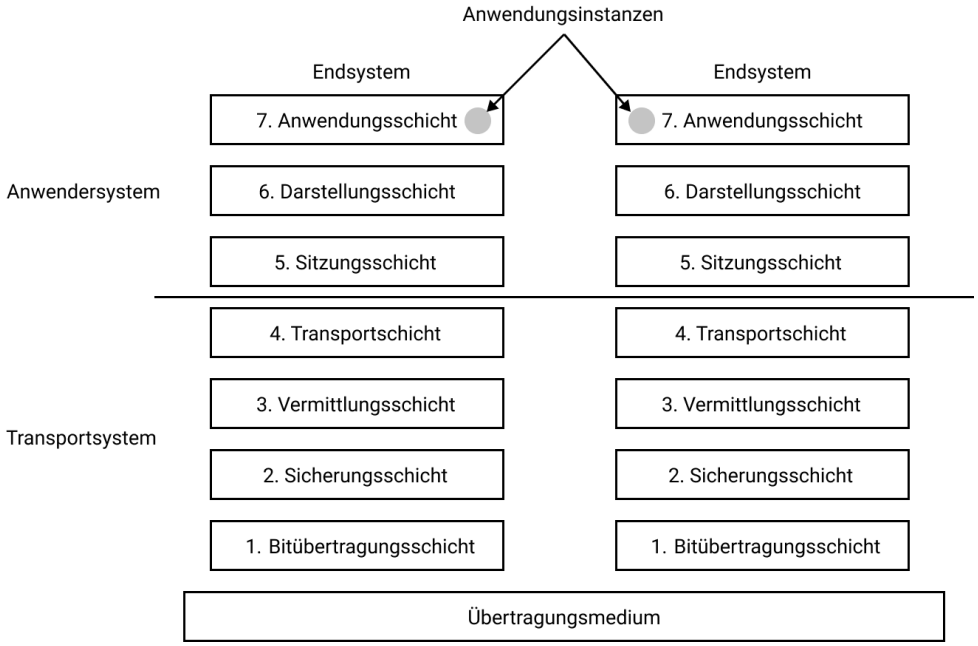

Abbildung 1: OSI Modell (nach [38])

Durch die erste Schicht, auch Layer genannt, wird die für das Netzwerk verwendete Hardware dargestellt. Der darauf folgenden Sicherungsschicht kann die Adressierung von Netzwerkadaptern mit Hilfe der MAC-Adresse zugeordnet werden. Die dritte Schicht wird als Vermittlungsschicht bezeichnet und beherbergt die dem Internet IP-Protokoll zugehörigen Funktionalitäten, welche das Routen von Paketen organisiert. Mit der vierten Transportschicht werden Datenpakete über sogenannte Ports Anwendungen zugeordnet. Die obersten Schichten des Modells stellen Funktionen für die Anwendungen bereit, welche über das Netzwerk kommunizieren. [39]

2.4.1 Unicast & Multicast

Unicast wird in Netzwerken für die Adressierung eines Paketes an einen einzigen Empfänger verwendet. [7]

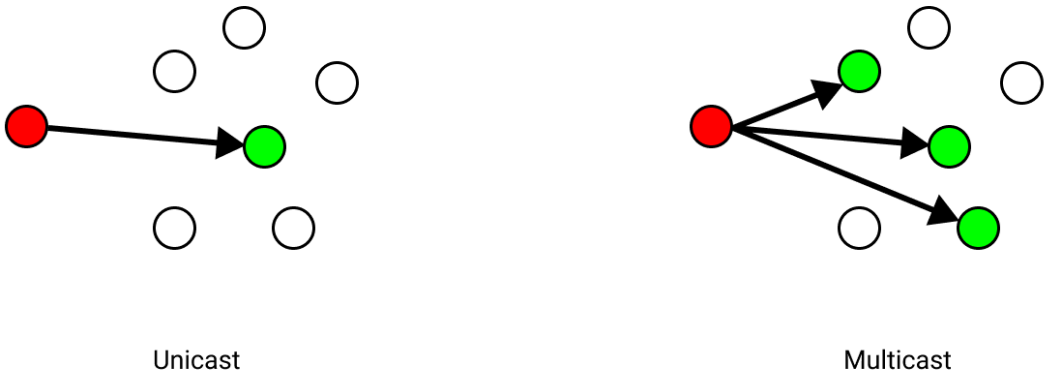

Abbildung 2: Unicast vs Multicast (Nach [37])

Multicast hingegen wird dann eingesetzt, wenn ein Paket gleichzeitig an mehrere Empfänger gesendet werden soll. [37] Anstatt gezielt ein Endgerät in die Metadaten des Netzwerkpaketes einzutragen wird beim Senden von Multicastpaketen auf eine Pseudo-Adresse zurückgegriffen. Über das *Internet Group Management Protocol* (IGMP) Protokoll einigen sich die Netzwerkteilnehmer dabei auf eine gemeinsame Pseudo-Adresse, sodass Pakete bereits durch ihre Metadaten erkannt werden können. [39]

2.5 Netzwerk-Protokolle

2.5.1 DNS

Das *Domain Name System* (DNS) erlaubt die Verwendung von Namen (Domains) anstelle von IP-Adressen beim Verbindungsaufbau zu Netzwerkteilnehmern. DNS arbeitet dabei analog zu einem Telefonbuch, indem es eine Liste von Domains mit zugehörigen IP-Adressen vorhält und auf Anfrage Einträge aus dieser ausgibt. [2]

2.5.2 HTTP

Über das *Hypertext Transfer Protocol* können die Inhalte von Webseiten, welche über einen Webbrowser aufgerufen werden können, übertragen werden. Für jede Anfrage an einen *HTTP*-Server muss eine neue Verbindung aufgebaut werden, da *HTTP* zustandslos arbeitet.

Durch die große Verbreitung von *HTTP*-Infrastruktur werden auch viele Schnittstellen

für verteilte Systeme über *HTTP* realisiert. Eine dieser Schnittstellen Standards ist *Representational State Transfer* (REST). [9]

2.5.3 IEEE 802.1AS - PTP

Das *Precision Time Protocol* (PTP) ist ein Protokoll zur Synchronisation mehrerer Uhrzeiten in einem Netzwerk. Dabei können Genauigkeiten von wenigen Nanosekunden erreicht werden. Ein Master verteilt das Zeitsignal im Netzwerk, sodass dieses von mehreren Slaves empfangen werden kann. Obwohl es Softwareimplementierungen für eine PTP-Unterstützung gibt, wird eine Hardwareunterstützung durch das verwendete Netzwerkgerät empfohlen, da so eine höhere Genauigkeit erreicht werden kann. [13]

2.6 IEEE 802.1BA - AVB

Audio Video Bridging (AVB) ist eine Sammlung von Standards für die Übertragung von zeitkritischen Audio- oder Videodaten. Dafür baut AVB auf Ethernet-Netzwerken auf und setzt somit an der zweiten Schicht des OSI-Modells (vgl. Abbildung 1) an. Für die AVB-Funktionen werden Endgeräte und Switches benötigt, die die AVB-Standards unterstützen, welche im IEEE 802.1BA [14] Standard zusammengefasst sind.

2.7 AES67 Standard

Mit AES67 legte die Audio Engineering Society 2013 einen Standard fest, wie hoch performante Übertragungen von Audio über IP-Netzwerke vorzugsweise arbeiten sollten. Dabei baut AES67 im Gegensatz zu *AVB* auf der dritten Schicht des OSI-Modells (vgl. Abbildung 1) auf. AES67 wurde zu Gunsten der Interoperabilität der bereits existierenden Netzwerk-Technologien Dante, Ravenna, AudioLan, Q-Lan entwickelt. Dabei werden im AES67 Standard folgende Punkte festgelegt: [3]

- Verwendung des PTP-Protokolls für die Zeit-Synchronisation
- Geräte verwenden die PTP-Zeit als Taktgeber
- Latenz zwischen Sender und Empfänger von weniger als 10ms
- Transport der Audiosignale über das IP-Protokoll in der Version 4
- Bevorzugte Behandlung der Netzwerkpakete über Quality-of-Service (QoS) Einteilung in Prioritätslevel
- Einsatz von IGMP um Multicast-Gruppen zu koordinieren
- Streams werden über RTP (siehe Absatz 2.7.1) übertragen

2.7.1 RTP

Das *Real-Time Transport Protocol* (RTP) ist ein Protokoll zur Übertragung von Echtzeitdaten, wie Audio, Video oder Simulationsdaten mit Hilfe von Multicast oder Unicast. RTP ist spezifiziert in der RFC 3550. [31]

2.7.2 RTSP

Durch das *Real-Time Streaming Protocol* (RTSP) können Datenströme von Protokollen wie RTP gesteuert werden. RTSP ist spezifiziert in der RFC 2326. [28]

2.7.3 SDP

Mit Hilfe des *Session Description Protocols* (SDP) können Metainformationen für die Übertragung von Multimedia ausgetauscht werden. Dabei können unter anderem Codecs, Transportprotokolle und -adressen ausgetauscht werden. Es wird im *RFC 4566* beschrieben. [30]

2.7.4 SAP

Das *Session Announcement Protocol* (SAP) kann verwendet werden, um Sitzungskonfigurationen, wie die des SDP Protokolls, über Multicast mit anderen Teilnehmern auszutauschen. Es wird im *RFC 2974* definiert. [29]

2.7.5 mDNS

Das *multicast DNS* erlaubt die Veröffentlichung von Details zu Netzdiensten ohne das dafür eine spezielle Konfiguration im Netzwerk erforderlich ist. [9]

2.8 Git

Git ist ein Tool zur verteileten Versionsverwaltung von Dateien. Es wird von vielen Entwicklern für die gemeinsame Arbeit am Quellcode von Software verwendet. [27] Dabei lässt sich jedes Projekt in einem *Repository* speichern.

Github Besonders bekannt ist die Online-Plattform *Github* für die Entwicklung und Veröffentlichung von Open-Source Projekten. [16] Github bietet den Entwicklern verschiedene Funktionalitäten, um unabhängig von Ort und Zeit gemeinsam Software zu entwickeln. So kann jeder Github Benutzer unter anderem eine Kopie eines Projektes, einen so genannten *Fork*, erstellen. An diesem *Fork* können Anpassungen oder neue Funktionen entwickelt werden und anschließend kann mit Hilfe eines *Pull-Requests*, eine Anfrage an das originale Projekt gestellt werden, diese Anpassungen zu übernehmen.

Continuous Integration Zur Gewährleistung einer konstanten Softwarequalität werden zusammen mit vielen Git-Projekten *Continuous Integration*-Systeme (CI) verwendet. So lässt sich *Github Actions*, die CI der Plattform von *Github*, zum Beispiel so konfigurieren, dass die Änderungen eines *Pull Requests* automatisch getestet werden und nachdem die Änderungen akzeptiert wurden, kann die Anwendung des Projektes automatisch gebaut und veröffentlicht werden.

2.9 Docker

Docker ist eine Software zur Containervirtualisierung. Im Gegensatz zu virtuellen Maschinen, welche einen virtuellen Computer mit vollständigem Betriebssystem benötigen, benutzen Container bei der Containervirtualisierung den Kernel des Hostsystems. Dies hat den Vorteil, dass Container ressourcenschonend arbeiten. Das Dateisystem eines Containers wird getrennt von dem Dateisystem des Hosts in einem *Image* gespeichert. Ein solches *Image* lässt sich vervielfältigen und kann auf andere Hostsysteme übertragen werden. So kann zum Beispiel eine bestimmte Anwendung mit allen ihren Abhängigkeiten und Einstellungen in einem Container installiert werden und anschließend verteilt werden. Wenn die Anwendung schließlich auf einem anderem Hostsystem zum Einsatz kommen soll, so muss lediglich *Docker* installiert und das *Image* als Container gestartet werden, ohne das eine weitere Installation der eigentlichen Anwendung notwendig wird. [15]

3 Media-IP-Netzwerk

3.1 Komponentensuche

Ein Media-IP-Netzwerk, welches Audiosignale in Echtzeit übertragen soll, benötigt einige notwendige Komponenten. Mindestens ein Endgerät, welches als Sender eines Audiosignales dient und ein Endgerät, welches als Empfänger genau dieses Signal annimmt, sind für einen Grundaufbau erforderlich. Da die Leistung von Funkverbindungen wie bei einer Wireless-LAN Verbindung meist stark von der Empfangsstärke abhängt und diese zudem oft eine höhere Latenz im Vergleich zu einer Kabelverbindung haben, sind für Echtzeitnetwerke kabelgebundene Verbindungen zu bevorzugen.

Die Synchronisation der Endgeräte erfolgt über das Precision Time Protocol (PTP). Für die Verwendung von PTP muss die Netzwerkkarte des Endgerätes den PTP-Standard *IEEE 802.1AS* unterstützen und auf jedem Gerät, welches an der Audioübertragung beteiligt ist, muss ein PTP-Slave betrieben werden. Pro Netzwerk wird außerdem ein PTP-Master benötigt, welcher nicht zusammen mit einem Slave auf einer Netzwerkkarte laufen kann. Somit wird ein weiteres Endgerät für den Betrieb eines PTP-Masters im Netzwerk erforderlich.

Da bei mehr als zwei Endgeräten ein einfaches Kabel als Verbindung nicht ausreicht, werden für solche Netzwerke Switches verwendet. Neben den Audiopaketen werden im Normalfall zusätzlich verschiedenste andere Pakete über ein Netzwerk versendet. Diese Pakete können unter anderem für die Konfiguration und den Betrieb des Netzwerkes erforderlich sein oder werden gegebenenfalls für Anwendungen anderer Nutzer benötigt. Um die Latzen bei der Audioübertragung gering zu halten, lassen sich die Audiopakete durch einen geeigneten Switch priorisieren.

Durch die Verwendung eines Switches mit mehreren Ports ist es möglich einen vorhandenen Router und einen weiteren Computer mit dem Netzwerk zu verbinden. Der Router kann dem Netzwerk als DHCP-Server dienen und so den Endgeräten automatisch IP-Adressen vergeben, wodurch ein manuelles Einstellen der Adressen an den Endgeräten entfällt. Außerdem lässt sich über den Router eine Verbindung zum Internet herstellen, welche besonders bei der Installation neuer Pakete auf den Endgeräten hilfreich ist. Der weitere Computer lässt sich für den Zugriff auf die Endgeräte per Remote-Terminal Anwendungen wie der Secure Shell (SSH) verwenden.

Bei der Suche nach geeigneten Komponenten für das Medianetzwerk wurden Geräte bevorzugt, die sich über den Handel frei beziehen lassen. So konnte soweit wie möglich sichergestellt werden, dass der Aufbau eines solchen Netzwerkes mit vergleichbarer Hard-

ware nachbaubar ist.

3.2 Konzept zum Aufbau eines minimalen Netzwerkes

Ein Netzwerk zur Übertragung eines Audiosignales mit Hilfe von AES67 könnte hierfür aus drei Endgeräten und einem Netzwerkswitch bestehen.

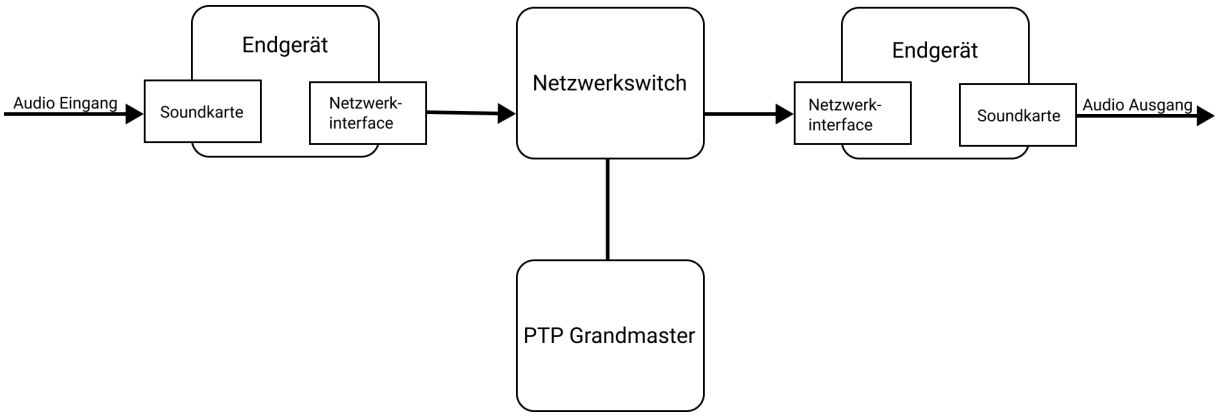

Abbildung 3: Aufbau eines minimalen Audionetzwerkes

Ein Endgerät nimmt dabei ein Audiosignal per Soundkarte auf und sendet dieses über das Netzwerk an ein zweites Endgerät, welches wiederum das Signal über seine Soundkarte abspielt. Ein drittes Endgerät dient dem Netzwerk als PTP-Master und unterstützt bei der Synchronisierung der Zeiten der einzelnen Teilnehmer.

3.3 Endgeräte

Als Endgerät eignet sich jeder Computer, welcher über eine Netzwerkkarte mit Hardwareseitiger *IEEE 802.1AS* Unterstützung verfügt und somit PTP kompatibel ist. Bei Desktopcomputern oder Servern lassen sich diese Netzwerkkarten meist über eine PCI-Schnittstelle nachrüsten. Zusätzlich sollte der Computer je nach Anwendung die Möglichkeit bieten, eine Soundkarte mit Ein- und Ausgängen anzuschließen oder diese direkt enthalten. Während dieser Arbeit wurde unter anderem ein Desktopcomputer mit einer Intel-Netzwerkkarte und eine gängige USB-Soundkarte verwendet. Da Desktopcomputer für das reine Streamen von Audio im Normalfall die Anforderungen an ihre Leistung großzügig erfüllen und daher in der Anschaffung nicht unerhebliche Beträge kosten können, wurden zusätzlich Einplatinencomputer eingesetzt. Der weit verbreitete Raspberry Pi kam dabei nicht in Frage, da dieser kein PTP mit seinem Netzwerkinterface bereitstellt.

3.3.1 BeagleBone-Black

Ein ähnlicher Einplatinencomputer zu dem Raspberry PI, welcher allerdings mit seinem Netzwerkinterface PTP unterstütz, ist der *BeagleBone-Black* (BBB). Dieser ist ein Einplatinencomputer, dessen Hardwaredesign frei und kostenlos verfügbar ist und wird im Handel für ca. *45 US-Dollar* vertrieben.

