Implementierung eines IP-Audio-Streamingverfahrens (AoIP) auf eingebetteten Systemen

Bachelor-Thesis

zur Erlangung des akademischen Grades Bachelor of Science
an der Fachhochschule Kiel
im Fachbereich Informatik und Elektrotechnik
im Studiengang Informationstechnologie

Vorgelegt von: Stefan Neufeldt
Matrikelnummer: XXXXXX

Am: 10.08.2020

Erstprüfer: Prof. Dr. Robert Manzke, Fachhochschule Kiel
Zweitprüfer: Dipl. Ing. Helmut Scholz, Bayerischer Rundfunk

Inhaltsverzeichnis

Zusammenfassung

Diese Thesis beschäftigt sich mit der Implementierung eines nativen Linux-Gerätetreibers, genauer gesagt eines ALSA-Audiotreibers, welcher einige der Anforderungen des IP-Audio-Streamingverfahrens AES67 berücksichtigt und implementiert, um in einer Audio over IP-Umgebung (AoIP) verwendet werden zu können.
Das Streamingprotokoll AES67, welches vor allem im professionellen Broad–castumfeld eingesetzt wird, setzt bei der Übertragung auf das Real-Time Transport Protocol (RTP) in Kombination mit dem User Datagram Protocol (UDP).
Die Kernpunkte dieses Protokolls sind eine geringe Latenzzeit von <10 Millisekunden sowie eine hohe Audioqualität (24 Bit unkomprimiertes PCM-Signal mit 48 kHz Abtastrate) bei der Übertragung.
In der Arbeit sind hierfür zunächst anhand der Norm die erforderlichen technischen Anforderungen analysiert worden. Anschließend ist darauf basierend ein Audiotreiber entwickelt worden.

Abstract

This thesis deals with the implementation of a native Linux device driver, more precisely an ALSA audio driver, which takes into account and implements some of the requirements of the IP audio streaming method AES67 in order to be able to be used in an Audio over IP environment (AoIP).
The streaming protocol AES67, which is mainly used in the professional broadcast environment, relies on the Real-Time Transport Protocol (RTP) in combination with the User Datagram Protocol (UDP) for transmission.
The key points of this protocol are a low latency of <10 milliseconds and high audio quality (24-bit uncompressed PCM signal with 48 kHz sampling rate) during transmission.
In the thesis, the necessary technical requirements were first analyzed using the standard. An audio driver was developed based on these technical requirements.

Abkürzungsverzeichnis

A

ADU	*Analog-Digital-Umsetzer*
AES	*Audio Engineering Society*
ALSA	*Advanced Linux Sound Architecture*
AoIP	*Audio over IP*
ASoC	*ALSA System on Chip*

B

BR	*Bayerischer Rundfunk*

C

CRC	*Cyclic Redundancy Check*
CSRC	*Contributing Source*

D

DAU	*Digital-Analog-Umsetzer*
DiffServ	*Differentiated Services*
DSCP	*Differentiated Services Code Point*

E

ECN	*Explicit Congestion Notification*

F

FCS	*Frame Check Sequenz*
FIFO	*First In – First out*

G

GPL	*GNU Public License*

H

Hz	*Hertz*

I

ICMP	*Internet Control Message Protocol*
IEEE	*Institute of Electrical and Electronics Engineers*
IGMP	*Internet Group Management Protocol*
IP	*Internet Protocol*
ISO	*International Standards Organization*

L

LAN	*Local Area Network*

M

MAC	*Medium Access Control*
MLD	*Multicast Listener Discovery Protocol*
MTU	*Maximum Transmission Unit*

N

NDR	*Norddeutscher Rundfunk*

O

OSI	*Open Systems Interconnection*

P

PCM	*Pulscodemodulation*
PTPv2	*Precision Time Protocol in der 2. Version*

Q

QoS	*Qualtity of Service*

R

RFC	*Request for Comments*
RTCP	*Real-Time Control Protocol*
RTP	*Real-Time Transport Protocol*

S

SAP	*Session Announcement Protocol*
SDP	*Session Description Protocol*
SFD	*Starting Frame Delimiter*
SSRC	*Synchronization Source*
Sysfs	*Sys-Filesystem*

T

TAI	*Temps Atomique International (Internationale Atomzeit ohne Schaltsekunden)*
TCP	*Transmission Control Protocol*
TDM	*Time Division Multiplex*
ToS	*Type-of-Service*
TTL	*Time to live*

U

UDP	*User Datagram Protocol*

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Diese Bachelorarbeit beschäftigt sich mit der Entwicklung eines virtuellen Audiotreibers auf einem eingebetteten Linux-System zur Verwendung in einer Audio over IP-Umgebung (AoIP).

1.1. Audio over IP – Eine Begriffserklärung

Unter Audio over IP versteht man die Übertragung von digitalen Audiosignalen als Datenstrom über das Internetprotokoll (IP). Dabei werden im Sender mehrere Audio-Samples zusammen in ein IP-Paket verpackt und über ein lokales Netzwerk (LAN) zum Empfänger übertragen.

Einer der Vorteile bei dieser Übertragungsart von Audiosignalen ist es, dass sehr viele Audiosignale zusammen mit anderen IP-basierten Signalen über dieselbe Leitung transportiert werden können. Durch die Verwendung der Ethernet-Technologie kommt ein ausgereiftes System zum Einsatz, welches mit Hilfe von Switchen ein sehr großes und komplexes, auch über weite Distanzen nutzbares Netzwerk ermöglicht.

AoIP wird im professionellen Audiobereich erst seit einigen Jahren eingesetzt [1].
Es ist abzugrenzen zu Layer-1-Systemen und Layer-2-Systemen, welche auch als Audio over Ethernet bezeichnet werden. Sie ermöglichen zwar auch die Übertragung von mehreren Audiosignalen über Twisted-Pair-Kabel wie Cat5 oder Glasfasern, jedoch werden zur Übertragung nicht das IP-Protokoll, sondern eigene proprietäre Protokolle verwendet.
Während bei Layer-1-Systemen eine eigene Netzwerkinfrastruktur erforderlich ist, können bei Layer-2-Systemen die üblichen Netzwerkkomponenten verwendet werden.
Gemeinsam haben beide Typen jedoch, dass die darüberliegenden Schichten durch proprietäre Protokolle umgesetzt sind. [2]