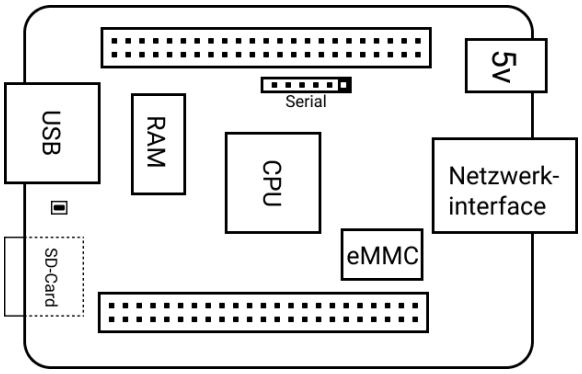

Abbildung 4: BeagleBoneBlack

Neben *512 MB* RAM verfügt der BeagleBone-Black über einen *AM3358 Arm Cortex-A8* Prozessor von *Texas Instruments*, welcher mit *1 GHz* getaktet ist. Für die Speicherung von Daten verfügt der BeagleBone über einen Onboard eMMC Flashspeicher und einen SD-Karten Slot. Ein USB sowie Mini-HDMI Anschluss ist direkt auf der Platine verbaut. Über 69 digitale GPIO Pins und sieben analoge Eingänge lassen sich eine Vielzahl von zusätzlichen Komponenten anbinden. So können unter anderem digitale Ein-, Ausgänge, SPI und serielle Schnittstellen angeschlossen werden. Zudem lassen sich ganze Platinen auf den eigentlichen BeagleBone-Black stecken und mit sogenannten Capes können dann komplette Erweiterungen, wie zum Beispiel eine Soundkarte, verwendet werden.

3.3.2 Desktopcomputer

Bei dem verwendeten Desktopcomputer handelt es sich um einen Rechner mit Intel Prozessor und *16 GB* Arbeitspeicher. Als Betriebssystem war ein *Ubuntu 20.04* installiert, welches mit einem Linux Kernel der Version *5.4* betrieben wurde. Da die auf dem Mainboard verbaute Netzwerkkarte keine PTP-Unterstützung bot, wurde zusätzlich eine Intel Netzwerkkarte des Typs *I210-T1* [8] verbaut.

3.4 Netzwerkswitches

Es gibt besondere Netzwerkswitches die den Quality-of-Service (QoS) Standard, also das Priorisieren der Netzwerkpakete durch das Einteilen in verschieden Level, unterstützen. Oft werden auch weitere zusätzliche Funktionen mit bereitgestellt, welche speziell für priorisiertes und synchronisiertes Streaming von Audio- und Videodaten vorgesehen sind.

So unterstützen einige Switches automatische Multicast Adressen Zuordnungen oder auch Methoden für das Reservieren von Streams und Zeit-Synchronisationen.

3.4.1 Motu AVB Netzwerkswitch

Der Motu AVB Netzwerkswitch verfügt über sechs 1-Gigabit Ports für die AVB Kommunikation nach dem *IEEE 802.1* Standard und einen weiteren Ethernet-Port um ein bestehendes Netzwerk anzuschließen. Der Switch verwendet intern einige nicht öffentlich bekannte Techniken, durch welche er die verbundenen Geräte bei ihrer Übertragung von Audio und Videodaten auf den Netzwerk-Layern zwei und vier aktiv unterstütz.

4 Software und Treiber zur Audioübertragung

Um von einem Gerät ein Audiosignal, welches zum Beispiel aus einer Datei geladen wird, in Echtzeit auf ein zweites Gerät zu übertragen, welches wiederum diesen Stream empfängt und in eine Datei speichert, sind neben einer eingerichteten Netzwerkverbindung auch weitere Treiber und Software zu installieren. Durch bestimmte Versionsabhängigkeiten und benötigte Bibliotheken dieser Anwendungen bedarf es einiger Vorbereitungen.

Als Betriebssystem für den BeagleBone-Black wurde Linux verwendet. Zusätzlich wurden ein AES67 Treiber samt Hilfsprogramme, der Treiber der verwendeten Soundkarte und die Tools für PTP installiert.

4.1 PTP Master

Ab der Version *3.0* des Linux-Kernels wird PTP mit Hilfe des notwendigen Treibers unterstüzt. Dabei ist zu beachten, dass PTP pro Netzwerkkarte entweder als Master oder Slave betrieben werden kann, jedoch ein paralleler Betrieb im Master und Slave Modus nur mit dem Loopback *lo* Netzwerkinterface möglich ist.

Mit dem Befehl *ethtool -T eth0* lässt sich prüfen, ob die Hardwareunterstützung von PTP für die Netzwerkkarte, in diesem Fall *eth0* aktiviert ist [25]. Sollte dies der Fall sein lässt sich über den Befehl *ptp4l -i eth0* ein Daemon starten, der die Zeit der PTP-Hardware-Uhr (PHC) mit der Netzwerkkarte *eth0* synchronisiert und somit einen Master für das Netzwerk startet.

4.2 AES67 Treiber

Für AES67 standen zwei quelloffene Treiber zur Wahl. Zum einen existiert ein Treiber von *Stefan Neufeldt* [22], da dieser allerdings keine PTP Unterstützung enthält, fiel die Wahl auf einen Treiber von *Andrea Bondavalli*.

Der gewählte AES67 Treiber von *Andrea Bondavalli*, welcher auf der Plattform *Github* [5] veröffentlicht wurde, besteht aus drei Teilen. Als Basis wird ein Kerneltreiber für Ravenna verwendet [4], welcher AES67 kompatibel ist. Dieser wurde ursprünglich von der Firma *Merging* entwickelt [34]. Da das zugehörige Userspace Modul *Butler* der Firma Merging nicht unter einer OpenSource Lizenz steht, entwickelte *Andrea Bondavalli* zwei alternative Module. Hierzu gehört ein Daemon, der im Hintergrund läuft und für die Steuerung des Kernelmodules verantwortlich ist. Um die *REST* Schnittstelle des Daemon benutzerfreundlich aufrufbar zu machen, wurde zusätzlich eine Weboberfläche entwickelt, welche neben verschiedensten Einstellungsmöglichkeiten auch eine Übersicht automatisch entdeckter Streams darstellt.

4.2.1 Vorraussetzungen

Für den Betrieb des Treibers wird ein Linux Betriebssystem mit einem Kernel der Version 4.10 oder neuer, der *GCC* Compiler [10] in der Version 7 oder neuer, *cmake* und die *boost* Bibliotheken vorausgesetzt. Zusätzlich wird eine Ubuntu oder Debian Distribution empfohlen, da hierfür ein Hilfsskript mit sämtlichen zu installerenden Softwarepaketen bereits existiert.

4.2.2 Komponenten

Kernelmodul Das Kernelmodul lässt sich als ALSA Treiber registrieren und stellt anschließend eine virtuelle Soundkarte zur Verfügung. Des weiteren ermöglicht es den Empfang und Versand von RTP Audiopaketen, welche an die Soundkarte gekoppelt werden. Die Synchronisation der Audioübertragung wird mit Hilfe einer Interrupt-Loop durch den integrierten PTP Slave betrieben. Über eine Netlink-Verbindung lassen sich mit einem Userspace Modul sämtliche Einstellungen der Audiostreams anpassen.

Daemon Der Daemon stellt das zentrale Steuerungselement des AES67 Treibers da. Er kommuniziert mit dem Kernelmodul und kann dieses anweisen bis zu 64 Multicast oder Unicast Quellen und Empfänger für Audioübertragungen zu erstellen. Während einer aktiven Übertragung bearbeitet der Daemon ankommende SDP Nachrichten und sendet für eigene Streams die notwendigen SDP Nachrichten ebenfalls an andere Teilnehmer. Über eine REST-Schnittstelle lässt sich der Daemon konfigurieren und überwachen. Unter anderem können Einstellungen des Daemons, wie zum Beispiel das aktuelle Loglevel angepasst werden. Des weiteren kann der Status des PTP-Slaves abgefragt werden und auch dessen Einstellungen lassen sich anpassen. Auch das Hinzufügen und Editieren von eingehenden und ausgehenden Streams, sowie das Überwachen von eingehenden Streams lässt sich über die REST-Schnittstelle realisieren. Um das Anlegen von neuen eingehenden Übertragungen zu vereinfachen unterstütz der Daemon die Discovery Protokolle SAP und mDNS und gibt hierfür beim Aufrufen seiner Schnittstelle gefundene Übertragungen zurück. Auch ein RTSP Client und Server wird durch den Daemon verwaltet, um so die Verwaltung von SDP Dateien über die *DESCRIBE* und *ANNOUNCE* Methoden nach dem Ravenna Standard zu unterstützen. Um mit mehreren Knoten über das Multicast System zu arbeiten verwendet der Daemon zudem IGMP für die Behandlung von SAP, PTP und RTP Sitzungen.

Web-Oberfläche Die Web-Oberfläche des AES67 Treiber ist nach dem Prinzip des *JAMStacks* aufgebaut.

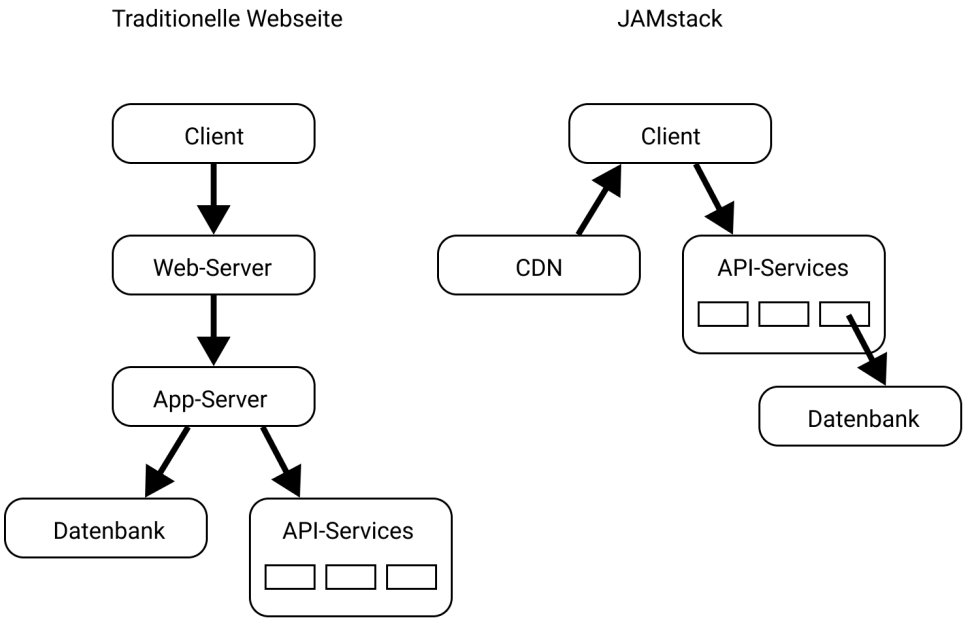

Abbildung 5: Vergleich traditionelle Webseite zu JAMStack

Statische Javascript, HTML und CSS Dateien, werden von einem Content-Delivery-Network (CDN), in diesem Fall einem einfachen Webserver, welcher im Daemon enthalten ist, ausgeliefert. Wenn ein Browser die Dateien und somit die Web-Oberfläche geladen hat, fragt diese über die REST-Schnittstelle des Daemon Daten ab, welche dann in die Web-Oberfläche eingefügt werden. Der Vorteil eines solchen Aufbaus besteht darin, dass im Gegensatz zu einer traditionellen Webanwendung die Web-Oberfläche vollständig unabhängig von der zu steuernden Logik entwickelt werden kann. Durch eine solche Trennung wird nicht nur die Entwicklung vereinfacht, sondern auch das Risiko von Sicherheitslücken wird verringert und auch die Wartbarkeit des System wird optimiert. Man könnte sich zum Beispiel entscheiden die gesamte Webanwendung neuzuentwickeln ohne das dafür an der hinterliegenden Logik der REST-Schnittstelle etwas angepasst werden muss.

Für den AES67 Treiber wurde bei der Entwicklung auf das React Framework zurückgegriffen. React wird von Facebook entwickelt und in den meisten Fällen wird eine React Anwendung mit der Software Webpack gebündelt und optimiert, sodass die entstehenden Dateien von einem Webserver an jeden möglichen Browser ausgeliefert werden können.

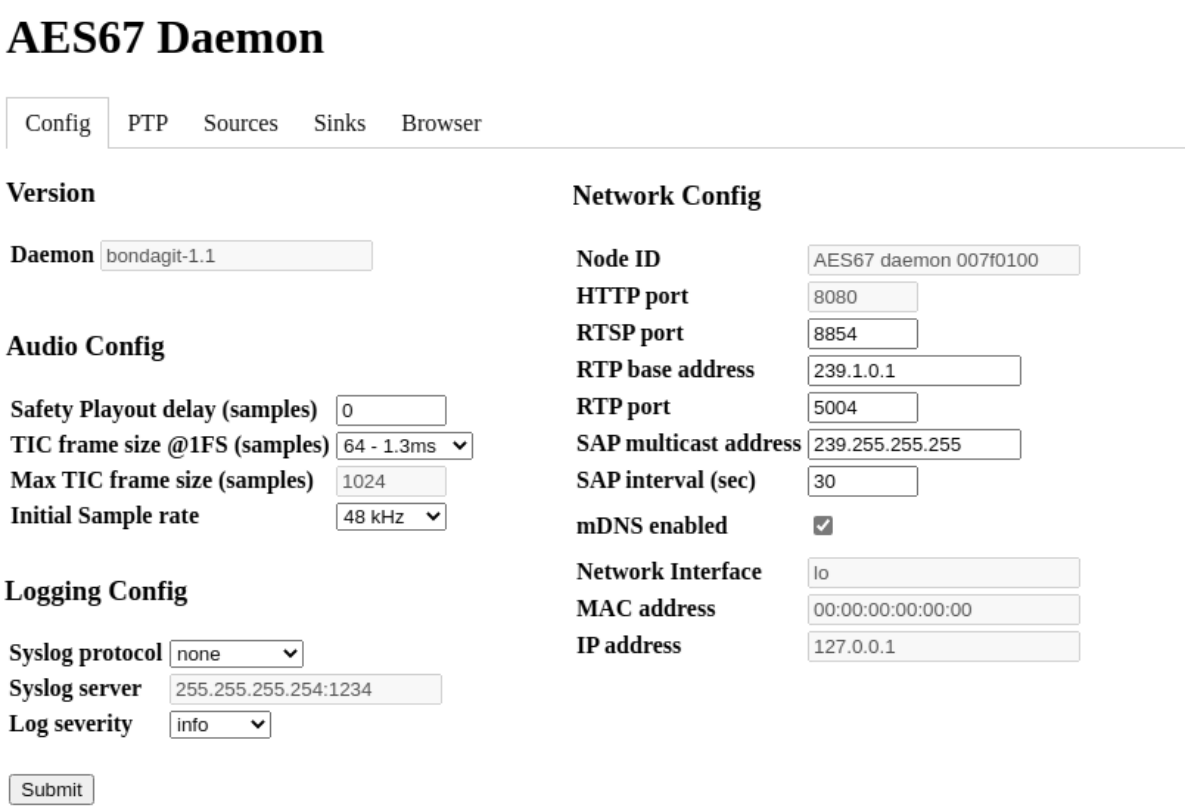

Abbildung 6: Web-Oberfläche des AES67 Treibers

4.2.3 Installation

Nachdem die benötigten Softwarepakete installiert sind, beginnt die eigentliche Installation der AES67-Software. Hierfür existiert ebenfalls ein Skript, welches zu erst den Ravenna Treiber mit dem Tool *make* kompiliert. Als nächstes baut das Skript den Daemon und zuletzt installiert es die Web-Oberfläche in einem Ordner innerhalb des Daemon Ordners, sodass der Daemon die Web-Oberfläche später zur Verfügung stellen kann.

4.2.4 Betrieb

Um nach dem Hochfahren den Treiber automatisch zu laden und den Daemon im Hintergrund laufen zu lassen, wurde ein sogenanntes *init.d*-Skript unter dem Pfad */etc/init.d/aes67* erstellt (siehe Listing 3). Über den Befehl *sudo update-rc.d aes67 defaults* wurde anschließend der automatisch Start ds Skriptes aktiviert.

Die Basiskonfiguration des Daemons wird über eine JSON-Datei, welche unter */opt/aes/daemon.conf* abgelegt wird, verwaltet. Sobald der Daemon mit dieser Konfigurationsdatei über das *init.d*-Skript gestartet wurde, kann über die lokale IP-Adresse, des Computers auf dem die Installation durchgeführt wurde, mit der Url *http://[computer-ip]:8080/* die Web-Oberfläche für die weitere Konfiguration aufgerufen werden.

4.2.5 Anpassungen am AES67 Treiber

Im Rahmen dieser Arbeit wurden einige Anpassungen in Form von Optimierungen am AES67 Treiber von *Andrea Bondavalli* vorgenommen. Zunächst wurde von dem originalen Git-Repository ein Fork erstellt und anschließend wurde für jede Anpassung ein Pull-Request geöffnet.

Download eines fertigen Paketes Ein Vorteil des JAMStacks ist es, dass man die Browser-Anwendung separat bauen kann. Hierdurch entsteht die Möglichkeit das Bauen der Weboberfläche zentral und einmalig für sämtliche Architekturen vorzunehmen. So muss bei der Installation des Treibers nur ein fertig gebautes Paket heruntergeladen und entpackt werden und muss nicht, wie bei dem in C geschriebenen Teil des Treibers, direkt auf dem Ziel-System kompiliert werden. Das Bauen der Weboberfläche dauerte im Schnitt auf einem System, wie zum Beispiel dem BeagleBone-Black, aufgrund seiner geringen Rechenleistung weit über 30 Minuten. So wurde eine Continous-Integration bei Github eingerichtet, welche nach jeder Änderung an der Weboberfläche diese baut und als herunterladbare Datei verpackt. Ein weiterer Vorteil dieser Methode ist, dass auf dem System, auf welchem der Treiber installiert werden soll, keine weiteren Programme, wie Node.js, zum Bauen der Software notwendig sind.

Austausch von Webpack durch Vite Das Packen von Javascript-Modulen mit Webpack kann im Regelfall mehrere Minuten dauern. Während der Bau der Produktivumgebung einmalig durchläuft, wird bei der Entwicklung auf einen Hot-Reload zurück gegriffen. Bei einem Hot-Reload wird die Anwendung immer dann neu gebaut, wenn eine der relevanten Quelldateien angepasst wurde. Teilweise wird hier auch nur der Bereich neugebaut, welcher seit dem letzten Bau verändert wurde. Diese Art des Bauens spart besonders in der Entwicklung viel Zeit und kann zum Beispiel kleine Änderungen in der Benutzeroberfläche für den Entwickler sofort sichbar machen. Mit der Veröffentlichung der Software Vite, welche eine Alternative zu Webpack darstellt, wurden die Zeiten für das Neubauen einer Seite von Minuten auf Millisekunden verringert. Aus diesem Grund wurde ein Pull-Request während dieser Arbeit erstellt, welcher Webpack durch Vite ersetzten sollte.

Nach einer Überprüfung durch den Maintainer des originalen Projektes, wurden diese Änderungen schlussendlich übernommen.