1.2. Motivation und Relevanz der Aufgabenstellung

AES67 ist ein Audio over IP Standard, welcher es ermöglicht, Audiodaten in sehr hoher Qualität mit geringer Latenzzeit über das Internetprotokoll zu übertragen. Hauptsächlich wird dieser Standard im Bereich der Broadcasttechnik genutzt. [3]
AES67 wurde 2013 entwickelt, um eine Interoperabilität zwischen verschiedenen Herstellern zu ermöglichen [4]. Seitdem setzen immer mehr Hersteller auf diesen Standard und bauen ihre Produkte darauf auf.
Dabei sieht der Standard vor, dass die Audiopakete unkomprimiert über das verbindungslose User Datagram Protocol (UDP) und als Pulscodemodulation (PCM) übertragen werden. Als darüberliegende Schicht kommt das Real-Time Transport Protocol (RTP) zum Einsatz.
Die Synchronisation aller Geräte erfolgt über das Precision Time Protocol in der 2. Version (PTPv2), welches 2008 in der IEEE-1588-2008 definiert worden ist. [3]

Seit Erscheinen des AES67-Standards im Jahr 2013 gilt er mittlerweile im professionellen Audiobereich als etabliert, auch Technologien wie Ravenna oder Dante unterstützen einen AES67-kompatiblen Modus.
Auch der zur Übertragung von Videosignalen verwendete Standard SMPTE ST 2110 baut bei der Audioübertragung auf den AES67-Standard auf. [5]

Seitens des Bayerischen Rundfunks sowie der Open-Source Community kam der Wunsch nach einer virtuellen Soundkarte als ein Linux Treiber auf, damit der neue Standard auch in eigenentwickelten Projekten integriert werden kann [6].

1.3. Zielsetzung der Arbeit

Im Rahmen dieser Thesis ist eine virtuelle Soundkarte entwickelt worden, die auf Linux-Systemen lauffähig ist, um RTP-Streams empfangen und senden zu können.

1.4. Aufbau der Arbeit

Zunächst wird eine Übersicht über die Grundlagen der digitalen Audioübertragung, im Bereich der Netzwerktechnik, des AES67-Standards sowie des Betriebssystems Linux gegeben.
Darauf folgt eine Anforderungsanalyse, welche sich aus Sicht der Norm sowie des Stakeholders ergibt. Anschließend werden die vorgestellten Anforderungen ausgewertet.
Die allgemeine Funktionsweise des Treibers wird zunächst im vierten Kapitel vorgestellt. Dabei werden die Treiberarchitektur sowie die Datenstruktur erläutert.
Im fünften Kapitel dieser Arbeit wird die genaue Implementierung des Treibers beleuchtet.
Anschließend erfolgen eine Analyse und Bewertung des Entwicklungsprojektes.
Schlussendlich wird ein Gesamtfazit über die Arbeit gezogen.

2. Grundlagen

In diesem Kapitel werden Grundlagen in den folgenden vier Bereichen gelegt:

- Grundlagen der digitalen Audioübertragung
- Grundlagen im Bereich der Netzwerktechnik
- Einführung in den AES67-Standard
- Grundlagen des Betriebssystems Linux und der ALSA-Architektur

In den Grundlagen der digitalen Audioübertragung werden unter anderem Begriffe eingeführt und erläutert, die entscheidend für die Qualität digitaler Audiosignale sind. Des Weiteren wird der Ablauf der Pulscode-Modulation dargelegt.

Im Bereich der Netzwerktechnik wird zunächst der grundsätzliche Aufbau eines Netzwerks gezeigt. Außerdem wird ein Referenzmodell erklärt, anhand dessen die verwendeten Protokolle der Netzwerktechnik zugeordnet werden können. Auch der Aspekt der Qualität im Netzwerk wird genannt.

Im nächsten Abschnitt wird der AES67-Standard vorgestellt und auf die Anforderungen dieses Standards eingegangen.

Im vierten Abschnitt dieses Kapitels wird zunächst der grundsätzliche Aufbau eines Linuxsystems gezeigt. Außerdem werden ausgewählte Grundlagen, die zur Entwicklung eines Treibers notwendig sind, näher erläutert.

2.1. Grundlagen der digitalen Audioübertragung

Als analoges Signal bezeichnet man ein zeit- und wertekontinuierliches Signal, also ein fortlaufendes Signal ohne feste Stufen.
Ein analoges Signal kann mittels eines Analog-Digital-Umsetzers (ADU) in ein digitales, das heißt in ein zeit- und wertediskretes, Signal gewandelt werden. Genauso lassen sich digitale Signale mittels eines Digital-Analog-Umsetzers (DAU) in analoge Signale wandeln.
Der Ablauf einer Analog-Digitalwandlung und insbesondere die Pulscodemodulation werden im Folgenden kurz erläutert. [7]

2.1.1. Pulscodemodulation

Als Pulscodemodulation wird ein Verfahren zur Wandlung bzw. Darstellung eines analogen Signals in ein digitales Signal bezeichnet.
Den Ablauf einer Pulscodemodulation verdeutlicht die folgende Abbildung.

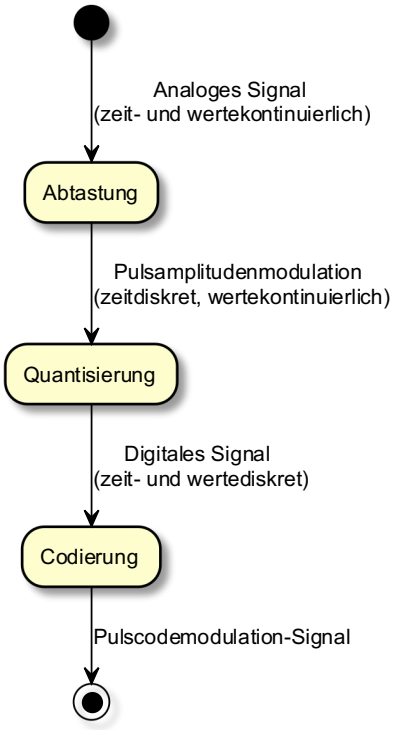

Abbildung 1: Ablauf Pulscodemodulation [Eigene Darstellung]

1. Das analoge Signal wird in Abhängigkeit der zuvor definierten Abtastfrequenz abgetastet. Dieses nun zeitdiskrete, aber wertekontinuierliche Signal wird als Pulsamplitudensignal bezeichnet.
2. Bei der Quantisierung und Codierung wird dem zuvor abgetasteten Signal ein, in Abhängigkeit von der Abtasttiefe, endlicher Zeichenvorrat zugeordnet. Das Signal ist nun zeit- und wertediskret und wird als Pulscodemodulation-Signal bezeichnet. [8] [9]

Die oben bereits erwähnten Parameter Abtastfrequenz sowie Abtasttiefe werden nun in den beiden folgenden Abschnitten näher erläutert.

Abtastfrequenz

Bei der Wahl der Abtastfrequenz ist das Abtasttheorem zu beachten. Vereinfacht besagt dies, dass die Abtastfrequenz mindestens doppelt so hoch gewählt werden muss, wie die höchste im Signal vorkommende Frequenz, die obere

Grenzfrequenz. Das zuvor Erläuterte wird auch anhand der folgenden Formel verdeutlicht: [10]

$$\frac{1}{T_a} = f_a \geq 2 \cdot f_g$$

$T_a = Abtastintervall$

$f_a = Abtastfrequenz$

$f_g = obere\ Grenzfrequenz$

Wird das Abtasttheorem nicht eingehalten, man spricht hierbei von einer Unterabtastung, werden höhere Signalfrequenzen fehlinterpretiert und erscheinen als tieffrequente Signale. Dies wird als Alias-Effekt bezeichnet.
Daher muss bei der Abtastung sichergestellt werden, dass das Eingangssignal keine höheren Frequenzanteile als die gewählte obere Grenzfrequenz enthält. Um dies zu erreichen, werden Tiefpassfilter eingesetzt, welche höhere Signalanteile herausfiltern. [10]

Da das menschliche Gehör Frequenzen zwischen 20 Hz und 20 kHz wahrnehmen kann, ist nach dem Shannon-Theorem folglich eine Abtastrate von mindestens 40 kHz erforderlich [11].
In der Praxis haben Filter keine ideale, rechteckige Filterwirkung, weshalb in den gewählten Abtastraten eine zusätzliche Toleranz (etwa 10%) berücksichtigt werden sollte.
Daher sind typische Abtastraten 44,1 kHz (CD-Qualität) sowie 48 kHz (im professionellen Bereich) üblich. Weitere, jedoch nicht so gängige Abtastraten wie 96 kHz sowie 192 kHz, sind ein Vielfaches von 48 kHz. [10]

Abtasttiefe

Der Wert der Abtasttiefe bestimmt den Quantisierungsfehler, welcher bei der Quantisierung eines analogen Signals entsteht. In der Regel wird ein möglichst geringer Quantisierungsfehler angestrebt, weshalb die Abtasttiefe für eine möglichst verlustfreie Wandlung nicht zu gering gewählt werden sollte. [12]

Ein gängiges Maß, um den Quantisierungsfehler auszudrücken, ist das sogenannte Quantisierungsrauschen, welches sich aus dem Quantisierungsfehler ergibt [12].

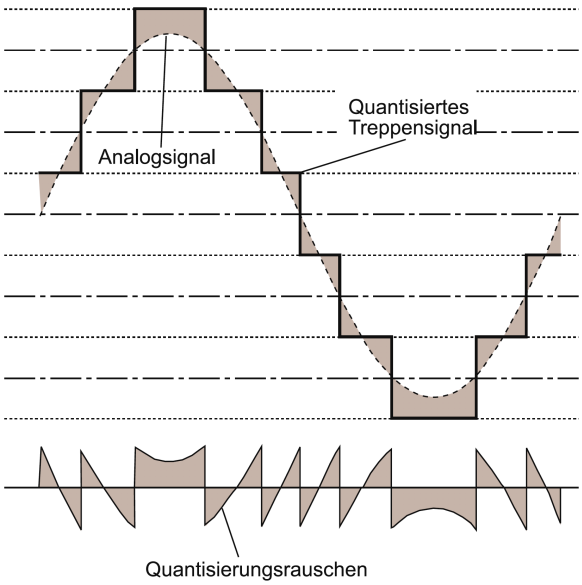

Abbildung 2: Quantisierungsfehler und Quantisierungsrauschen im Vergleich [12]

Wie die obenstehende Abbildung verdeutlicht, ist das Quantisierungsrauschen eine Differenzbildung aus dem quantisierten Signal und dem Analogsignal. Üblicherweise werden Abtasttiefen von 16 (CD-Qualität) oder 24 Bit (hochauflösende Qualität) verwendet [10].

Datenrate

Digitale Audiosignale werden in Strömen, den sogenannten Audio-Streams, übertragen. Die pro Sekunde anfallenden zu übertragenden Bits werden als Bitrate oder auch Datenrate bezeichnet. Dabei werden mehrkanalige Signale mittels des Zeitmultiplex-Verfahrens Time Division Multiplex (TDM) übertragen. Die Datenrate eines digitalen Audiosignals lässt sich mit folgender Formel berechnen: [10]

$$R = f_A \cdot M \cdot n$$

R = Datenrate

f_A = Abtastrate

M = Wortbreite (Abtasttiefe)

n = Anzahl der Kanäle

Zur Verdeutlichung sei als Beispiel ein Stereosignal mit der Quantisierungstiefe von 24 Bit und einer Abtastrate mit 48 kHz genannt:

$$R = 48\,kHz \cdot 24\,bit \cdot 2 = 2{,}304\,\frac{Mbit}{s}$$

2.2. Grundlagen der Netzwerktechnik

Der folgende Abschnitt behandelt die Grundlagen der Netzwerktechnik, um ein grundlegendes Verständnis zu vermitteln, welches für die weiteren Themen dieser Thesis notwendig ist.

2.2.1. Begriffsdefinition

Unter einem Rechnernetzwerk versteht man mehrere miteinander verbundene Computer, welche in diesem Zusammenhang auch als Host bezeichnet werden. Dadurch sind diese in der Lage, Informationen untereinander auszutauschen. [13]

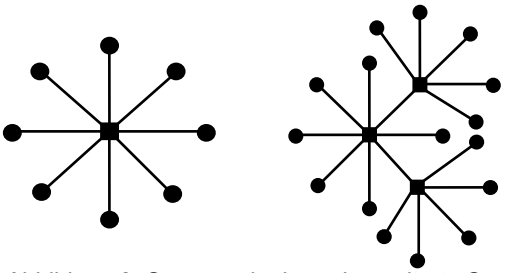

Abbildung 3: Sterntopologie und erweiterte Sterntopologie [Eigene Abbildung]

Private Netzwerke innerhalb eines Gebäudes mit Reichweiten von wenigen Kilometern werden als Local Area Network (LAN) bezeichnet. Üblicherweise werden die einzelnen Rechner innerhalb eines LANs in einer (erweiterten) Sterntopologie über Switches miteinander verbunden, wie in Abbildung 3 dargestellt. Dabei beträgt die Leitungsgeschwindigkeit der einzelnen Teilnehmer in der Regel 1000 Megabit/s. [13]

2.2.2. ISO-OSI-Referenzmodell

Das ISO-OSI-Referenzmodell ist eine Schichtenarchitektur und unterteilt die Kommunikation in sieben verschiedene Schichten.
Jede Schicht erfüllt eine genau definierte Aufgabenstellung und nutzt hierfür die darunterliegende Schicht.
Im Gegenzug bietet jede Schicht ihren Dienst der nächst höheren Schicht an.