4.3 Soundkarte: CTAG Face 2|4

Da der BeagleBone-Black keine eigene Soundkarte verbaut hat, muss eine externe verwendet werden. Hierfür ließe sich die HDMI-Schnittstelle verwenden, jedoch ist dieser Aufbau umständlich, da für jeden BeagleBone je ein Bildschirm für die reine Audioausgabe verwendet werden würde. Alternativ kann eine USB Soundkarte oder ein Cape mit dem BeagleBone benutzt werden. Mit der *CTAG Face 2|4* Soundkarte existiert eine professionelle Soundkarte, welche neben vier Ausgabekanälen zusätzlich über zwei Eingabekanäle verfügt. Für die Verwendung wird sie einfach auf den BeagleBone-Black aufgesteckt. Anschließend muss der Linux-Kernel für das BeagleBone Betriebssystem um einen Treiber für das *CTAG Face 2|4* erweitert werden.

Einbau in den Kernel Der Treiber der Soundkarte wird direkt als Kernel-Module integriert. Hierfür hat *Henrik Langer* den offiziellen BeagleBone Kernel in der Version 4.4 angepasst und seine Änderungen in das offizielle Projekt einbauen lassen. Für den *4.19* Kernel wurde 2020 von *Niklas Wantrupp* der Treiber aktualisiert. Im Rahmen dieser Arbeit wurde der Treiber zunächst an die Änderungen des Kernel-Updates auf die Version 5 angepasst [26]. Dabei lies sich durch das Entfallen einiger ALSA-Komponenten und Funktionen die SPI Schnittstelle, über welche die Soundkarte mit dem BeagleBone kommuniziert, nicht ohne erheblichen Aufwand anpassen.

Device-Tree Overlay Für die passende Konfiguration der einzelnen Pins, welche zur Ansteuerung der Soundkarte verwendet werden sollen, wird ein Device-Tree Overlay verwendet. In diesem Overlay wird jedem Pin ein Modus wie GPIO, SPI oder serielle Schnittstelle zugewiesen. Anschließend wird das Overlay durch den Befehl *dtc -O dtb -o /lib/firmware/BB-CTAG.dtbo -b 0 -@ BB-CTAG.dts* kompiliert und über die Zeile *uboot_overlay_addr1=BB-GPIO-50-00A0.dtbo* in der Datei */boot/uEnv.txt* nach einem Neustart geladen.

4.4 Entwicklung eines BBB Images

Das Betriebssystem mit dem passenden Kernel, sowie sämtliche Software und Treiber auf einem BeagleBone-Black zu installieren benötigt spezifische Einstellungen und ist mit einem hohen Zeitaufwand verbunden. Um diesem Problem entgegen zu wirken wurde im Rahmen dieser Arbeit ein spezielles Image erstellt. Diese Image lässt sich auf eine SD-Karte schreiben, sodass ein BeagleBone von dieser booten kann. Damit bei kleinen Anpassungen oder Fehlern nicht die gesamte vorherige Arbeit an dem Image verloren ging, wurden hierfür Skripte erstellt.

Inhalt des Images Es entstand ein Image für den BeagleBone-Black, welches auf einem Root-Dateisystem der Distribution Debian in der Version *Buster 10.9* von *Robert C. Nelson*, einem Maintainer des BeagleBone Linux, basiert. Es wurde ein 4.19 Linux-Kernel mit RT-Patch und enthaltenem CTAG Face 2|4 Treiber eingebaut. Der Device-Tree für das CTAG Face 2|4 wurde kompiliert und auf dem Image hinterlegt.

U-Boot ist eine Software die für den BeagleBone-Black als Bootloader verwendet wird. Um den Device-Tree der CTAG-Soundkarte zu laden, musste die U-Boot Konfiguration entsprechend angepasst werden.

Zusätzlich wurde der AES67 Treiber installiert und so eingerichtet, dass der Treiber bim Hochfahren des BeagleBone-Blacks automatisch geladen und anschließend der Daemon des Treibers gestartet wird. Somit ist auch die Weboberfläche des Treibers des BeagleBones automatisch über dem Netzwerkport *8080* erreichbar. Da der Port *8080* Anfangs von einem Nginx Webserver, welcher indem verwendetem Root-Dateisystem enthalten war, belegt wurde, musste dieser anschließend noch deinstalliert werden.

Ebenso wurde die Software Node.js entfernt, da diese automatisch nach dem Booten ein Programm im Hintergrund startete, welches nicht benötigt wurde und einen hohen Leistungsbedarf aufwies.

Die geringe Rechenleistung des BeagleBone-Blacks lässt Aufgaben wie das Kompilieren des Linux-Kernels sehr zeitanspruchsvoll werden. Dieser Problematik kann durch das sogenannte *Cross-Compilen* entgegen gewirkt werden. Dabei wird anstelle des *GCC* Compilers ein besonderer Cross-Compiler, wie er in der *Linaro* Toolchain enthalten ist, verwendet. Mit Hilfe eines solchen Compilers lässt sich auf einer x86 Architektur eines leistungstärkeren Computers, ein ARM fähiges Programm bauen.

Durch einen bekannten Fehler, der beim Cross-Compilen des Linux Kernels auftritt, werden beim Bauen falsche Teile, des Systems auf dem der Cross-Compiler ausgeführt wird, mit in den Kernel des Zielsystems eingebaut. Aufgrund dieses Fehlers lassen sich auf dem Zielsystem später weitere Treiber nicht kompilieren, sodas auch der AES67-Treiber cross-kompiliert werden musste.

Automatischer Bau des Images Zur Verbesserung der Wiederverwendbarkeit der Skripte zum Erstellen des Images, wurden diese in einen Docker Container eingebaut. So musste bis auf die Docker Software keine weitere Software, wie zum Beispiel der spezielle Compiler direkt auf dem eigenen Computer installiert werden. Zusätzlich unterstütz Docker durch seine Abstraktion auch bei der Wahl der richtigen Abhängigkeiten bei der Installa-

tion von Software.

Veröffentlicht wurde die Sammlung der Skripte und das daraus resultierende Ergebnis in Form eines Git-Repositories. Durch die Continous- Integration von Github lies sich das Image bei jeder gepushten Änderung automatisch neubauen und steht anschließend zum Download bereit.

Installation und Betrieb des Images Um das Image auf einer SD-Karte zu installieren wurde das Programm *dd* unter Linux verwendet. Der BeagleBone-Black konnte daraufhin von der SD-Karte gestartet werden, indem vor dem Verbinden mit einer Spannungsquelle ein Knopf neben dem SD-Kartenhalter gedrückt wurde. Schwierigkeiten beim Booten von der SD-Karte konnten durch eine an den BeagleBone angeschlossene serielle Verbindung überprüft werden. Über diese Verbindung gab U-Boot verschiedenste Informationen zum Boot-Vorgang aus. Nachdem dieser erfolgreich durchlief, konnte über so genannte *printk* Ausgaben das Starten des Linux Kernel weiter beobachtet und bei einem Fehler mit Hilfe eines Crashdumps der Grund dieses Fehlers untersucht werden. Ein häufiger Grund für Fehler während des Boot-Vorgangs mit U-Boot war dabei die mangelnde Verbindung der kleinen Taster zu den Kontakten der SD-Karte. Dieser Problem lies sich durch leichtes Andrücken der Taster im Normalfall beheben.

5 Evaluierung

Nachdem ein minimales Netzwerkkonzept aufgestellt wurde, sollte dieses in drei unterschiedlichen Tests evaluiert werden. Zusätzlich zu einem Test der Latenz zwischen dem Senden und Empfangen eines Audiosignales wurde auch ein Test zur Überprüfung der die Synchronisierung dieses Signales geplant. Desweiteren sollte das Senden an mehrere Empfänger untersucht werden.

5.1 Zielsetzung

Idee und Ziel der Evaluierung ist es, die verwendeten Komponenten aus dem Konzept für ein minimales Netzwerk 3.2, auf ihr Zusammenspiel und die verschiedenen Eigenschaften der einzelnen Komponenten zu untersuchen. Durch diese Tests soll unter anderem festgestellt werden, wie stabil und somit reproduzierbar die einzelnen Programme und Treiber sowie auch Hardwarekomponenten miteinander interagieren. Zusätzlich soll die Latenz, also die Zeit zwischen der Übergabe eines Audiosignales an den AES67 Treiber bis zum Ausgeben dieses Signales über eine Soundkarte oder über einen GPIO-Pin, gemessen werden.

5.2 Basisaufbau

Anhand des Konzeptes wurde, wie im Abschnitt 4 beschrieben, die notwendige Software auf drei BeagleBone-Blacks und einem Linux-Computer installiert. Anschließend wurden diese über ein Kabel mit dem verwendeten Motu Netzwerkswitch verbunden (vgl. Abbildung 3). Auf einem der BeagleBone-Blacks wurde der zentrale PTP-Master gestartet.

5.2.1 Oszilloskop

Es wurde ein PicoScope 3206DMSO Zweikanal Oszilloskop mit einer Bandbreite von 200 MHz, einer Auflösung von 8 Bits bei 1 GigaSample/s und einem Pufferspeicher von 512 MS verwendet. Das Oszilloskop bietet zusätzliche Funktionen, wie einen Funktionsgenerator, einen Generator für anwenderdefinierte Wellenformen, sowie einen 16-Kanal Logikanalysator. Das Gerät ist ein PC-Oszilloskop und lässt sich über einen USB-Anschluss mit der offiziellen Software *PicoScope* des Herstellers an einem Computer mit Windows, Linux oder MacOS betreiben.

Für diese Evaluierung wurde das Oszilloskop an einem Windows Computer mit der Software in der Version *7.0.75* betrieben, da diese im Gegensatz zu der Linux-Version stabiler lief und zusätzlich einige weitere Funktionen in der Benutzeroberfläche verfügbar waren.

5.3 Machbarkeit

Der AES67 Treiber, sowie der Treiber der CTAG-Soundkarte, lassen sich wie in Punkt 4.4 beschrieben, mit einem Linux Kernel der Version 4.19 samt RT-Patch betreiben. Der Betrieb eines PTP-Masters scheint ohne besondere Einstellungen umsetztbar zu sein. Ob sich eine Synchronisation der Endgeräte herstellen lässt muss in den Tests weiteruntersucht werden.

Das *PicoScope 3206DMSO* Oszilloskop hat mit seiner Abtastrate von 1 GS/s die Möglichkeit eine Zeitdifferenz von < 10 ms, welche bei einer AES67-Übertragung im Netzwerk gewünscht wird, zu messen und sollte daher für die geplanten Versuche nutzbar sein.

Die Schwierigkeit und Aufgabe der Tests wird darin bestehen, die Latenzen des Systems so zu ermitteln, dass mögliche Faktoren wie eine Latenz der Soundkarte oder Verzögerungen beim Setzten des GPIO Pins so gering wie möglich gehalten werden oder im besten Fall vollständig umgangen werden. Sollten hierbei Fehler und unerwartete Messwerte auftreten, könnte es durch die vielen betroffenen Schichten, wie zum Beispiel die Netzwerkübertragung, Einstellungen des AES67 Treibers oder Eigenschaften des Kernels, zu Schwierigkeiten bei der Eingrenzung von möglichen Fehlern kommen.

5.4 Test I - Point-to-point Übertragung mit Soundkarte

Bei dem ersten Versuch soll die Latenz einer Audioübertragung gemessen werden. Dazu soll ein Audiosignal zum einen direkt von einem BeagleBone auf einer CTAG-Soundkarte ausgegeben werden und zeitgleich erst über das Netzwerk per AES67 an einen weiteren BeagleBone übertragen werden, um dann ebenfalls per CTAG-Soundkarte ausgegeben zu werden.

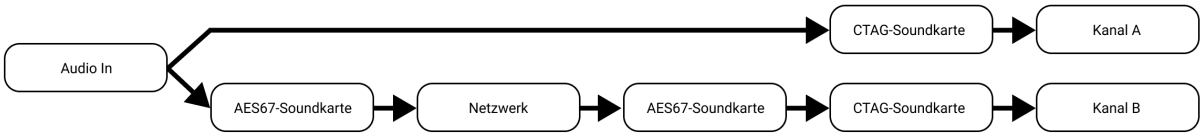

Abbildung 7: Test I - Weg des Audiosignales

Da das Signal möglichst zeitgleich an die virtuelle Soundkarte des AES67 Treibers und direkt an eine CTAG-Soundkarte geschickt werden soll, müsste so die Zeitdifferenz zweier Punkte an identischen Stellen der aufgezeichneten Tonspuren messbar sein. Dafür werden die beiden Eingangskanäle des Oszilloskops jeweils an eine der beiden CTAG-Soundkarten angeschlossen. Diese so gemessene Zeitdifferenz sollte somit genau dem zusätzlichen Weg, den das Audiosignal über das Netzwerk und bei der Weiterleitung an die CTAG-Soundkarte länger brauchte (vgl. Abbildung 7), entsprechen.

5.4.1 Versuchsaufbau

Neben dem PTP-Master und einem Netzwerkswitch wird der Aufbau um zwei weitere BeagleBone-Blacks mit CTAG-Soundkarte ergänzt.

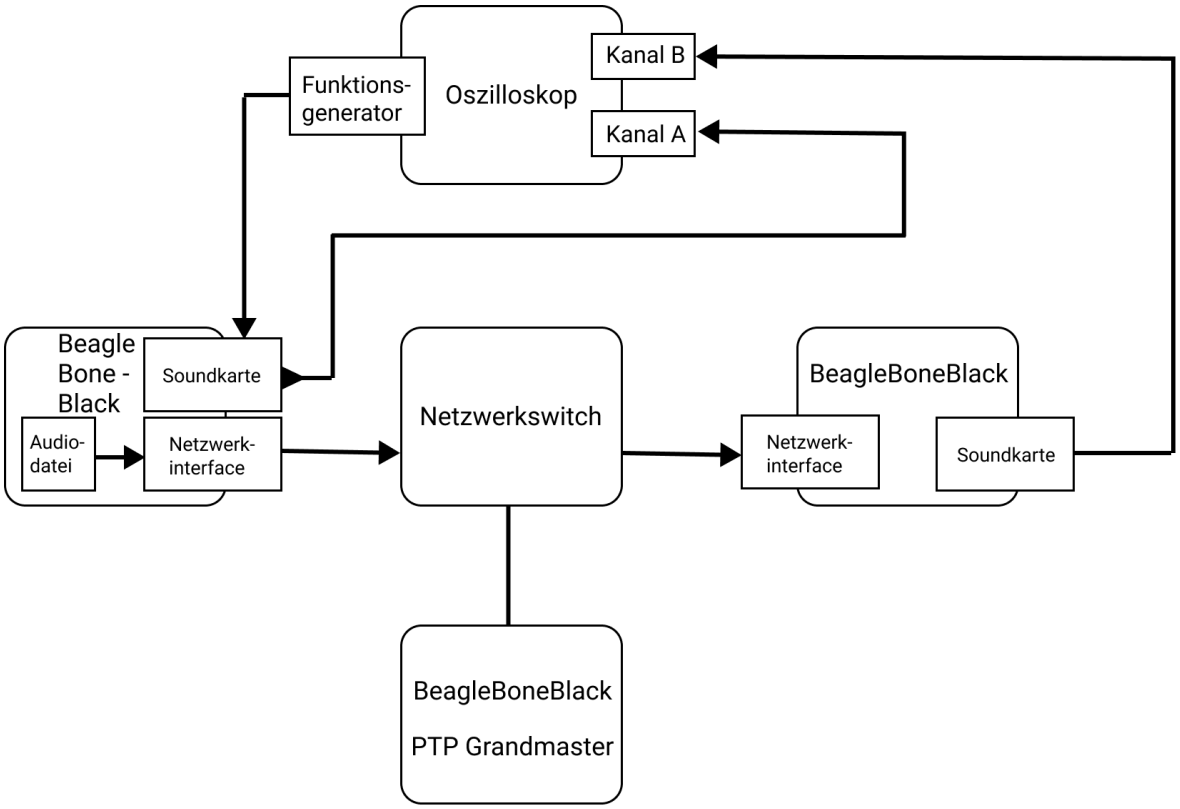

Abbildung 8: Test I Versuchsaufbau - Point-to-point Übertragung mit Soundkarte

An einen Soundkarteneingang des ersten BeagleBones soll der Funktionsgenerator des Oszilloskop angeschlossen werden, damit eine konfigurierte Welle des Funktionsgenerators als Audiosignal über die CTAG-Soundkarte aufgenommen werden kann, wie in der Abbildung 8 zu erkennen ist. Diese Signal soll anschließend parallel durch das weiterleiten an die AES67-Soundkarte als Übertragung gesendet werden und auf der CTAG-Soundkarte erneut ausgegeben werden. Durch das Ausgeben über die CTAG-Soundkarte kann ein Tastkopf des Oszilloskops das Audiosignal wieder abfangen, um dieses später als Trigger und Referenz verwenden zu können.

Ein weiterer BeagleBone-Black soll im Netzwerk als AES67-Empfänger eingesetzt werden. Damit das Oszilloskop die bei der AES67-Übertragung empfangene Tonspur ebenfalls aufzeichen kann, soll auch von diesem BeagleBone das Audiosignal durch einen Tastkopf aufgenommen werden, in dem es von der virtuellen Soundkarte an die CTAG-Soundkarte weitergeleitet wird.

Durch das Abspielen der Tonspur über die zwei identischen CTAG-Soundkarten sollen mögliche Verzögerungen egalisiert werden, sodass möglichst nur die Latenz der AES67-Übertragung gemessen werden kann. Eine Verzögerung bei Verwendung der CTAG-Soundkarten könnten unter anderem durch die Konvertierung der digitalen Daten auf einen analogen Soundpegel entstehen.

Einstellung AES67 Übertragung Für den Versuch soll die Option *"Max samples per packet"* bei der Quelle des AES67 Streams auf einen möglichst geringen Wert von von 6 samples gestellt werden. Aus diesem Wert folgt dabei eine Verzögerung von circa 125µs.

Als Codec ist der hochauflösende *L24*-Codec zu wählen, welcher eine Pulse-Code-Modulation mit einer Bittiefe von 24 Bit verwendet. Bei der Pulse-Code-Modulation wird eine analoges Signal, wie das einer Tonspur, in ein digitales Signal umgewandelt. [21] [23]

Alle weiteren Einstellungen können auf ihren Standardwerten belassen werden, wie in der folgenden Abbildung 9 zu sehen ist.