Die folgende Abbildung stellt den Aufbau des Referenzmodells dar. Die einzelnen Schichten werden im folgenden Abschnitt näher erläutert. [13]

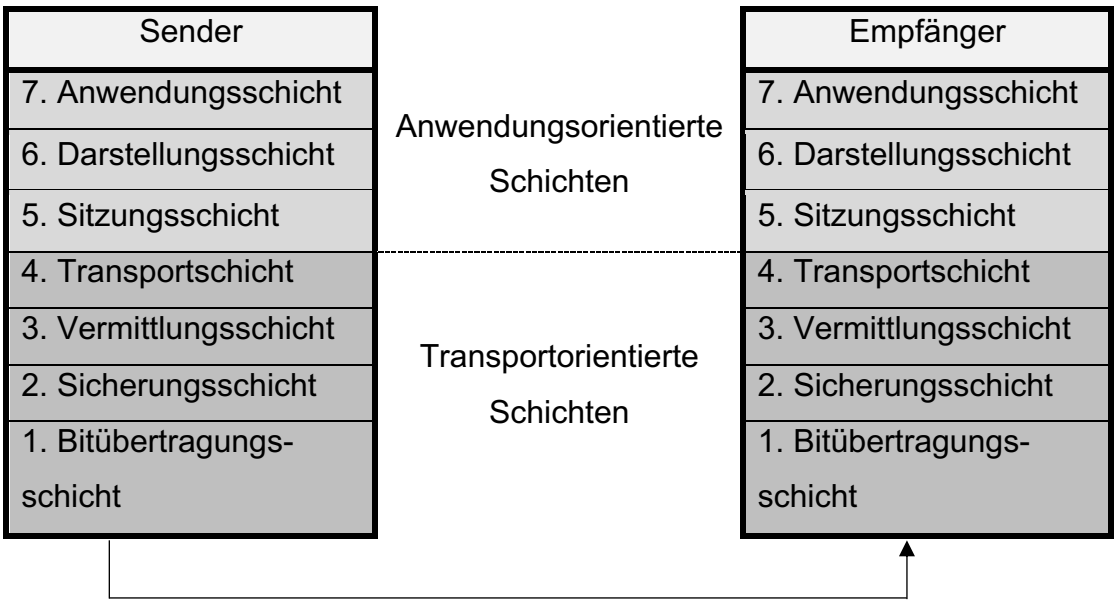

Abbildung 4: ISO-OSI-Referenzmodell [Eigene Abbildung]

Bitübertragungsschicht

Diese Schicht legt die technischen, mechanischen und elektrischen Eigenschaften fest, um die einzelnen Bits über ein Übertragungsmedium zu transportieren.
Das häufig in LAN-Netzwerken verwendete Ethernet-Protokoll, welches als IEEE 802.3 standardisiert ist, umfasst unter anderem Festlegungen für Kabeltypen und Stecker sowie deren Kodierungen. [13]

Sicherungsschicht

Die Sicherungsschicht hat die Aufgabe, den darüberliegenden Schichten einen zuverlässigen Kommunikationskanal zur Verfügung zu stellen, der frei von unerkannten Übertragungsfehlern ist.
Dazu werden Verfahren zur Fehlerkorrektur, Flusskontrolle und Kollisionserkennung definiert.
Außerdem wird die korrekte Adressierung in dieser Schicht sichergestellt.
Dazu werden die Eingangsdaten, also einzelne Bits, in Datenblöcke, die sogenannten Datenrahmen (Engl. Data Frames), aufgeteilt und anschließend sequenziell übertragen.

Die physikalische Adressierung der Netzwerkteilnehmer wird innerhalb des Netzsegments mithilfe von MAC-Adressen sichergestellt.

Mithilfe von Prüfsummen kann der Empfänger fehlerhaft übertragene Daten erkennen und ggf. korrigieren. [13]
Der genaue Aufbau des häufig verwendeten Ethernet-Datenrahmens wird in Abschnitt 2.2.3 genauer erläutert.

Vermittlungsschicht

Die Vermittlungsschicht hat die Aufgabe, Datenpakete über mehrere Teilnetze hinweg vom Ursprungs- zum Bestimmungsort zu übertragen.
Router ermöglichen die Kopplung mehrerer Teilnetze.
Im häufig verwendeten Internet Protocol (IP) besitzt jeder Teilnehmer eines Netzwerks eine IP-Adresse, mit deren Hilfe die logische Adressierung innerhalb des Netzwerks erfolgt.
Durch Routingverfahren werden Zustellungen von Paketen über mehrere Netzwerkknoten hinweg ermöglicht und somit wird die korrekte Übertragung der einzelnen Pakete sichergestellt. [13]
Das häufig eingesetzte Internet Protocol wird in Abschnitt 2.2.3 genauer vorgestellt.
Zu den Aufgaben der Vermittlungsschicht gehört auch die Sicherstellung der Dienstgüte (Quality of Service). Dies wird ebenfalls in Abschnitt 2.2.3 erläutert.

Transportschicht

Innerhalb der Transportschicht wird zwischen dem verbindungsorientierten Protokoll TCP und dem verbindungslosen UDP unterschieden.
Das TCP ermöglicht eine zuverlässige Ende-zu-Ende-Kommunikation zwischen zwei Hosts.
Vor der eigentlichen Übertragung von Nachrichten wird eine Verbindung mittels des Three-Way-Handshakes aufgebaut.
Danach wird im Sender der Bytestrom der Sitzungsschicht in einzelne Nachrichten zerlegt. Anschließend werden beim Empfänger diese Nachrichten wieder zu einem Ausgabestrom zusammengesetzt und der Sitzungsschicht übergeben.
Dem Sender wird der korrekte Empfang der einzelnen Pakete stets durch Quittierungsnachrichten bestätigt.