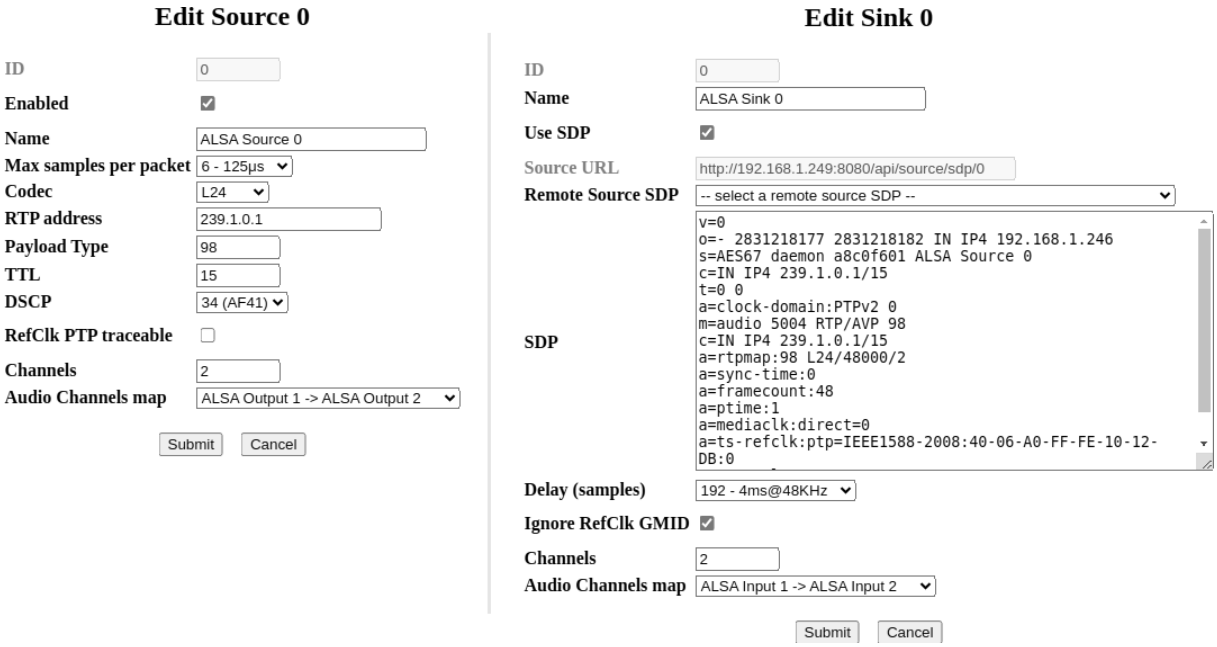

Abbildung 9: Test I - AES67 Einstellungen

Desweiteren ist für den BeagleBone, welcher das Signal empfangen soll, eine AES67 *"Sink"* in der Weboberfläche des Treiber einzurichten. Diese kann durch das Aktiveren der Einstellung *"Use SDP"* über die automatische Erkennung mit Hilfe von SDP unter dem Punkt *"Remote Source SDP"* gewählt werden. Als *"Delay (Samples)"* ist ein Wert von 192 Samples, welcher 4ms Verzögerung bei einer Abtastrate von 48kHz ergibt, zu wählen.

Die Anzahl der Audikanäle wird identisch zu der Anzahl der Kanäle bei der Quelle gesetzt, sodass ein Stereoeingang durch die virtuelle Soundkarte bereitgestellt wird.

Oszilloskop Einstellungen Beide Eingangskanäle des Oszilloskops werden auf eine Empfindlichkeit von $\pm 2V$ gestellt.

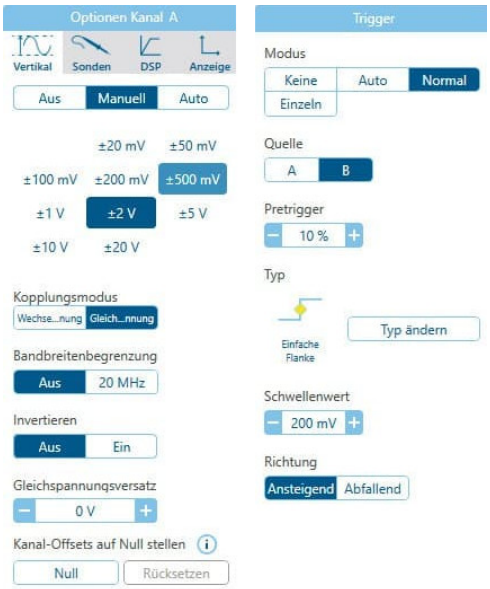

Abbildung 10: Test I Oszilloskop Einstellungen

Ein Trigger soll für eine steigende Flanke auf dem Kanal B mit einem Schwellenwert von +200mV erstellt werden, um so den Anfang des empfangen Signales auf dem Kanal B zu erfassen. Dabei soll ein Pretrigger-Wert von +10% genutzt werden, sodass der Anfang des Signales, welches über AES67 empfangen wurde, soweit links wie möglich in der Darstellung der Oszilloskop-Grafik zusehen ist.

Sender Der BeagleBone, welcher als Sender verwendet wird, soll durch die Verwendung der Befehle *nice -n -10 alsaloop -C plughw:C8CH -P plughw:RAVENNA* und *nice -n -10 alsaloop -C plughw:C8CH -P plughw:C8CH* das Signal des Funktionsgenerators an die lokale CTAG-Soundkarte und an die AES67-Soundkarte weiterleiten. Durch die Verwendung des Linux-Befehls *nice* mit dem Level *-10*, soll die Priorität der folgenden Befehle *alsaloop* angehoben werden. Dabei entspricht *-20* dem Echtzeit ähnlichstem Level.

Empfänger Der BeagleBone-Black, der das Audiosignal über AES67 empfängt, soll dieses mit dem Befehl: *nice -n -10 alsaloop -C plughw:RAVENNA -P plughw:C8CH* von der AES67-Soundkarte aufnehmen und über seine CTAG-Soundkarte abspielen.

5.4.2 Durchführung

Nachdem der Test nach der Versuchsbeschreibung aufgebaut und erstmalig durchgeführt wurde, stellte sich heraus, dass der BeagleBone-Black die vom Funktionsgenerator des Oszilloskops empfangen Daten nicht ohne Probleme auswerten und an zwei Audiogeräte

weiterleiteten konnte. Der Hierfür verwendete Befehl *nice -n -10 alsaloop -C plughw:C8CH -P plughw:RAVENNA* überlastete das System. Da aufgrund einer hohen Priorisierung, Routinen anderer Programme später und seltener ausgeführt werden können, geben diese dem Endanwender oft keine Rückmeldung, weshalb schnell der Eindruck entstehen kann, dass ein Programm oder der gesamte Computer überlastet sind. Um diesem Phänomen auszuschließen wurde der Befehl *alsaloop* ohne *nice* aufgerufen. Trotzdem geriet das System durch doppelte Verwendung des Befehls *alsaloop* erneut an seine Leistungsgrenzen und konnte nicht mehr auf Benutzereingaben reagieren, sodass es anschließend durch das Entfernen der Stromzufuhr neugestartet werden musste. Eine weitere Schwierigkeit in der Verwendung des Funktionsgenerator bestand darin, dass die erzeugte Sinus-Schwingung kontinuierlich war und ohne einen markanten Punkt, wie einen Anfangspunkt der Kurve, die Verschiebung der gemessen Audiosignale nicht erkennbar waren.

Audiodatei Anstatt die Welle des Funktionsgenerators zu verwenden, wurde daher eine Audiodatei mit der Audiosoftware *Audacity* erstellt, um diese anschließend auf dem sendenen BeagleBone-Black parallel über die CTAG-Soundkarte und die virtuelle AES67-Soundkarte abzuspielen.

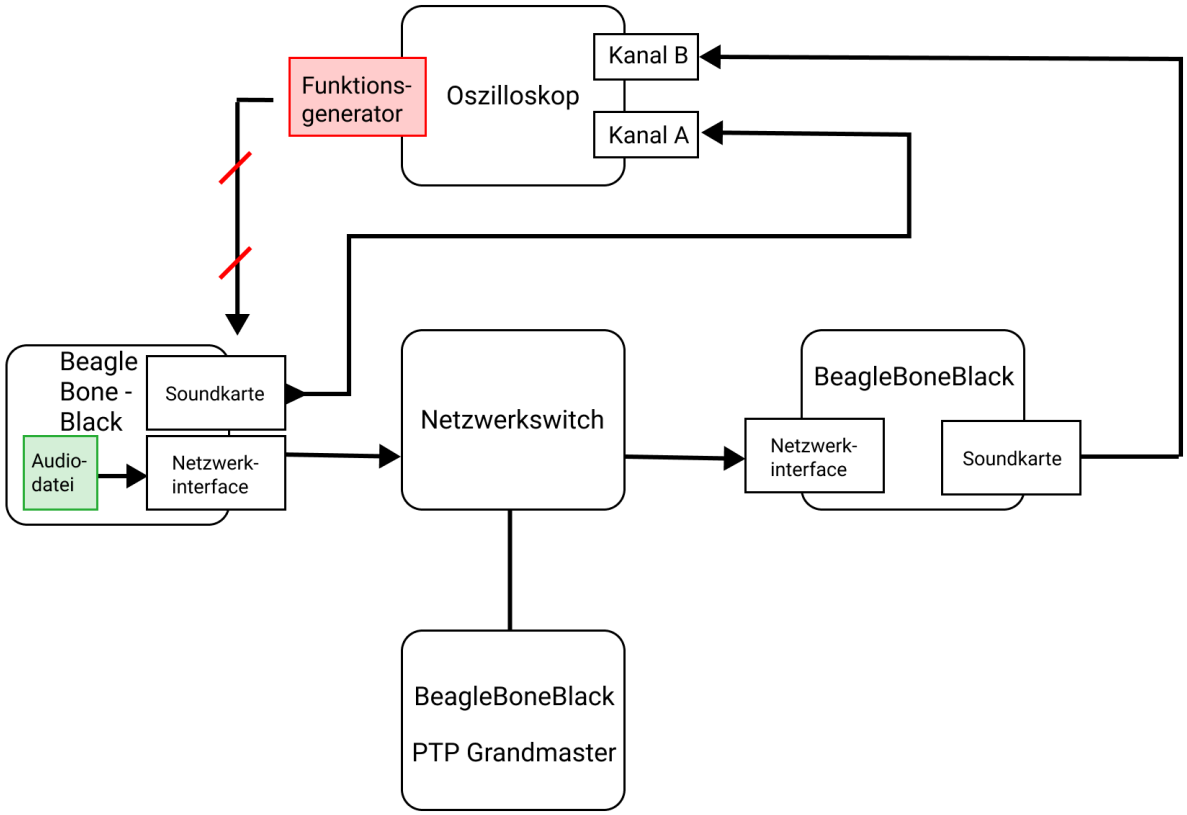

Abbildung 11: Test I Angepasster Versuchsaufbau - Point-to-point Übertragung mit Soundkarte

Die Audiodatei beginnt mit einer Pause (vgl. Abbildung 12), um vor der Flanke des Pegels bereits einige Zeit lang die Übertragung am Oszilloskop beoachten zu können.

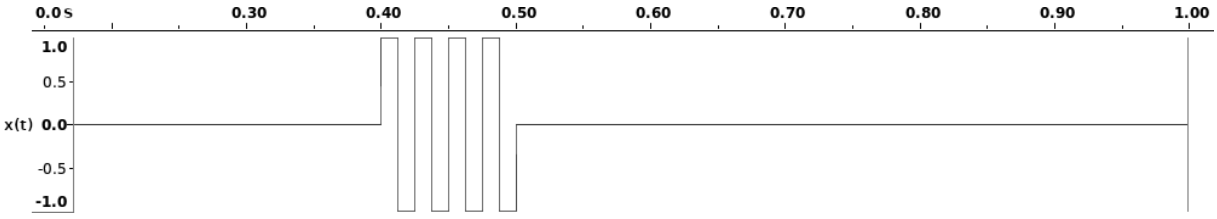

Abbildung 12: Tonspur der Audiodatei - 48kHz, Mono, 24bit

Nach 0,4 Sekunden startet dann ein Rechtecksignal, welches mit einer steigenden Flanke beginnt und anschließend, vier Ausschläge des maximalen Signalpegels im Wechsel mit vier Ausschlägen des minimalen Pegels durchläuft. Anschließend folgt erneut eine Pause bis die Audiodatei nach einer Sekunde endet.

ALSA *multi* Plugin Um parallel ein Audiosignal auf zwei Soundkarten auszugeben wurde ein virtuelles Ausgabegerät mit ALSA erstellt. Über die ALSA Erweiterung *multi* [18] lassen sich mit Hilfe der ALSA Konfigurationsdatei *~/.asoundrc* zwei Stereo-Ausgabegeräte in ALSA als ein einzelnes Ausgabegerät mit 4 Kanäle zusammenfassen. Durch die Verwendung des *route* Modules, kann wiederrum ein Stereogerät vor das neue Ausgabegerät geschaltet werden, welches seine zwei Kanäle auf die 4 Kanäle des neuen Ausgabegerätes routet. Hierbei können durch die Unterschiede der beiden verwendeten Soundkarten jedoch Probleme auftreten, da zum Beispiel die Synchronisierung der beiden Soundkarten nicht gewährleistet werden kann.

GNU Parallel Aufgrund der Probleme mit einem *ALSA multi* Ausgabegerät wurde die Software *parallel* des GNU Projektes verwendet [33]. Mit Hilfe des Tools *parallel* lassen sich mehrere Terminal-Befehle parallel auf verschieden CPU-Threads ausführen. So wurde für die parallele Wiedergabe der Audiodatei auf der lokalen CTAG-Soundkarte und der virtuellen AES67-Soundkarte zweimal eine Wiedergabe über die Software *aplay* mit dem *parallel* Tool gestartet: *echo -e "C8CH\nRAVENNA parallel –max-args=1 nice -n -10 aplay -D plughw:{1} single-spike.wav* Zusätzlich wird dabei mit *nice* die Priorisierung der beiden Wiedergaben angehoben.

Die dabei aufgezeichneten Tonspuren implizierten eine nicht erwartete Latenz der AES67-Übertragung von 1,638 Sekunden. Zusätzlich fehlte bei dem empfangen Signal eine fallende Flanke am Start der Übertragung.

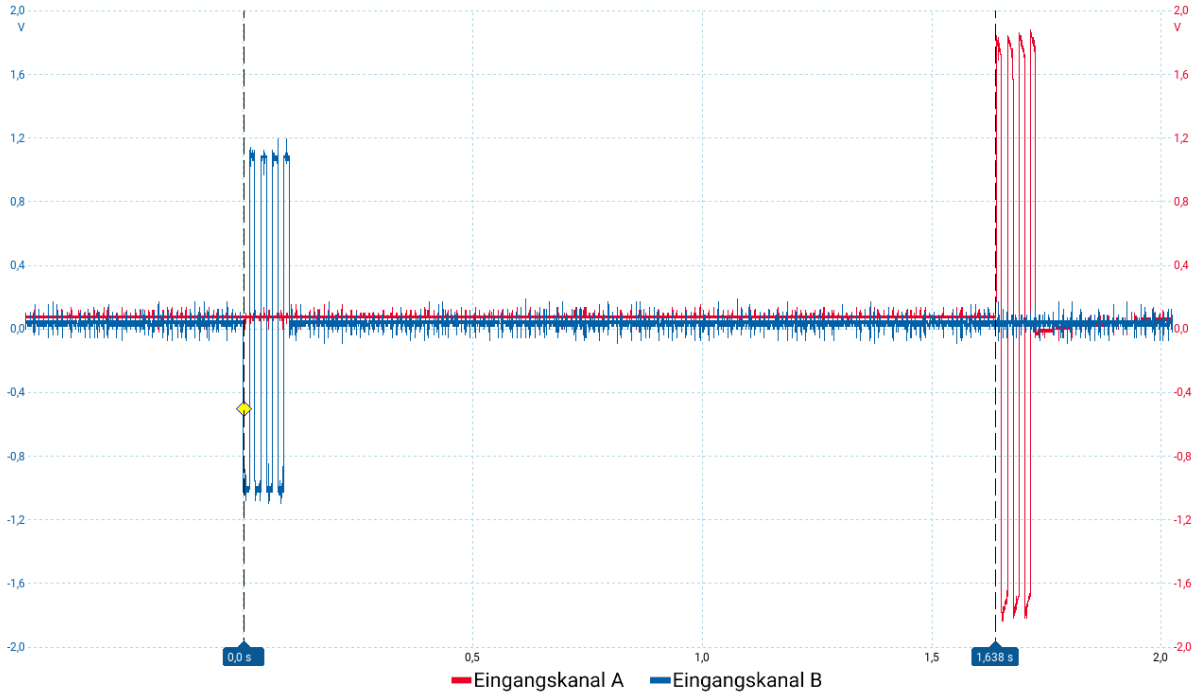

Abbildung 13: Test I - Messergebnis mit dem Tool *parallel*

Desweiteren wurde die Funktion des Tools *parallel* weiter untersucht, indem die Reihenfolge der beiden Soundkarten beim Aufruf des Befehls getauscht wurde: *echo -e "RAVENNA\nC8CH parallel –max-args=1 nice -n -10 aplay -D plughw:{1} single-spike.wav*

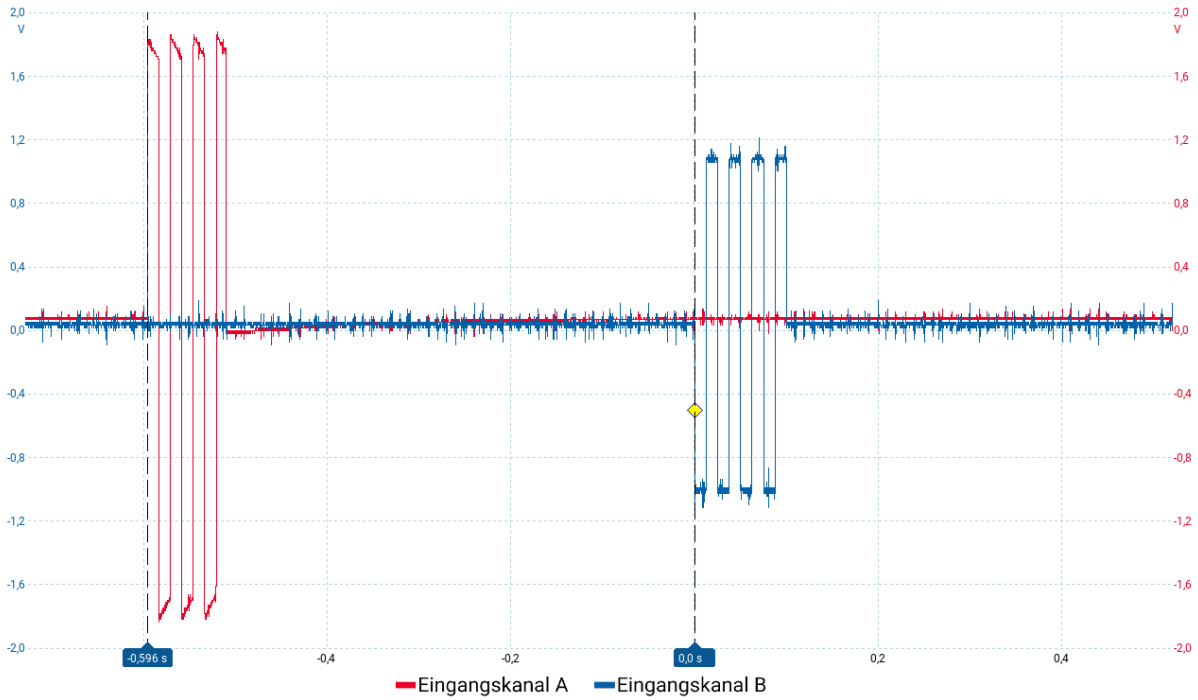

Abbildung 14: Test I - Messergebnis nach Tausch der Reihenfolge der Soundkarten mit dem Tool *parallel*

Das hierbei erhaltene Messergebnis (vgl. Abbildung 14) überrascht ebenso wie das erste Messergebnis (vgl. Abbildung 13). Beide Messungen zeigen eine große Zeitdifferenz der empfangen Daten. Das Rechtecksignal wurde zuerst durch den Eingangskanal B (rot) empfangen, bis es dann *0,596s* später auch von Kanal A (blau) aufgezeichnet wurde. Sollte die zweite Messung der Realität entsprechen, so hätte das Audiosignal zu erst empfangen werden müssen, bevor es abgeschickt wurde. Dies lässt auf einen Fehler schließen, welcher durch ein nicht zeitgleiches Abspielen über die beiden Soundkarten entstanden sein muss. Da der Tausch der Reihenfolge der Soundkarten beim Aufruf durch das *parallel* Tool zwei derartig unterschiedliche Ergebnisse liefert, muss dieses Tool intern den Aufruf der *aplay* Befehle stark versetzt ausführen oder die *aplay*-Befehle müssen sich gegenseitig blockieren.