Das UDP ist ein verbindungsloses Protokoll, wodurch der oben beschriebene Aufbau einer Verbindung entfällt. Nachteilig ist jedoch, dass es keine Kontrollmechanismen gibt, die eine korrekte Übertragung sicherstellen.
Das UDP wird vorrangig dort eingesetzt, wo die Schnelligkeit der Zustellung wichtiger als die Genauigkeit ist. Dies ist bspw. bei der Audio- und Videoübertragung der Fall. [13]
Auf das UDP wird in Abschnitt 2.2.3 genauer eingegangen.

Anwendungsorientierte Schichten

Die OSI-Schichten 5 – 7 zählen zu den anwendungsorientierten Schichten. Sie bauen auf die darunterliegenden transportorientierten Schichten auf und müssen sich somit nicht mehr um den korrekten Transport der einzelnen Nachrichten kümmern. Stattdessen bilden sie eine Schnittstelle zwischen dem Netzwerk und den Anwendungen. [13]
Im Vergleich zu den unteren Schichten kommen bei den anwendungsorientierten Schichten auch heutzutage noch eine Vielzahl von Protokollen zum Einsatz.

2.2.3. Vorstellung ausgewählter Dienste und Protokolle

Im folgenden Abschnitt werden die Dienste und Protokolle erläutert, die im Bereich der Audio over IP-Umgebungen eine wichtige Rolle spielen.

Ethernet-Paket (Layer 1 und 2)

Die folgende Abbildung zeigt den Aufbau eines Ethernet-Pakets Ethernet II nach IEEE 802.3.

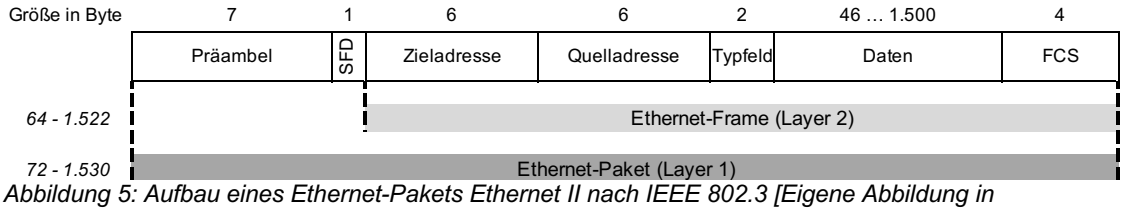

Abbildung 5: Aufbau eines Ethernet-Pakets Ethernet II nach IEEE 802.3 [Eigene Abbildung in Anlehnung an [14]]

Die 7 Byte lange Präambel, welche die alternierende Bitfolge 1010…10 enthält, kennzeichnet den Beginn eines Ethernet-Pakets und dient den Netzwerkgeräten zur Bit-Synchronisation.

Dieser Präambel folgt ein 1 Byte großer Starting Frame Delimiter (SFD), welcher die feste Bitfolge von 10101011 hat und den Beginn des eigentlichen Frames mitteilt.
Anschließend folgt das Ethernet-Frame, welches dem 2. Layer zuzuordnen ist, in dem zuerst die Zieladresse, anschließend die Quelladresse jeweils als eine 6 Byte große MAC-Adresse übertragen wird.
Das darauffolgende Typfeld gibt an, welches Protokoll in der darüberliegenden Vermittlungsschicht verwendet wird.
Darauf folgen die eigentlichen Nutzdaten der darüberliegenden Vermittlungsschicht mit einer maximalen Länge von 1500 Byte. Diese maximale Länge wird als Maximum Transmission Unit (MTU) bezeichnet.
Nach den Nutzdaten folgt die Frame Check Sequenz (FCS), welche eine 4-Byte große CRC-Prüfsumme darstellt. [13]

IP – Internet Protocol (Layer 3)

Das Internet Protocol (IP) hat die Aufgabe, den Datenverkehr in IP-Paketen zu organisieren. Dazu gehört unter anderem das korrekte Zustellen der Pakete durch Paket-Routing. Es ist in dem ISO-OSI-Referenzmodell der Vermittlungsschicht zuzuordnen.
Im Vergleich zu der Sicherungsschicht, wo die Zustellung durch MAC-Adressen sichergestellt wird, erfolgt die Zuordnung der Teilnehmer durch IP-Adressen. [13]

Details zu IPv4

Der ältere IPv4-Standard, spezifiziert in der RFC 791, verwendet zur Adressierung 32-Bit große IP-Adressen.
Während ursprünglich der gesamte IP-Adressraum in Netzklassen von A bis E unterteilt worden ist, wird heutzutage stattdessen eine Subnetzmaske verwendet, welche den Netz- und Hostbereich angibt. Dadurch ist der Betrieb von Teilnetzen (Subnetting) möglich. Dabei wird die IP-Adresse in eine Netzadresse sowie eine Hostadresse aufgeteilt.
Aufgrund der stark wachsenden Zahl internetfähiger Geräte wird angenommen, dass der Adressraum in naher Zukunft erschöpft sein wird. Daher verwendet der nachfolgende IPv6-Standard einen 128-Bit großen Adressraum, was

$3{,}4 \cdot 10^{38}$ statt $4{,}3 \cdot 10^{9}$ Adressen ermöglicht. Derzeit kommen sowohl IPv4 als auch IPv6 zum Einsatz. [13]

Aufbau eines IPv4-Pakets

Der Aufbau eines IPv4-Pakets wird anhand der folgenden Abbildung verdeutlicht und in der folgenden Tabelle beschrieben.
Die Übertragung findet von links nach rechts statt, das heißt, sie beginnt mit dem Versionsfeld und endet mit dem Optionenfeld. Im Anschluss daran werden die eigentlichen Nutzdaten übertragen. [13]