Go Testprogramm Zum ausschließen von Fehlern durch das *parallel* Tool, wurde ein Programm mit der Sprache Golang entwickelt, welches eine Audiodatei lesen und den dabei gelesen Puffer an die beiden Soundkarten senden soll.

Das Programm verwendet für diese Aufgabe ein Go Module mit dem Namen *Portaudio* [17]. Diese Modul basiert auf der Portaudio Bibliothek, welche eine Abstraktionsschicht für die Nutzung von Audioeingabe- und Audioausgabegeräten auf verschieden Plattformen

bietet. Neben der Sprache Go musste somit über den Befehl *apt-get install portaudio19-dev* die Portaudio Bibliothek auf dem BeagleBone installiert werden.

Zunächst öffnet das Programm eine mono Audiodatei (siehe Listing 5 im Anhang). Als nächstes startet es über das *Portaudio* Modul zwei Ausgabestreams zu den beiden Soundkarten und beginnt schließlich mit dem Einlesen der Audiodatei. Während die Audiodatei in einen Puffer gelesen wird, wird dieser Puffer, sobald er gefüllt ist, kopiert und an die beiden Soundkarten gesendet. Somit kann an beide Soundkarten parallel die Tonspur übermittelt werden.

Überarbeitete Audiodatei Da für die Verwendung durch das Go-Programm die Audiodatei im Format *AIFF* mit einer Frequenz von *48kHz* und einer *24-bit* Samplingtiefe vorliegen musste, wurde die ursprüngliche Datei angepasst. Die Tonspur wurde zu dem anfänglichen Rechtecksignal um weitere Sinus- und Rechtecksignale mit unterschiedlichen Amplituden ergänzt.

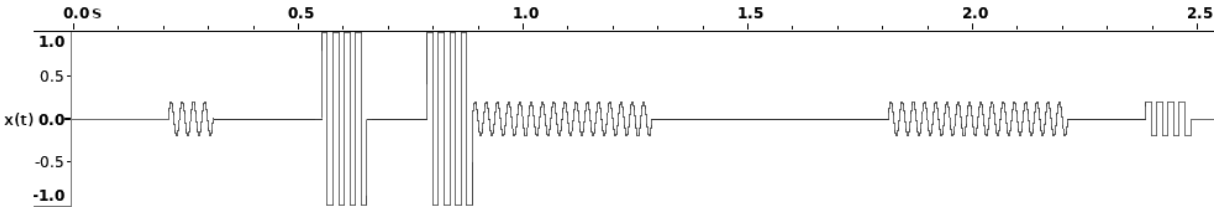

Abbildung 15: Tonspur des überarbeiteten Testsounds - 48kHz, Mono, 24bit

Zum Testen der neuen Datei wurde diese mit dem Befehl *aplay -D plughw:C8CH ./audio* auf der lokalen CTAG-Soundkarte abgespielt und über einen an der Soundkarte befestigten Tastkopf aufgezeichnet.

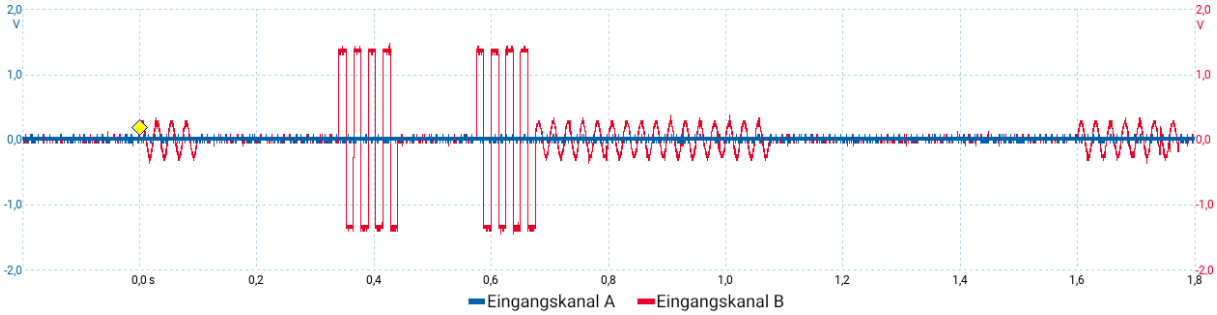

Abbildung 16: Messung des überarbeiteten Testsounds

5.4.3 Messergebnisse

Mit Hilfe des Go Programms und der überarbeiteten Audiodatei konnten folgende Messungen gemacht werden. Abbildung 17 zeigt einen Ausschnitt des Anfangs der aufgezeichneten Tonspuren und zwei Zeitpunkte. Der eine Zeitpunkt liegt dabei am Anfang des empfangenen Audiosignales zu Zeit des Triggers von *0ms* und ein zweiter an der identischen Stelle des gesendeten Audiosignales bei *-50,91ms*.

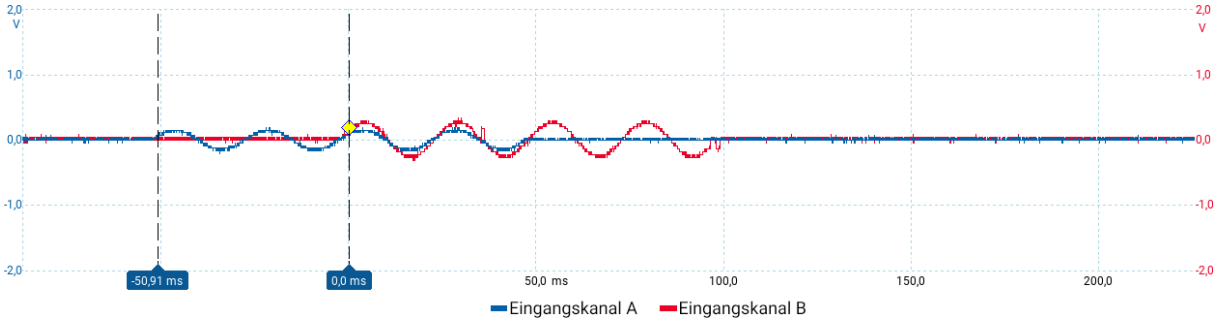

Abbildung 17: Test I Messung - Sinus

Auch an den Rechtecksignalen, welcher der Sinuswelle folgten, konnten zwei markante Zeitpunkte bei *289ms* und *339,7ms* markiert werden.

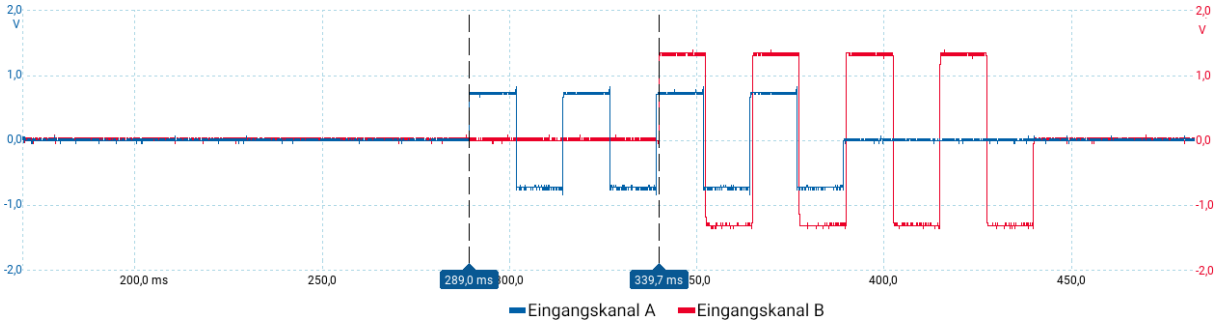

Abbildung 18: Test I Messung - Rechtecksignal

Die dritte Grafik zeigt die gesamte Messung der beiden Audiosignale beginnend vom Trigger des empfangen Signales bis zum Ende der Übertragung nach circa 2,5 Sekunden.

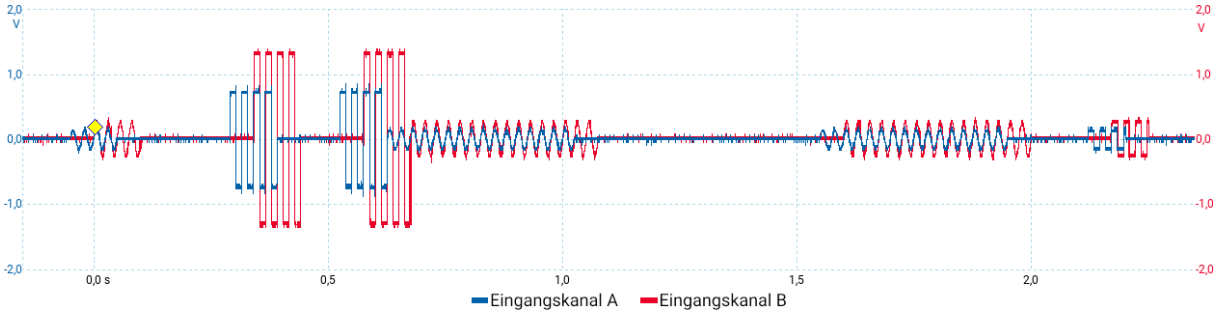

Abbildung 19: Test I Messergebnis nach Wechsel der parallel Aufrufe

5.4.4 Auswertung

Anhand der ersten Messung (Abbildung 17) lässt sich eine Latenz von *50,91 - 0ms = 50,91ms* bestimmen. Eine ähnliche Latenz ergibt auch die zweite Messung (Abbildung 18) mit einer Zeitdifferenz von *339,7ms - 289ms = 50,7ms* zwischen den aufgezeichneten Signalen. In der letzten Messung (Abbildung 19) wird die konstante Synchronisierung der beiden Tonspuren erkennbar, welche identisch zu den vorherigen Messungen eine konstante Latenz der empfangenen zur gesendet Spur zeigt.

5.4.5 Diskussion

Zeitverögerung des *parallel* Tools Die Verwendung eine speziell entwickelten Programmes schien die vorherigen Probleme mit dem *parallel* Tool vollständig umgangen zu haben. Neben der eventuellen Ungenauigkeit die das Tool *parallel* durch seine Programmierung und Art der Ausführung mitbringt, könnte ein möglicher Grund der starken zeitlichen Verzögerung auch der zeitgleiche Zugriff durch zwei *aplay* Wiedergaben auf eine Audiodatei sein. Möglicherweise blockiert hierbei der langsame Zugriff auf das Dateisystem die Ausführung des zweiten Aufrufs, sodass diese erst nach dem beendeten Lesezugriff des ersten *aplay* Befehls fortfahren kann. Genau diese Problematik könnte das Go-Programm gelöst haben, da es nur einen Lesezugriff nutzt, um die Daten für die Wiedergabe über die beiden Soundkarten zu laden.

Ursprung der Zeitdifferenz Da bei AES67 eine Latenz von $< 10ms$ verlangt wird, stellt sich bei einer gemessenen Zeitdifferenz von circa *50ms* die Frage, ob die gemesse Latenz ausschließen durch die AES67-Übertragung entstanden ist oder zusätzliche Komponenten des Versuchsaufbaus Einflüsse auf die Zeit genommen haben können.

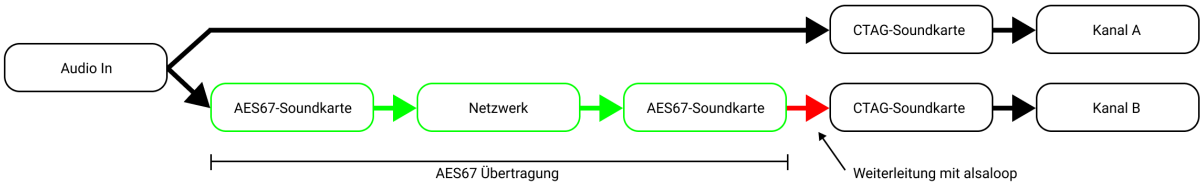

Abbildung 20: Test I - Weg des Audiosignales mit gekennzeichneten Abschnitten

AES67 fordert diese Latenzvorgabe von der reinen Ende-zu-Ende-Übertragung ohne Konvertierung oder analog-digital oder digital-analog Weiterverarbeitung des Signales (vgl. grüner Abschnitt Abbildung 20). Da zusätzlich zu der Übertragung das Signal durch weitere Programme, wie das Tool *alsaloop* (vgl. roter Abschnitt Abbildung 20), geleitet wurden, können somit zusätzliche Latenzen entstehen.

Synchronität des Signales Wie in Abbildung 19 zu sehen ist, wird das Signal während der gesamten Übertragung zeitlich synchron mit einer konstanten Latenz zu dem gesendeten Signal empfangen.

Überarbeitete Audiodatei Durch die überarbeitete Audiodatei konnte visuell gezeigt werden, mit welcher Verschiebung die Signale aufgezeichnet wurden. Zusätzlich wurden durch verschiede Funktionsarten auch verschiedene mögliche Pegel getestet. Der leichte Amplitudenunterschied der beiden Tonspuren lässt sich auf mögliche Unterschiede in der Lautstärkeeinstellung der Soundkarten zurückführen.

Verbesserungen Aufgrund des aufwendingen Versuchsaufbaus und die Limitierungen der Oszilloskop-Software, war es nicht möglich ohne einen extremen händischen Aufwand, eine reproduzierbare Messreihe zu erstellen. Für das automatische Messen wäre ein Programm hilfreich, welches markante Stellen auf beiden Tonspuren erkennen könnte, sodass ein Zeitintervall zwischen den Zeitpunkten der beiden Stellen ermittelt werden könnte.

5.5 Test II - Point-to-point Übertragung mit GPIO Ausgabe

Im Gegensatz zu der Ausgabe des Audio Signales soll im zweiten Test ein GPIO Pin auf logisch *An* gesetzt werden, sobald ein Audiosignal empfangen wurde. Hierdurch könnten mögliche Zeitverzögerungen, die von der Ausgabe über die Soundkarte entstehen, offen gelegt werden. Wie bereits im ersten Versuch soll durch das Oszilloskop die Latenz zwischen dem Senden und dem Empfang des Signales gemessen werden.

5.5.1 Versuchsaufbau

Anders als bei dem Versuchsaufbau des ersten Tests, soll im zweiten Test statt das Audiosignal zusätzlich an eine CTAG-Soundkarte zu schicken, ein GPIO Pin des BeagleBone-Blacks auf logisch *An* gesetzt werden. Als Audioquelle kann eine Datei verwendet werden, die möglichst zeitgleich beim Setzten des GPIO Pins an die virtuelle Soundkarte des AES67 Treibers gesendet werden soll.

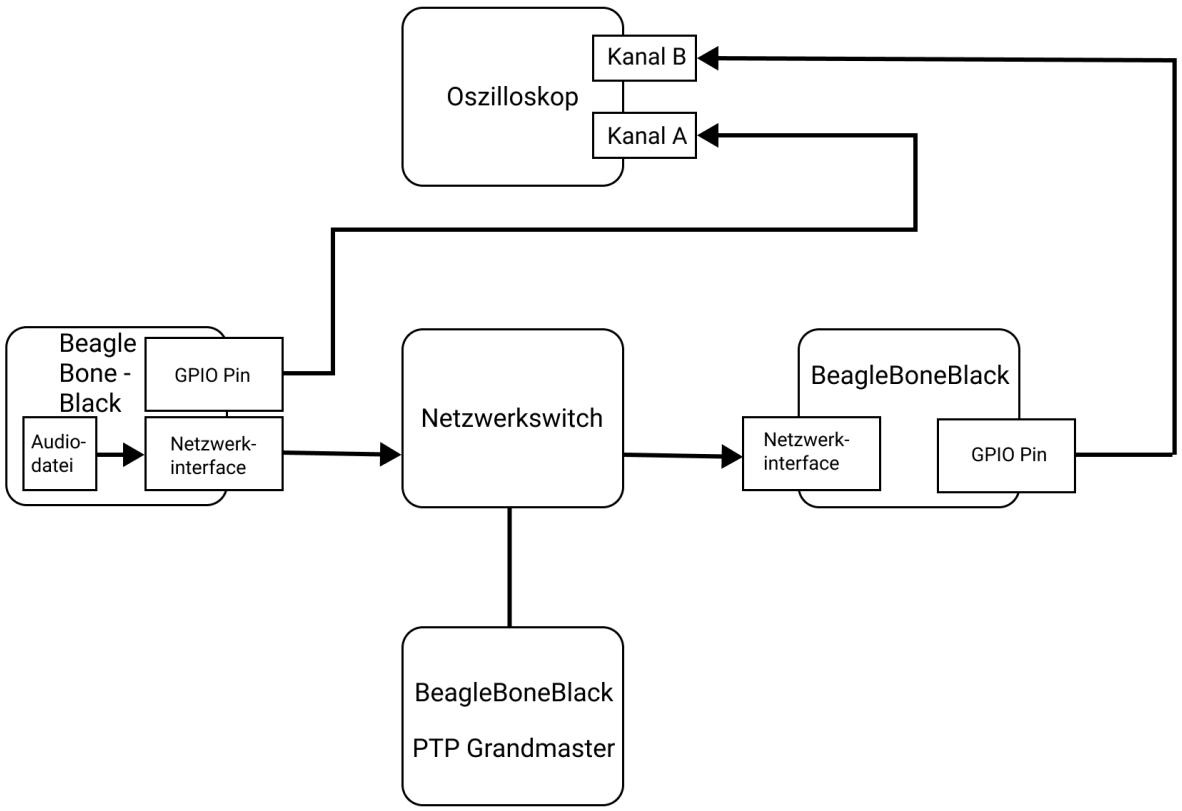

Abbildung 21: Test II Versuchsaufbau - Point-to-point Übertragung mit GPIO Ausgabe

Sobald das Signal am zweiten BeagleBone-Black über die AES67 Soundkarte empfangen wird, wie in Abbildung 21 zusehen ist, soll von diesem ebenfalls ein GPIO Pin auf logisch *An* gesetzt werden. Das Oszilloskop soll dabei die Pegel der beiden GPIO Pins aufzeichnen, sodass die Zeit zwischen den steigenden Flanken der beiden Pegel gemessen werden kann.