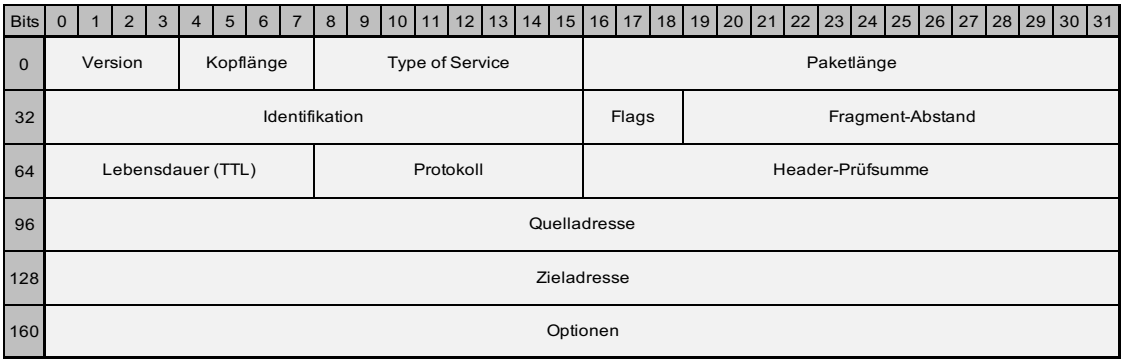

Abbildung 6: Aufbau des IPv4-Headers [Eigene Abbildung in Anlehnung an [15]]

Feldname	Beschreibung	Feldgröße in Bit
Version	Gibt die zugehörige Version des Protokolls des Datengramms an.	4
Kopflänge	Angabe über die Länge des Headers.	4
Type of Service	Angabe über den gewünschten Dienst (Priorität, Verzögerung, Durchsatz, Zuverlässigkeit).	8
Paketlänge	Länge des Datengramms inklusive Header und Nutzdaten.	16
Identifikation	Nummerierung der Datenpakete. Dient zur Zuordnung von Datengrammen durch den Zielhost zu Fragmenten.	16
Flags	1. Bit ungenutzt. 2. Bit steht für Don't Fragment, was bedeutet, dass eine Fragmentierung nicht gewünscht ist. 3. Bit steht für More Fragments, welches angibt, dass noch weitere Fragmente eines Datengramms folgen werden.	3

Fragment-Abstand	Gibt die zugehörige Stelle eines Fragments im Datengramm an.	13
Lebensdauer (TTL)	Zählwert, um die Lebensdauer von Paketen zu begrenzen. Bei jedem Weiterleiten des Pakets wird der Wert dekrementiert. Hat der Wert 0 erreicht, wird das Paket verworfen.	8
Protokoll	Gibt das verwendete Protokoll der Transportschicht an[1].	8
Header-Prüfsumme	Prüfsumme, mit deren Hilfe die Korrektheit des IP-Headers geprüft werden kann.	16
Quelladresse	Netz- und Hostadresse des Senders.	32
Zieladresse	Netz- und Hostadresse des Empfängers.	32
Optionen	Das Optionsfeld wird hauptsächlich zu Diagnosezwecken eingesetzt und findet ansonsten kaum Verwendung.	32

Tabelle 1: Aufbau des IP-Headers [Eigene Abbildung]

IP-Multicast (Layer 3)

Es wird unter anderem zwischen den Adressierungsarten Unicast, Multicast und Broadcast unterschieden.

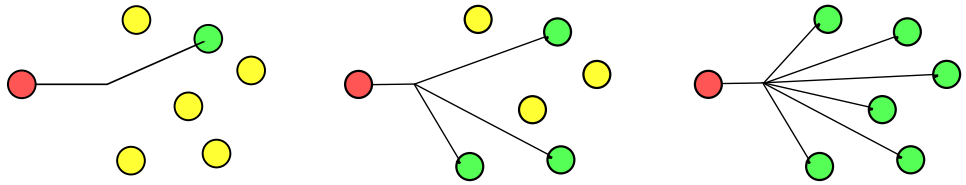

Abbildung 7: Vergleich von Unicast, Multicast und Broadcast [16] [17] [18]

Eine direkte Verbindung zwischen zwei Hosts (Punkt-zu-Punkt-Verbindung) wird als Unicast bezeichnet (siehe Abbildung 7 links).
Soll bei einer Unicast-Verbindung eine Nachricht an mehrere Empfänger zugestellt werden, muss diese mehrfach losgeschickt werden. Somit steigt der Bandbreitenbedarf für jeden zusätzlichen Empfänger sowohl für den Sender als auch das gesamte Netzwerk an.
Im Vergleich dazu bezeichnet man das Übertragen einer einzigen Nachricht an mehrere Empfänger als Multicasting (siehe Abbildung 7 Mitte). Dazu wird die

[1] Nummerierung der Protokolle im RFC 1700 definiert [13].

Nachricht vom Sender (Rot) an eine für den Sender anonyme Gruppe (Grün) übertragen. Es handelt sich um eine Punkt-zu-Mehrpunkt-Verbindung.
Im Vergleich zum Broadcasting (siehe Abbildung 7 rechts) wird die Nachricht nicht an alle Teilnehmer im Netz, sondern nur an die Teilnehmer der entsprechenden Multicast-Gruppe übermittelt. [13]

Multicast-Routing

Der zugrundeliegende Routing-Algorithmus wird als Multicast-Routing bezeichnet. Alle Komponenten im Netzwerk (Host, Router, Switches), die an dem Multicast-Routing beteiligt sind, müssen Multicast-Routing unterstützen.
Jeder Empfänger (Teilnehmer) meldet sich zuvor in seiner Gruppe an und ist anschließend Teil der entsprechenden Multicast-Gruppe.
Das Management der Gruppen in IPv4-Netzen wird durch das im nächsten Abschnitt erläuterte IGMP und in IPv6-Netzen durch MLD realisiert.
Die Verwendung von Multicast ist nur in Verbindung mit verbindungslosen Protokollen wie UDP möglich, da nur bei diesen keine explizite Punkt-zu-Punkt-Verbindung erforderlich ist. [13]

Multicast-Adressbereich

Für Multicast ist der Adressbereich im IPv4-Netz von 224.0.0.0 bis 239.255.255.255 vorgesehen. Dieser Adressraum wird auch als Class-D bezeichnet. In diesem stehen Adressen für über 250 Millionen Gruppen zur Verfügung. [13]

Anwendungsbereich von Multicast

Die Nutzung von Multicast ist vor allem deshalb sinnvoll, da das zeitgleiche Übertragen von Nachrichten nicht die Datenübertragungsrate beim Sender erhöht. Der Sender muss die jeweilige Nachricht nur ein einziges Mal übertragen. Die Vervielfältigung der Datenpakete kann in jedem einzelnen Verteiler (Router, Switch oder Hub) auf der Route stattfinden. Dadurch wird erreicht, dass jede Nachricht nur einziges Mal zum nächsten Verteilungspunkt übertragen wird und es keinen redundanten Netzwerkverkehr gibt.
Es ist somit unerheblich, wie viele Teilnehmer einer Multicast-Gruppe angehören. Insgesamt wird hiermit die Übertragungsrate des gesamten Netzwerks geschont. [13]

ICMP (Layer 3)

Mittels des Internet Control Message Protocols (ICMP) können Informations- und Fehlermeldungen wie bspw. Diagnose-Informationen zwischen Netzwerkteilnehmern übermittelt werden.
Das ICMP ist im RFC 792 definiert und Bestandteil des IPv4. Für IPv6 wird analog dazu das ICMPv6 verwendet.

Aufbau eines ICMP-Pakets

Das ICMP nutzt zur Übertragung den Header des IPv4. Hierbei wird die Protokollversion auf den Wert 1 (= ICMP) und der Type of Service auf 0 gesetzt. Den Aufbau eines ICMP-Pakets verdeutlicht auch die folgende Abbildung.

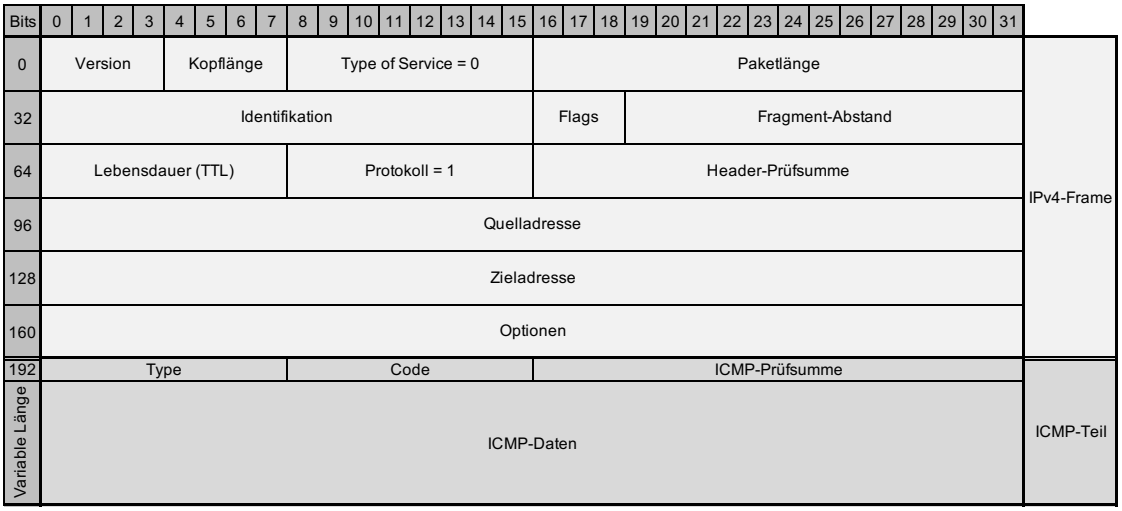

Abbildung 8: Aufbau des ICMP auf Basis des IPv4-Headers [Eigene Abbildung in Anlehnung an [19]]

Der Nachrichtentyp wird innerhalb des ICMP-Teils in einem 1 Byte großen Type-Feld angegeben. Mittels des Code-Felds wird die Nachricht zusätzlich zum Type genauer spezifiziert. Mit der ICMP-Prüfumme lässt sich die übermittelte Nachricht auf Korrektheit überprüfen.
Im Anschluss folgen die ICMP-Daten. Diese können wiederum in einem eigenen strukturierten Format übertragen werden. [19]

IGMP (Layer 3)

Das Internet Group Management Protocol (IGMP) ist ein Management-Protokoll, welches die Verwaltung von IP-Multicast-Gruppen ermöglicht. Es ist erstmalig 1989 in der RFC 1112 spezifiziert worden [20]. Die zweite Version ist

1997 in der RFC 2236 erschienen und 2002 in der Version 3 als RFC 3376 aktualisiert worden [21] [22].

Das IGMP ermöglicht in einem IPv4-Netzwerk den Versand von IP-Datenströmen an mehrere Empfänger.
Dazu müssen die Empfänger mittels des IGMP zuerst dem Router bzw. Switch mitteilen, dass sie einer Multicast-Gruppe beitreten möchten. Daraufhin sind diese in der Gruppe registriert und die entsprechenden IP-Datenströme werden an alle Teilnehmer dupliziert.
Sowohl die Hosts, die einen Multicast-Stream empfangen möchten, als auch die Router und Switches müssen somit das IGMP implementieren. [13]

Die 2. Version des Protokolls ermöglicht das explizite Austreten aus einer Gruppe, während dies in der ersten Version lediglich durch ein Timeout realisiert worden ist.
Die meisten Geräte verwenden heutzutage Version 2 oder 3.
Jede Version ist jedoch auch abwärtskompatibel, sodass alle Versionen miteinander verwendet werden können. [23]

MLD (Layer 3)

Das Multicast Listener Discovery Protocol (MLD) verwaltet IP-Multicast-Gruppen in IPv6-Netzwerken und ist an IGMP angelehnt. Dabei entspricht die erste Version des MLD, veröffentlicht im Jahr 1999 als RFC 2710, funktional der IGMPv2 und die zweite Version, veröffentlicht im Jahr 2004 unter der RFC 3810, der IGMPv3 [24] [25].

Quality of Service (QoS)

Unter der Dienstqualität, englisch als Quality of Service (QoS) bezeichnet, versteht man die qualitative Güte eines Kommunikationsnetzes aus Sicht des Anwenders.
Dabei lassen sich verschiedene Qualitätsmerkmale wie Bandbreite, Paketverluste oder Latenz als Merkmal nehmen. [13]

DiffServ (Schicht 3)

Differentiated Services (DiffServ) ist ein System zur Klassifizierung von IP-Paketen, um eine Priorisierung der QoS zu ermöglichen.
Während ursprünglich, wie oben beschrieben, im IPv4-Paket das Type-of-Service (ToS)-Byte zur Kennzeichnung der Prioritäten (unterteilt in Priorität, Verzögerung, Durchsatz, Zuverlässigkeit) verwendet wurde, sieht DiffServ ein alternatives Schema zur Klassifizierung von IP-Paketen vor.
Das in der RFC 2474 beschriebene Schema sieht vor, das 8-Bit große ToS-Feld des IPv4-Pakets, welches nun als DS-Feld bezeichnet wird, in ein 6-Bit großes Differentiated Services Code Point (DSCP)-Feld und ein 2-Bit großes Explicit Congestion Notification (ECN)-Feld aufzuteilen.
Zunächst werden mehrere Klassen von Services gebildet, welche mit unterschiedlicher Priorität im Netzwerk behandelt werden sollen. Das DSCP-Feld sieht einen Wert von 0 bis 63 vor, wobei ein höherer Wert nicht zwingend einer höheren Priorität entspricht. Stattdessen erfolgt die Zuordnung der Klassen durch die Netzwerk-Infrastruktur, was eine entsprechende Konfiguration dieser voraussetzt. [26]