Oszilloskop Damit am Oszilloskop die Zeitabstände der beiden steigenden Flanken der GPIO Pegel automatisch gemessen werden können, müssen zunächst beide Eingangskanäle auf $\pm 5\ V$ gestellt werden. Anschließend wird ein Flanken-Trigger auf dem Kanal, welcher mit dem sendenen BeagleBone verbunden ist, eingerichtet. Dafür wird ein Schwellenwert von +1 V für eine ansteigende Flanke festgelegt, da trotz geringer Schwankungen der *Aus*-Pegel des GPIO Pins nie über diesen Wert kommt und die Spannung des *An*-Pegels der internen Spannung des BeagleBone-Blacks von 3,3 V entspricht.

Zusätzlich zu den beiden Eingangskanälen soll ein dritter Kanal eingerichtet werden. Dieser sogenannte Rechenkanal, kann das Ergebnis einer Rechnung zeigen, die für jeden Zeitpunkt den absoluten Wert aus der Differenz der Pegel der beiden Eingangskanäle berechnet.

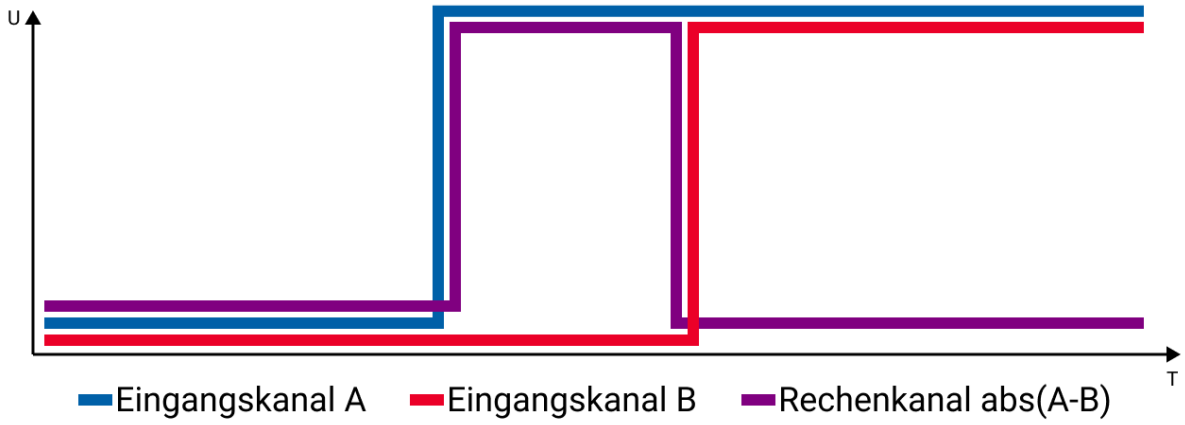

Abbildung 22: Test II - Schematische Darstellung des Rechenkanals abs(A-B)

So liegt der Pegel dieses Rechenkanals auf 0 V, solange beide Eingangskanäle die selbe Spannung haben. Sollte sich nun die Spannung einer der Eingangskanäle anheben, so würde auch die absolute Differenz der beiden Eingangskanäle diesen Wert annehmen, sodass eine steigende Flanke entsteht. Wechselt schließlich auch der zweite Eingangskanal den Pegel auf den Wert des anderen Eingangskanals so ist die Differenz der beiden wieder Null und der Rechenkanal zeigt eine fallende Flanke. Über die Software des Oszilloskop lässt sich dann die Breite des dabei entstehenden positiven Impuls des Rechenkanals automatisch messen. Diese gemessene Zeit entspricht dabei der Verzögerung, mit der das Audiosignal nach dem Senden empfangen wurde.

5.5.2 Durchführung

Sender Auf dem BeagleBone-Black, welcher das Audiosignal in das Netzwerk sendet (in der Abbildung 21 links zu sehen), wurde analog zu dem Vorgehen in Test I ein Go-Programm (siehe Listing 7 im Anhang) entwickelt. Dieses wurde dann verwendet, um

parallel zum Start des Audiostreams, einen GPIO Pin auf logisch *An* zu setzen. Als Tonspur wurde eine programmierte Sinus-Funktion genutzt.

Da bereits einige Pins des BeagleBone-Blacks durch ein Device-Tree Overlay der CTAG-Soundkarte belegt waren, musste nach einem freien GPIO Pin gesucht werden. Hierfür wurde der Pin 50 gewählt, welcher anschließend durch ein weiteres spezielles Overlay (siehe Listing 1 im Anhang) als GPIO Ein- und Ausgang gesetzt wurde. Dafür wurde das Overlay analog zu der Beschreibung im Kapitel 4.3 kompiliert und ebenfalls durch das Eintragen in die Datei */boot/uEnv.txt* nach einem Neustart aktiviert.

Empfänger Für das Endgerät, welches das Audiosignal empfangen soll, wurde ebenfalls ein Programm entwickelt (siehe Listing 6 im Anhang), um den Eingangs-Pegel der virtuellen Soundkarte des AES67 Treibers zu beobachten.

Das Go-Programm erhält dabei von *Portaudio* einen Eingangspegel mit dem Datentyp *float32* der virtuellen Soundkarte, der im Ruhezustand 0 ist. Sobald Audiodaten über AES67 empfangen und an die virtuelle Soundkarte übergeben werden, erhält das Programm Zahlenwerte, welche ungleich 0 sind, und kann in diesem Fall den GPIO Ausgang auf logisch *An* setzten. Nachdem das Go-Programm mit *go build -o aes-detector main.go utils.go* gebaut wurde, konnte es in einer Endlosschleife von der Konsole des BeagleBone-Blacks mit dem Befehl *while true; do nice -n -10 ./aes-detector; sleep 1; done* gestartet werden.

Die Verwendung des Kommandos *nice* erlaubt es dem Go-Programm mit erhöhter Priorität den Audioeingang zu überwachen und gegebenenfalls das Signal zum Schalten des GPIO Pins per serieller Schnittstelle schnellstmöglichst zu geben. Durch die umgebende *while*-Schleife um den Befehl, startet das Programm nachdem es ein Audiosignal erkannt und gemeldet hat, automatisch neu und setzt dabei den GPIO Port wieder auf logisch *Aus*, sodass eine neue Zeitmessung getätigt werden kann. *sleep 1* verhindert dabei, dass der GPIO Port zu schnell zurückgesetzt wird, sodass eine Messung durch das Oszilloskop problemlos durchgeführt werden kann.

5.5.3 Messergebnisse

An Hand der Abbildung 23 lässt sich eine Zeit von *78,37ms* für die *Hohe Impulsbreite* des *abs(A-B)*-Kanals ablesen.

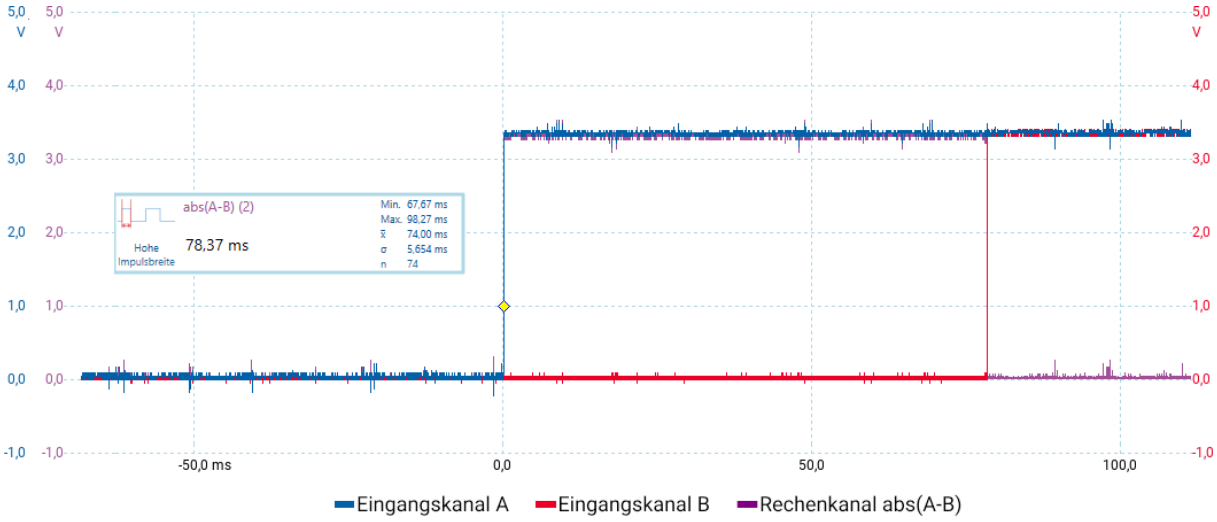

Abbildung 23: Test II - Messung der Impulsbreite von abs(A-B)

Da die Messung größtenteils automatisch ablief, wurde eine Messreihe von 75 Werten erzeugt.

lfd. Nr.	Latenz in ms	lfd. Nr.	Latenz in ms	lfd. Nr.	Latenz in ms
1	71,55	26	70,33	51	71,81
2	78,01	27	73,49	52	77,26
3	70,42	28	71,31	53	71,93
4	71,93	29	95,62	54	71,83
5	72,50	30	72,46	55	98,27
6	71,61	31	72,27	56	72,23
7	71,79	32	69,71	57	74,28
8	71,62	33	71,73	58	71,62
9	76,68	34	71,55	59	72,10
10	78,21	35	75,68	60	76,07
11	70,87	36	76,47	61	70,96
12	72,53	37	72,29	62	92,37
13	70,03	38	72,46	63	78,31
14	73,56	39	71,60	64	77,06
15	71,95	40	71,66	65	71,25
16	71,01	41	79,40	66	70,29
17	71,50	42	76,60	67	70,73
18	71,98	43	71,01	68	70,83
19	72,42	44	71,47	69	67,67
20	70,63	45	92,80	70	76,84
21	71,02	46	70,68	71	78,65
22	77,69	47	70,84	72	70,48
23	77,77	48	72,50	73	71,75
24	71,24	49	71,47	74	71,21
25	70,66	50	73,34	75	78,37

Tabelle 1: Test II Messerreihe - Latenz der Übertragung in ms

5.5.4 Auswertung

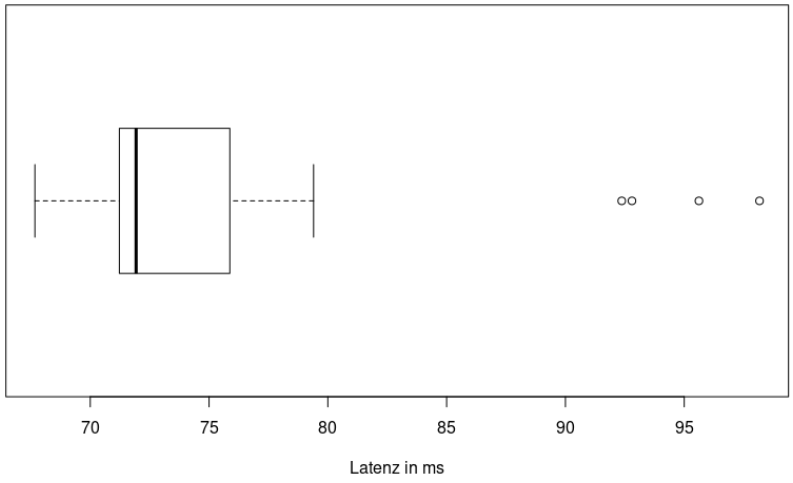

Abbildung 24: Test II Box-Plot - Latenz der Übertragung in ms

Die Box-Plot Grafik zeigt, dass ein Schwerpunkt der Messungen zwischen *71ms* und *76ms* liegt. Die Antenne deckt eine Bereich von circa *67* bis *80ms* ab. Zudem sind vier Ausreißer bei circa *95ms* zu erkennen.

	n	Median	σ	Min	Max
Latenz (ms)	75	71,93	5,65	67,67	98,27

Tabelle 2: Test II Auswertung - Latenz der Übertragung in ms

Der Median der 75 Messungen liegt bei *71,93ms*. Mit einer Standardabweichung von *5,65ms* sind ein Großteil der Werte auf einen verhältnismäßig kleineren Bereich verteilt. Der minimale gemessene Wert liegt bei *67,67ms*, während der maximale Werte mit *98,27ms* eine große Abweichung zu dem Median aufweist und somit als Ausreißer im Box-Plot zu erkennen ist.

5.5.5 Diskussion

Wie schon bei dem ersten Test, wurde auch im zweiten Test eine Latenz gemessen. Über die gesamten Messungen hinweg lag diese mit circa *71ms* über dem Wert des ersten Tests von circa *50ms*. Dies könnte möglicherweise auf das Go-Programm zurück zu führen sein, welches zum Erkennen des Audiosignales verwendet wurde. Obwohl bei der Entwicklung möglichst kleine Puffer gewählt wurden, werden möglicherweise Intern in der verwendeten *Portaudio* Bibliothek weitere Puffer eingesetzt. Da *Portaudio* als Open-Source Software zur Verfügung steht könnte hierbei ein Einlesen in den Quelltext interessante Aufschlüsse

liefern.

Auch durch die automatische Erkennung eines Audiopegels könnten weitere Messungenauigkeiten entstehen. Das zeitgleiche Setzten eines GPIO Pins ist immer mit einem geringen Zeitaufwand versehen. Dieser sollte allerdings durch das direkte Verwenden von Registern zum Setzten des Pins verhältnismäßig gering ausfallen und somit vernachlässigbar sein.

Ein interessanter Punkt in Verbindung mit dem Setzen des GPIO Pins ist dabei bereits in der Entwicklung des Go-Programmes aufgefallen. Sollte eine Ausgabe auf das Terminal zu Testzwecken des Programmes vor dem Setzten des GPIO Pins stattfinden, so ist eine erheblicher Latenzzuwachs zu verzeichnen.

Grundsätzlich hatte die Verwendung eines GPIO-Pegels den Vorteil, dass durch diese ein digitaler Pegel in Kombination mit einem Rechenkanals automatische Messung der *Hohen Impulsbreite* erlaubte. Diese Funktion war bei der Durchführung von großem Vorteil und ermöglichte dadurch eine sinnvolle Erstellung einer größeren Messreihe mit 75 Messungen.

Betrachtet man die gemessenen Werte ohne den nahezu konstanten Versatz von circa *70ms*, so erhält man eine Sammlung von Werten mit einer relativ geringen zeitlichen Streuung bei einer Standardabweichung von *5,65ms*. Besonders unter Beachtung der vielen Faktoren, durch die die Übertragung in einem derartig komplexen Versuchsaufbau beeinflusst werden kann, lässt diese Abweichung auf ein durchaus stabiles System schließen.

Die vier Ausreißer der Messreihe können auf mögliche äußere Einflüsse zurückgeführt werden. So wäre denkbar, dass durch andere Programme die echtzeitähnliche Ausführung der Testprogramme nicht gewährleistet werden konnte. Auch könnten das verwendete Netzwerk, welches zusätzlich zu den Paketen für die AES67-Übertragung andere Daten überträgt, für die Ausreißer verantwortlich sein.

Durch weitere Untersuchungen könnten möglicherweise die Ursachen der relativ konstanten Verzögerung von circa *70ms* weiter eingegrenzt werden. Hierfür könnten unter anderem die AES67 Datenpakete im Netzwerk mitgeschnitten werden und bei dem Empfang eines Paketes ein weiterer GPIO Pin gesetzt werden, sodass die Latenz vom Empfang bis zum Wandel des Paketes als Audiosignal über die ALSA Soundkarte, gemessen werden könnte.

5.6 Test III - Multicast Übertragung

Das parallele Senden von Audiodaten an zwei oder mehr Empfänger über Multicast sollte im dritten Test weiter untersucht werden. Hierfür sollte die absolute Zeitdifferenz der beiden steigenden Flanken der GPIO Ausgänge gemessen werden. Die GPIO Ausgänge sollten äquivalent zum Vorgehen vom zweiten Test genau dann von einem BeagleBone-Black auf *An* gesetzt werden, wenn dieser das gesendete Audiosignal empfangen hat.

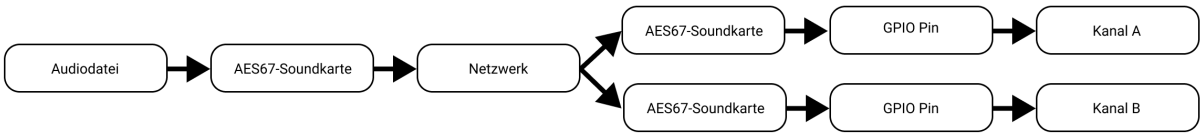

Abbildung 25: Test III - Weg des Audiosignales mit zwei Empfängern

Durch die Messung der theoretisch identischen Wege über das Netzwerk soll dabei untersucht werden, wie synchron die Endgeräte das Audiosignal erhalten.

5.6.1 Versuchsaufbau

Der Versuchsaufbau entspricht dem des zweiten Tests, jedoch soll ein weiterer BeagleBone-Black an das Netzwerk als Empfänger angeschlossen werden und identisch zu dem anderen empfangenden BeagleBone eingerichtet werden, sodass die selbe Übertragung wie im zweiten Versuch mit zwei statt einem Empfänger durchgeführt wird.

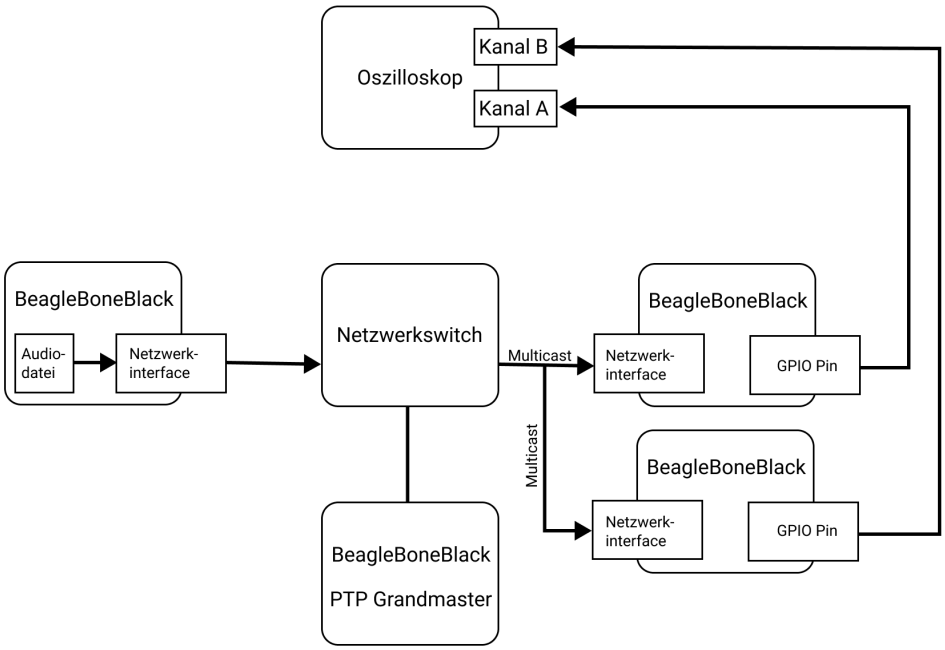

Abbildung 26: Test III Versuchsaufbau - Multicast Übertragung

Das Oszilloskop ist hierfür gleich zum zweiten Test einzustellen, jedoch soll der Kanal A, welcher zuvor das Audiosignal am sendenen BeagleBone abgegriffen hat, in diesem

Aufbau an den zweiten empfangenen BeagleBone angeschlossen werden. Somit können über beide Kanäle die Tonspuren der beiden Empfänger aufgezeichnet werden.

Sender Für den Sender soll der Befehl *aplay -D plughw:RAVENNA ./audio* zum Abspielen der überarbeiteten Audiodatei des ersten Tests verwendet werden.

Empfänger Die beiden Empfänger sollen analog zu Test II mit dem Programm zum Erkennen eines Audiosignales (vgl. Listing 6 im Anhang) ausgestattet werden. Über den Befehl *while true; do nice -n -10 ./aes-detector; sleep 1; done* soll anschließend das Test-Programm gestartet werden, sodass wiederholt zum Erstellen einer Messreihe Anfänge der Audiosignale erkannt werden können.

5.6.2 Durchführung

Der Aufbau des zweiten Tests wurde durch einen BeagleBone-Black erweitert, welcher identisch zu dem existierenden Empfänger konfiguriert und über Kanal A an das Oszilloskop angeschlossen wurde. Daraufhin wurde die Messung am Sender über den Aufruf des Befehls *aplay* gestartet und mehrfach wiederholt.

5.6.3 Messergebnisse

Abbildung 27 zeigt stellvertretend für die gesamte Messreihe einen möglichen Graphen der Zeitdifferenz der Empfangspunkte der Audiosignale, welche von den Empfängern ausgegeben wurden.

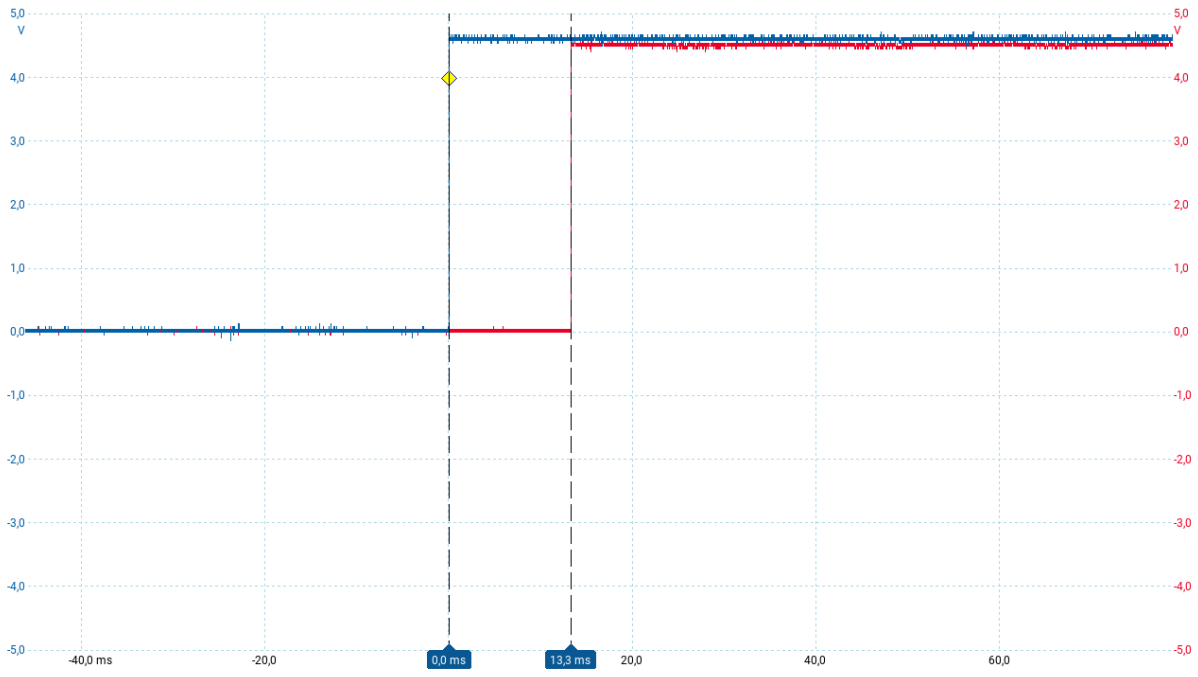

Abbildung 27: Test III - Messung der Zeitdifferenz beim Eintreffen der Audiosignale

Hierbei lassen sich zwei steigende Flanken festellen. Für den Eingangskanal A (blau) liegt die Flanke passend zu der Einstellung des Triggers bei *0ms* und für den Eingangskanal B (rot) bei *13,3ms*.

Zusätzlich wurde eine Messreihe mit 20 Messungen für den Test erstellt, in dem mit Hilfe des Befehls *aplay* eine Ausgabe wiederholend gestartet wurde.

lfd. Test-Nr.	Zeitdifferenz in ms
1	13,30
2	19,93
3	8,39
4	4,03
5	4,35
6	1,62
7	15,03
8	23,77
9	3,22
10	4,36
11	6,35
12	1,68
13	2,98
14	1,78
15	3,37
16	2,53
17	25,83
18	20,55
19	3,97
20	7,60

Tabelle 3: Test III Messreihe - Zeitdifferenz der Empfangspunkte in ms

5.6.4 Auswertung

Mit Hilfe eines Box-Plots wurde die Messreihe weiter analysiert, um somit die Verteilung der Messungen zu veranschaulichen.

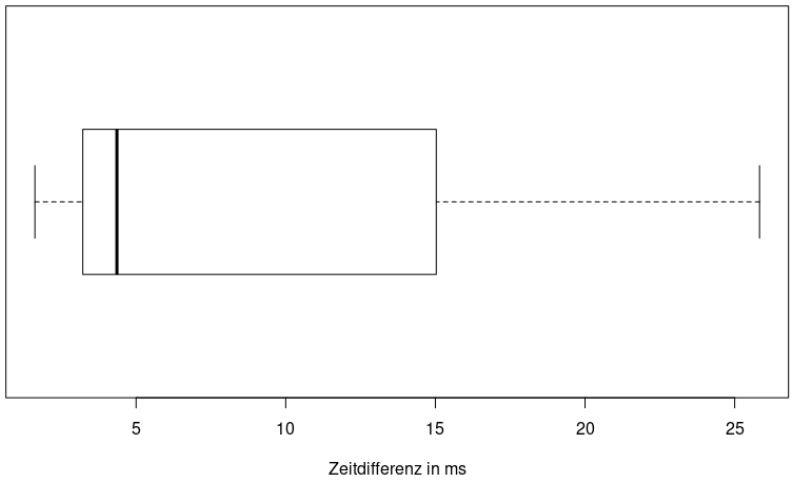

Abbildung 28: Test III Box-Plot - Zeitdifferenz der Empfangspunkte in ms

Die Whisker der Box-Plot Grafik stellen einen Bereich von circa *1* bis *26ms* dar, welcher sämtliche Werte der Messung umfasst. Somit existieren keine besonderen Ausreißer im Box-Plot. Während das untere Quantil zum Median mit einer Breite von einer Millisekunde gering ausfällt, beschreibt das obere Quantil eine Streuung von circa *10ms* und fällt somit wesentlich größer aus.

	n	Median	σ	Min	Max
Absolute Zeitdifferenz (ms)	20	4,36	8,08	1,62	25,83

Tabelle 4: Test III Auswertung - Absolute Zeitdifferenz (ms) zwischen Empfang durch BeagleBoneBone 1 und 2

Der Median der 20 Messungen liegt bei *4,36ms*. Mit einer Standardabweichung von *8,08ms* verteilen sich die Werte auf auf den circa halben Bereich, welcher von den beiden extremsten Werten der Messung abgegrenzt wird. Der minimale gemessene Wert liegt bei *1,62ms* und ist somit circa *2ms* vom Median entfernt. Der maximale Werte hingegen liegt bei *25,83ms* und ist mit circa *22ms* wesentlich größer als der Median.

5.6.5 Diskussion

Das theoretisch beste denkbare Ergebnis für die Zeitdifferenz des Empfangs des Audiosignales durch die beiden BeagleBones liegt bei einem Wert von *0ms*, da in diesem Fall die beiden Endgeräte das Signal zu genau dem selben Zeitpunkt empfangen hätten. Der Median von *4,36ms* liegt dennoch im Akzeptanzbereich von unter 10 Millisekunden, da bei einer Vorgabe dieser maximalen Latenz, der maximale Latenzunterschied bei genau 10 Millisekunden liegen würde. So könnte theoretisch ein Gerät das Signal ohne Latenz erhalten, während ein anderes Gerät das Signal erst mit einer Latenz von circa 10 Millisekunden empfängt und dennoch hätten beide Geräte diese AES67 Anforderungen erfüllt.

Eine mögliche Erklärung für die Standardabweichung von *8ms* könnte sich auf die doppelten Messungenauigkeiten durch das Verwenden von zwei Empfängern zurückführen lassen. So kann nicht nur bei dem ersten Empfänger durch zum Beispiel mögliche Unterbrechungen des Linux Kernels, die Verarbeitung des Audiosignales unterbrochen werden, sondern eine ähnliches Verzögerung kann ebenfalls bei dem zweiten Empfänger auftreten.

Der Test könnte fortgeführt werden, indem eine weitere Messreihe mit einer großen Anzahl an Messungen durchgeführt wird, um die Verteilung auf Streuung und Ausreißer weiter zu untersuchen. Zusätzlich könnten weitere Variationen des Tests durchgeführt werden. So könnte zum Beispiel die Audiospur der beiden Empfänger verglichen werden. Ein anderer Test könnte zu den beiden GPIO Ausgängen der Empfänger ebenfalls einen Ausgang des Senders aufgezeichnen, sodass neben der Synchronität der Empfänger auch die Latenz zum Sender bewertet werden kann. Durch das künstliche Generieren von einer großen Last für das Netzwerk, könnte ebenfalls ein Latenz und Synchronitäts Test durchgeführt werden.

6 Fazit

6.1 Zusammenfassung

Im Rahmen dieser Arbeit wurde zunächst ein Media-IP-Echtzeitnetzwerk konzipiert und nach der erfolgreichen Inbetriebnahme auf verschiedene Aspekte untersucht. Da das Medianetzwerk an erster Stelle Audio übertragen sollte, wurde hierfür an Hand einiger Grundlagen der Audio-Standard AES67 und das RAVENNA Protokoll beschrieben. Weitere Netzwerk-Protokolle, wie das für die Zeit-Synchronisierung mit hoher Präzision verwendete Protokoll PTP, wurden ebenfalls vorgestellt.

Nachdem die für eine Audioübertragung mit AES67 notwendigen Komponenten ausgearbeitet waren, wurde ein minimales Netzwerk mit den dafür relevanten Geräten konzipiert und aufgebaut. Hierzu wurden unter anderem ein PTP-Master für die Zeit-Synchronisierung, ein Motu Netzwerkswitch mit Paketpriorisierung für Audiodaten und mehrere Endgeräte eingesetzt. Neben einem Desktop-Computer wurden dabei insbesondere mehrere Einplatinencomputer mit dem Namen BeagleBone-Black verwendet. Um eine Audioausgabe über diese BeagleBones zu ermöglichen wurde die spezielle CTAG-Soundkarte benutzt, welche sich direkt auf die Einplatinencomputer aufstecken lässt, sodass diese über ihre GPIO Pins mit der Soundkarte kommunizieren können.

Für den Betrieb der CTAG-Soundkarte musste ein Treiber in den Kernel des Linux Betriebssystems für die BeagleBone-Blacks eingebaut werden. Anschließend wurde dieser zusammen mit einer Beschreibung für die zu benutzenden Pins der Soundkarte in ein Image für die BeagleBones integriert. Dieses Image, welches auf einem Debian 10 mit einem Linux Kernel in der Version 4.19 samt Realtime-Patch basiert, wurde zusätzlich um den ALSA-Treiber für eine virtuelle AES67-Soundkarte auf Basis eines RAVENNA Treibers erweitert.

Zuletzt wurde in drei unterschiedlichen Tests die Latenz und Synchronität einer AES67-Übertragung evaluiert. Dabei wurde mit Hilfe eines Oszilloskops die Latenz zwischen dem Senden und Empfangen eines Audiosignales untersucht. Zudem wurde mit zwei Endgeräten der gleichzeitige Erhalt eines gesendeten Signales überprüft.

6.2 Herausforderungen

Das Bauen und Anpassen eines Linux-Kernels wurde für die Vorbereitungen des Aufbaus eines Audionetzwerkes ausführlich behandelt und auch das Einarbeiten in das Testen mit einem Oszilloskop und die damit verbunden elektrotechnischen Aspekte wurden im Laufe dieser Arbeit erlernt.

Besonders die vielen Schichten, welche die Konzeptionierung eines Audionetzwerkes mit sich brachte, erforderten trotz spannender Teilgebiete, eine stark fokussierte Arbeit, um ein funktionierendes Netzwerk in der Zeit von drei Monaten nach dem zuvor erstellten Konzept zu betreiben.

Die Komplexität der vielen verwendeten Hardware- und Softwarekomponenten erschwerte teilweise das Eingrenzen möglicher Fehlerquellen beim Aufbau und Testen des Netzwerkes.

Um eine Zusammenarbeit unterschiedlichster Komponenten zu ermöglichen, mussten diese oft angepasst oder aufgrund von ausschließenden Abhängigkeiten getauscht werden.

So wurde zum Beispiel das Cross-Compiling eingesetzt, um die zeitaufwändigen Kompilierungsprozesse auf leistungsstarker Hardware durchzuführen. Hierbei wurde ein Fehler aufgedeckt, welcher fälschlicherweise Teile der Hostplattform in die Kompilate für das Zielsysteme mit einbindet, sodass ein weiteres Arbeiten mit den Kompilaten nicht uneingeschränkt möglich war.

6.3 Lessons Learned

Durch das Arbeiten mit ausschließlich kleinen Variationen an Versuchsaufbauten oder an Softwareteilen wie dem Linux-Kernel, konnten nicht funktioniernde Anpassungen direkt erkannt und alternativ gelöst werden.

Um oft auftretende Anpassungen schnellstmöglich durchführen zu können, war der Einsatz von Automatisierungen und Optimierungen hilfreich. Hierfür wurden zum Beispiel mehrere Skripte für das automatische Bauen eines SD-Karten Images für den BeagleBone erstellt, um so kleine Anpassungen am Linux-Kernel so schnell wie möglich zu testen. Das Verwenden einer virtuellen Maschine, welche äquivalent zu dem BeagleBone eingerichtet wird, könnte hierbei weitere Zeitersparnisse einbringen.

Obwohl die durchgeführten Tests ausführlich und gründlich geplant wurden, stellten sich einige Ideen als nicht zielführend heraus. So wurde unter anderem das Tool *GNU parallel* ausgetauscht, da dieses den Anforderungen der Tests nicht genügte. Dieser Wechsel

brachte zusätzliche positive Nebeneffekte, welche weitere Problematiken lösten.

Das Betrachten einiger Schwierigkeiten von neuen Perspektiven, deckte die Fehleinschätzung von einigen Fehlerquellen auf. So konnte nach langem Testen und Probieren verschiedenster U-Boot Optionen festgestellt werden, dass unregelmäßig fehlschalgende Bootvorgänge eines BeagleBones nicht durch Software- oder Anwenderfehlern ausgelöst wurden, sondern aus einer schlechten Verbindung der SD-Karten Kontakte resultierten. Dieses Problem lies sich schließlich durch leichtes Nachjustieren der Taster des SD-Karten-Slots beheben.

6.4 Ausblick und Empfehlungen

Die Tests dieser Arbeit ließen bereits erste Eindrücke auf die Latenz und Synchronität des konzipierten Netzwerkes zu. Zusätzliche Variationen der Tests könnten die Leistung des Systems weiter untersuchen.

Hierfür wäre eine Test, welcher zusätzlich zu den den gemessenen GPIO-Pegeln der beiden Empfänger des dritten Tests parallel die gesendeten und empfangenen Audiospuren aufzeichnet, sinnvoll.

Zum Evaluieren des Systems unter Last, könnte ein Test durchgeführt werden während zwei Geräte im Netzwerk große Datenmengen austauschen.

Durch Automatisierungen lassen sich umfangreiche Messreihen erstellen, auf Basis welcher besonders aussagekräftige Ergebnisse erzielt werden können.

Der Aufbau könnte des weiteren durch Endgeräte unterschiedlichster Hersteller ergänzt werden, um so die Interoperabilität mit dem vorhandenen System zu untersuchen. Besonders die Verwendung genormter Geräte, für welche zum Beispiel die Latenz in einem Datenblatt festgehalten wurde, könnte bei dem Testen weitere unbekannte Faktoren ausschließen.

Sollte die Zusammenarbeit weiterer Geräte mit dem konzipierten System erfolgreich getestet worden sein, so sollte dem Einsatz in einem produktiv System nichts entgegenstehen.

Literaturverzeichnis

[1] *Advanced Linux Sound Architecture (ALSA) project*. URL: https://www.alsa-project.org/wiki/Main_Page (besucht am 03. 07. 2021).

[2] Paul Albitz, Cricket Liu und Paul Albitz. *DNS und Bind: für Systemadministratoren*. 2., erweiterte und aktualisierte Aufl., 2., korrigierter Nachdruck. Beijing Köln: O'Reilly, 2001. 491 S. ISBN: 978-3-89721-160-5.

[3] Inc. Audio Engineering Society. *AES standard for audio applications of networks -High-performance streamingaudio-over-IP interoperability*. 2013.

[4] Andrea Bondavalli. *Adjusted ALSA RAVENNA/AES67 Driver*. URL: https://github.com/bondagit/aes67-linux-daemon (besucht am 04. 07. 2021).

[5] Andrea Bondavalli. *AES67 Linux Daemon*. URL: https://github.com/bondagit/ravenna-alsa-lkm/tree/aes67-daemon (besucht am 07. 07. 2021).

[6] Daniel P. Bovet. *Understanding the Linux Kernel*. O'Reilly Media, 17. Nov. 2005. 944 S. ISBN: 9780596554910.

[7] Cisco Corporation. *Handbuch Netzwerktechnologien [komplettes Gundwissen zu Networking und Internetworking*. München: Markt-und-Technik-Verl, 2001. ISBN: 3-8272-6080-9.

[8] Intel Corporation. *Intel Ethernet Controller i210 Datasheet*. Version 3.7. URL: http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/i210-ethernet-controller-datasheet.pdf (besucht am 28. 06. 2021).

[9] Roy Thomas Fielding. „Architectural Styles and the Design of Network-based Software Architectures". 2000. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation_2up.pdf (besucht am 03. 07. 2021).

[10] Free Software Foundation. *GCC, the GNU Compiler Collection*. URL: https://gcc.gnu.org/ (besucht am 06. 07. 2021).

[11] Thomas Gleixner u. a. *The Real-Time Linux (RTL) Collaborative Project*. 2021. URL: https://wiki.linuxfoundation.org/realtime/start (besucht am 06. 07. 2021).

[12] Christoph H. Hochstätter. „Android-Architektur: Wieviel Linux steckt in Googles OS?" In: *zdnet.de* (18. Mai 2011). URL: https://www.zdnet.de/41553061/android-architektur-wieviel-linux-steckt-in-googles-os/ (besucht am 01. 07. 2021).

[13] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*. DOI: 10.1109/IEEESTD.2008.4579760.

[14] *IEEE Standard for Local and metropolitan area networks–Audio Video Bridging (AVB) Systems*. 2008. DOI: 10.1109/ieeestd.2011.6032690.

[15] Docker Inc. *What is a Container?* URL: https://www.docker.com/resources/what-container (besucht am 10. 07. 2021).

[16] Github Inc. *Github Facts*. URL: https://github.com/about (besucht am 11. 07. 2021).

[17] Gordon Klaus. *Go bindings for the PortAudio audio I/O library*. 2021. URL: https://github.com/gordonklaus/portaudio (besucht am 09. 07. 2021).

[18] Jaroslav Kysela u. a. *ALSA - PCM (digital audio) plugins*. URL: https://www.alsa-project.org/alsa-doc/alsa-lib/pcm_plugins.html (besucht am 30. 06. 2021).

[19] Robert Love. *Linux Kernel Development*. ADDISON WESLEY PUB CO INC, 1. Juni 2010. 440 S. ISBN: 0-672-32946-8.

[20] Dipl.-Ing. Stefan Luber und Dipl.-Ing. Andreas Donner. „Was ist eine Fritzbox?“ In: *ip-insider.de* (20. Nov. 2019). URL: https://www.ip-insider.de/was-ist-eine-fritzbox-a-883753/ (besucht am 01. 07. 2021).

[21] Jens-Rainer Ohm; Hans Dieter Lüke. *Signalübertragung Grundlagen der digitalen und analogen Nachrichtenübertragungssysteme*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. ISBN: 9783642102004.

[22] Stefan Neufeldt. *ALSA audio-driver for AES67-Streaming*. URL: https://github.com/neufst/ALSA-AES67-Streaming (besucht am 05. 06. 2021).

[23] Rudolf Nocker. *Digitale Kommunikationssysteme 1*. Vieweg+Teubner Verlag, März 2013. 240 S. ISBN: 9783322802538.

[24] *OpenWrt Project*. URL: https://openwrt.org/ (besucht am 01. 07. 2021).

[25] The Linux PTP Project. *The Linux PTP Project*. 21. Juli 2021. URL: http://linuxptp.sourceforge.net/ (besucht am 23. 07. 2021).

[26] Jürgen Quade. *Linux-Treiber entwickeln : Eine systematische Einführung in die Gerätetreiber- und Kernelprogrammierung*. Heidelberg: Dpunkt-verlag, 2011. ISBN: 9783898646963.

[27] Ben Straub Scott Chacon. *Pro Git*. Apress, 9. Nov. 2014. 456 S. ISBN: 1484200772.

[28] *RFC 2326: Real Time Streaming Protocol (RTSP)*. 1998. URL: https://datatracker.ietf.org/doc/html/rfc2326 (besucht am 10. 07. 2021).

[29] *RFC 2974: Session Announcement Protocol*. 2000. URL: https://datatracker.ietf.org/doc/html/rfc2974 (besucht am 10. 07. 2021).

[30] *RFC 4566: Session Announcement Protocol*. 2000. URL: https://datatracker.ietf.org/doc/html/rfc2974 (besucht am 10. 07. 2021).

[31] The Internet Society, Hrsg. *RFC 3550: RTP - A Transport Protocol for Real-Time Applications*. 2003. URL: https://datatracker.ietf.org/doc/html/rfc3550 (besucht am 10. 07. 2021).

[32] StatCounter. *Mobile Operating System Market Share Worldwide*. 2021. URL: https://gs.statcounter.com/os-market-share/mobile/worldwide (besucht am 21. 06. 2021).

[33] Ole Tange. *GNU Parallel*. 2018. URL: https://www.gnu.org/software/parallel/.

[34] Merging Technologies. *ALSA RAVENNA/AES67 driver*. URL: https://bitbucket.org/MergingTechnologies/ravenna-alsa-lkm/src/master (besucht am 09. 07. 2021).

[35] Inc. The Linux Kernel Organization. *The Linux Kernel Archives*. 2021. URL: https://www.kernel.org/.

[36] W3Techs. *Usage of web servers for websites*. 2021. URL: https://w3techs.com/technologies/overview/web_server (besucht am 21. 06. 2021).

[37] Wikipedia. *Kommunikationsformen / Routing-Schemata*. URL: https://de.wikipedia.org/w/index.php?title=Multicast&oldid=212555926 (besucht am 11. 07. 2021).

[38] Wikipedia. *OSI-Modell*. URL: https://de.wikipedia.org/w/index.php?title=OSI-Modell&oldid=213601999 (besucht am 12. 07. 2021).

[39] Harald Zisler. *Computer-Netzwerke: Grundlagen, Funktionsweise, Anwendung ; [für Studium, Ausbildung, Beruf ; Theorie und Praxis: von der MAC-Adresse bis zum Router ; TCP/IP, IPv4, IPv6, (W)LAN, VPN, VLAN u.v.m ; Konfiguration, Planung, Aufbau und sicherer Betrieb von Netzwerken]*. 1. Aufl., 1., korr. Nachdr. Galileo Computing. Bonn: Galileo Press, 2012. 370 S. ISBN: 978-3-8362-1698-2.

Anhang

```
/*
* This is a template-generated file from BoneScript
*/

/dts-v1/;
/plugin/;

/{
    compatible = "ti,beaglebone", "ti,beaglebone-black";
    part_number = "BS_PINMODE_GPIO_50_0xf";

    exclusive-use =
        "GPIO.50",
        "gpio1_18";

    fragment@0 {
        target = <&am33xx_pinmux>;
        __overlay__ {
            bs_pinmode_GPIO_50_0xf: pinmux_bs_pinmode_GPIO_50_0xf {
                pinctrl-single,pins = <0x048 0xf>;
            };
        };
    };

    fragment@1 {
        target = <&ocp>;
        __overlay__ {
            bs_pinmode_GPIO_50_0xf_pinmux {
                compatible = "bone-pinmux-helper";
                status = "okay";
                pinctrl-names = "default";
                pinctrl-0 = <&bs_pinmode_GPIO_50_0xf>;
            };
        };
    };
};
```

Listing 1: BeagleBone-Black Device-Tree Overlay für GPIO Pin 50

```
{
  "http_port": 8080,
  "rtsp_port": 8854,
  "http_base_dir": "/opt/aes/webui",
  "log_severity": 2,
  "playout_delay": 0,
  "tic_frame_size_at_1fs": 48,
  "max_tic_frame_size": 1024,
  "sample_rate": 48000,
  "rtp_mcast_base": "239.1.0.1",
  "rtp_port": 5004,
  "ptp_domain": 0,
  "ptp_dscp": 48,
  "sap_mcast_addr": "239.255.255.255",
  "sap_interval": 30,
  "syslog_proto": "none",
  "syslog_server": "255.255.255.254:1234",
  "status_file": "/opt/aes/status.json",
  "interface_name": "eth0",
  "mdns_enabled": true,
  "mac_addr": "f0:45:da:7a:e7:fb",
  "ip_addr": "192.168.1.246",
  "node_id": "AES67 daemon a8c0f601"
}
```

Listing 2: AES67-Daemon Konfiguration

```
#! /bin/bash

### BEGIN INIT INFO
# Provides:          aes
# Required-Start:    $local_fs $network
# Required-Stop:     $local_fs
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: aes67 service
# Description:       Run AES67 service
### END INIT INFO

pname="aes"
exe="/opt/aes/aes67-daemon"
args="-c /opt/aes/daemon.conf"
pidfile="/var/run/${pname}.pid"
lockfile="/var/lock/subsys/${pname}"
log="/var/log/$pname"

[ -x $exe ] || exit 0

common_opts="--quiet --pidfile $pidfile"

# Carry out specific functions when asked to by the system
case "$1" in
  start)
    echo "Starting $pname ..."
    insmod /opt/aes/MergingRavennaALSA.ko > /dev/null 2>&1 || true
    start-stop-daemon --start $common_opts --make-pidfile \
      --background --startas /bin/bash -- -c "$exe $args > $log 2>&1"
    ;;
  stop)
    echo "Shutting down $pname ..."
    start-stop-daemon --stop $common_opts --signal INT --remove-pidfile
    ;;
  *)
    echo "Usage: /etc/init.d/$pname {start|stop}"
    exit 1
    ;;
esac

exit 0
```

Listing 3: Init.d Skript zum Starten des AES67 Treiber

```
import (
  "strings"
  "fmt"

  bbhw "github.com/btittelbach/go-bbhw"
  "github.com/gordonklaus/portaudio"
)

func getDevice(name string) (*portaudio.DeviceInfo, error) {
  devices, err := portaudio.Devices()
  chk(err)

  for _, d := range devices {
    if strings.Contains(d.Name, name) {
      return d, nil
    }
  }

  return nil, fmt.Errorf("Can't find device")
}

func openStream
  (deviceName string, inChannels int, outChannels int,
  sampleRate float64, framesPerBuffer int, args ...interface{})
  (*portaudio.Stream, error) {

  device, err := getDevice(deviceName)
  chk(err)

  var p portaudio.StreamParameters
  if inChannels == 0 && outChannels == 0 {
    return nil, fmt.Errorf("Input or output has to be used")
  } else if inChannels != 0 {
    p = portaudio.LowLatencyParameters(device, nil) // input
  } else {
    p = portaudio.LowLatencyParameters(nil, device) // output
  }

  p.Input.Channels = inChannels
  p.Output.Channels = outChannels
  p.SampleRate = sampleRate
  p.FramesPerBuffer = framesPerBuffer
  return portaudio.OpenStream(p, args...)
}

func chk(err error) {
```

```
47    if err != nil {
48      panic(err)
49    }
50  }
51
52  func enablePin(pin *bbhw.MMappedGPIO, enable bool) {
53    err := pin.SetState(enable)
54    chk(err)
55  }
```

Listing 4: Go Bibliothek mit Hilfsfunktionen

```
package main

import (
  "encoding/binary"
  "fmt"
  "os"
  "os/signal"
  "io"
  "github.com/gordonklaus/portaudio"
)

func main() {
  if len(os.Args) < 2 {
    fmt.Println("missing required argument:  input file name")
    return
  }
  fmt.Println("Playing.  Press Ctrl-C to stop.")

  sig := make(chan os.Signal, 1)
  signal.Notify(sig, os.Interrupt, os.Kill)

  fileName := os.Args[1]
  f, err := os.Open(fileName)
  chk(err)
  defer f.Close()

  id, data, err := readChunk(f)
  chk(err)
  if id.String() != "FORM" {
    fmt.Println("bad file format")
    return
  }
  _, err = data.Read(id[:])
  chk(err)
  if id.String() != "AIFF" {
    fmt.Println("bad file format")
    return
  }
  var c commonChunk
  var audio io.Reader
  for {
    id, chunk, err := readChunk(data)
    if err == io.EOF {
      break
    }
    chk(err)
```

```
47      switch id.String() {
48      case "COMM":
49        chk(binary.Read(chunk, binary.BigEndian, &c))
50      case "SSND":
51        chunk.Seek(8, 1) //ignore offset and block
52        audio = chunk
53      default:
54        fmt.Printf("ignoring unknown chunk '%s'\n", id)
55      }
56    }
57
58    //assume 48000 sample rate, mono, 32 bit, AIFF formatted file
59    portaudio.Initialize()
60    defer portaudio.Terminate()
61
62    out := make([]int32, 8192)
63    outA := make([]int32, len(out))
64    outB := make([]int32, len(out))
65
66    streamA, err := openStream("CTAG", 0, 1, 48000, len(outA), &outA)
67    chk(err)
68    defer streamA.Close()
69    chk(streamA.Start())
70    defer streamA.Stop()
71
72    streamB, err := openStream("RAVENNA", 0, 1, 48000, len(outB), &outB)
73    chk(err)
74    defer streamB.Close()
75    chk(streamB.Start())
76    defer streamB.Stop()
77
78    for remaining := int(c.NumSamples); remaining > 0; remaining -= len(out)
       {
79      if len(out) > remaining {
80        out = out[:remaining]
81      }
82      err := binary.Read(audio, binary.BigEndian, out)
83      if err == io.EOF {
84        break
85      }
86      chk(err)
87
88      // copy file buffer to stream buffers
89      copy(outA, out)
90      copy(outB, out)
91      chk(streamA.Write())
92      chk(streamB.Write())
```

```
93      copy(out, outA)
94
95      select {
96      case <-sig:
97        return
98      default:
99      }
100   }
101 }
102
103 func readChunk(r readerAtSeeker) (id ID, data *io.SectionReader, err error)
        {
104   _, err = r.Read(id[:])
105   if err != nil {
106     return
107   }
108   var n int32
109   err = binary.Read(r, binary.BigEndian, &n)
110   if err != nil {
111     return
112   }
113   off, _ := r.Seek(0, 1)
114   data = io.NewSectionReader(r, off, int64(n))
115   _, err = r.Seek(int64(n), 1)
116   return
117 }
118
119 type readerAtSeeker interface {
120   io.Reader
121   io.ReaderAt
122   io.Seeker
123 }
124
125 type ID [4]byte
126
127 func (id ID) String() string {
128   return string(id[:])
129 }
130
131 type commonChunk struct {
132   NumChans      int16
133   NumSamples    int32
134   BitsPerSample int16
135   SampleRate    [10]byte
136 }
```

Listing 5: Test I: Go Programm zum Senden von Audio an zwei Soundkarten

```go
package main

import (
  "fmt"
  "os"
  "os/signal"

  "github.com/gordonklaus/portaudio"
  bbhw "github.com/btittelbach/go-bbhw"
)

func main() {

  sig := make(chan os.Signal, 1)
  signal.Notify(sig, os.Interrupt, os.Kill)

  pin50 := bbhw.NewMMappedGPIO(50, bbhw.OUT)
  defer pin50.Close()

  enablePin(pin50, false)

  portaudio.Initialize()
  defer portaudio.Terminate()

  buffer := make([]float32, 8)
  stream, err := openStream("Merging RAVENNA", 1, 0, 48000, len(buffer),
    buffer)
  chk(err)

  fmt.Println("listening to stream, waiting for data ...")
  chk(stream.Start())
  defer stream.Close()

  nSamples := 0
  for {
    chk(stream.Read())
    nSamples += len(buffer)
    if nSamples > 0 {

      for _, v := range buffer {
        // check sound-level
        if v > 0 {
          enablePin(pin50, true)
          fmt.Println("received audio data", v)
          nSamples = 0
          return
```

```
46            }
47          }
48
49        }
50        select {
51        case <-sig:
52          return
53        default:
54        }
55    }
56    chk(stream.Stop())
57
58    fmt.Println("end")
59  }
```

Listing 6: Test II und III: Go Programm um beim Empfang von Audio einen GPIO Pin zu schalten

```
1  package main
2
3  import (
4    "fmt"
5    "os"
6    "os/signal"
7    "runtime"
8    "sync"
9    "time"
10   "math"
11
12   bbhw "github.com/btittelbach/go-bbhw"
13   "github.com/gordonklaus/portaudio"
14 )
15
16 func main() {
17   runtime.GOMAXPROCS(2)
18
19   sig := make(chan os.Signal, 1)
20   signal.Notify(sig, os.Interrupt, os.Kill)
21
22   pin50 := bbhw.NewMMappedGPIO(50, bbhw.OUT)
23   defer pin50.Close()
24
25   enablePin(pin50, false)
26
27   portaudio.Initialize()
28   defer portaudio.Terminate()
29
30   s := newStereoSine(256, 320, 48000)
31   defer s.Close()
32
33   time.Sleep(2 * time.Second)
34   fmt.Println("Starting Go Routines")
35
36   var wg sync.WaitGroup
37   wg.Add(2)
38   go func() {
39     defer wg.Done()
40
41     enablePin(pin50, true)
42   }()
43
44   go func() {
45     defer wg.Done()
46
```

```
47      chk(s.Start())
48      time.Sleep(2 * time.Second)
49      chk(s.Stop())
50    }()
51    wg.Wait()
52
53    fmt.Println("end")
54  }
55
56  type stereoSine struct {
57    *portaudio.Stream
58    stepL, phaseL float64
59    stepR, phaseR float64
60  }
61
62  func newStereoSine(freqL, freqR, sampleRate float64) *stereoSine {
63    s := &stereoSine{nil, freqL / sampleRate, 0, freqR / sampleRate, 0}
64    var err error
65    s.Stream, err = openStream("Merging RAVENNA", 0, 1, 48000, 0, s.
        processAudio)
66    chk(err)
67    return s
68  }
69
70  func (g *stereoSine) processAudio(out [][]float32) {
71    for i := range out[0] {
72      out[0][i] = float32(math.Sin(2 * math.Pi * g.phaseL))
73      _, g.phaseL = math.Modf(g.phaseL + g.stepL)
74      out[1][i] = float32(math.Sin(2 * math.Pi * g.phaseR))
75      _, g.phaseR = math.Modf(g.phaseR + g.stepR)
76    }
77  }
```

Listing 7: Test II: Go Programm zum parallelen Setzten eine GPIO Pins und Senden von Audio

Eidesstattliche Erklärung

Eidesstattliche Erklärung zur Bachelorarbeit

Ich versichere mit meiner eigenhändigen Unterschrift, dass ich die vorliegende Abschlussarbeit selbstständig und ohne unzulässige, fremde Hilfe angefertigt habe und dass ich alle von anderen Autoren*innen wörtlich übernommenen Stellen wie auch die an die Gedankengänge und Strukturen anderer Autoren*innen eng angelehnten Ausführungen meiner Arbeit besonders gekennzeichnet und die entsprechenden Quellen angegeben habe. Weiterhin versichere ich, dass die vorliegende Abschlussarbeit noch keiner Prüfungsbehörde vorgelegen hat.

$Unterschrift:$ $Ort, Datum:$