UDP – User Data Protocol (Layer 4)

Das User Data Protocol (UDP), spezifiziert in der RFC 768, ist ein verbindungsloses Protokoll, wodurch es möglich ist, Informationen zu übertragen, ohne zuvor eine Verbindung explizit zwischen zwei Systemen aufzubauen.
Auch die in verbindungsorientierten Protokollen übliche Bestätigung (acknowledges) über den korrekten Erhalt einer Nachricht entfällt bei diesem Protokoll. Dadurch ist es unter anderem möglich, eine Nachricht an mehrere Teilnehmer zu übertragen. Diese Übertragungsart, Multicasting genannt, ist weiter oben bereits näher erläutert worden. [13]

Im Vergleich zu verbindungsorientierten Protokollen wie das Transmission Control Protocol (TCP) ist das UDP schneller und benötigt weniger Overhead, was jedoch auf Kosten der Zuverlässigkeit geschieht. Es gilt als ein verwaltungsarmes, aber unzuverlässiges Protokoll, weil eine korrekte Übermittlung von Datenblöcken nicht garantiert werden kann.

Es wird vorrangig dort eingesetzt, wo die Priorität auf Schnelligkeit und nicht auf Zuverlässigkeit liegt. [13]

Aufbau eines UDP-Pakets

Die folgende Abbildung zeigt den Aufbau eines UDP-Pakets, welcher nachfolgend genauer beschrieben wird.

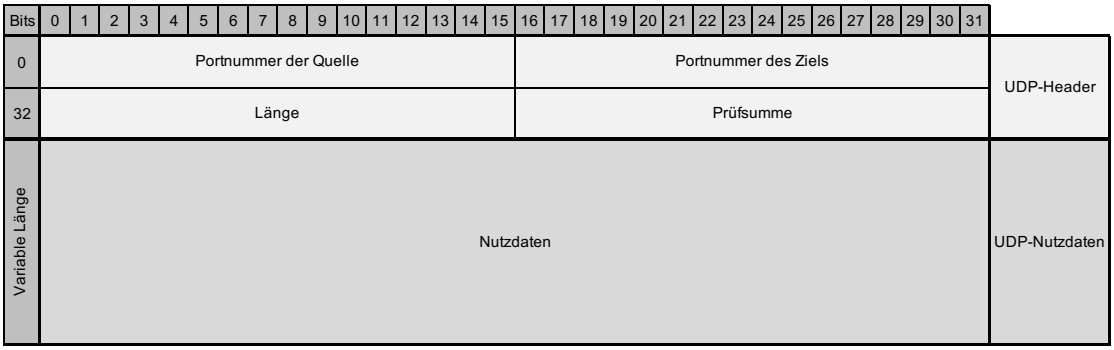

Abbildung 9: Aufbau eines UDP-Pakets [Eigene Abbildung in Anlehnung an [27]]

In dem 8 Byte großen Header des UDP-Pakets wird zuerst sowohl die Portnummer der Quelle als auch des Ziels angegeben und adressiert so die Endpunkte der Verbindung.
Die Länge gibt die gesamte Länge des UDP-Pakets einschließlich des Headers an.
Die Prüfsumme dient zur Verifikation des übermittelten Pakets.
Im Anschluss daran folgen die Nutzdaten des UDP-Pakets. Beispielsweise kann das RTP zu den Nutzdaten gehören, welches im folgenden Abschnitt erläutert wird. [27] [28]

RTP – Real-Time Transport Protocol (Layer 5 – 6)

Das Real-Time Transport Protocol (RTP), spezifiziert in der RFC3550, dient zur Echtzeitkommunikation im Bereich der Audio- Videoanwendungen.
Aufgrund der Eigenschaften ist es im ISO-OSI-Referenzmodell sowohl der Sitzungs- als auch der Darstellungsschicht zuzuordnen.

Das RTP wird in der Regel zusammen mit dem zuvor erläuterten verbindungslosen Protokoll UDP verwendet. Dadurch werden auch Multicast-Anwendungen auf Basis des RTP ermöglicht.

RTP-Pakete werden als zusammenhängende Folge von Segmenten übertragen, die im Empfänger wieder zu einem Datenstrom zusammengesetzt werden können. [29]

Aufbau eines RTP-Pakets

Der Aufbau eines RTP-Pakets wird im Nachfolgenden erläutert und anhand der folgenden Abbildung verdeutlicht.

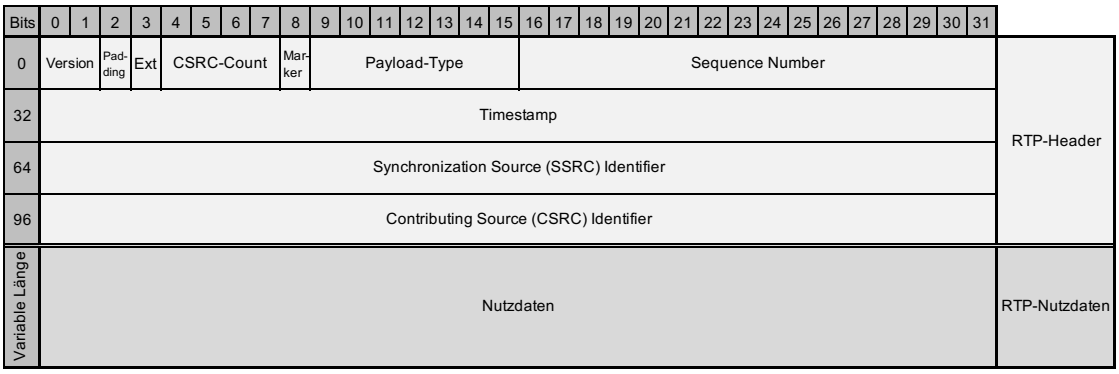

Abbildung 10: Aufbau eines RTP-Pakets [Eigene Abbildung in Anlehnung an [30]]

Die ersten beiden Bits im RTP-Header dienen zur Kennzeichnung der verwendeten Version des RTP-Pakets.
Das Paddingbit kennzeichnet, dass Füllbytes am Ende der Payload angehängt sind, die nicht zum eigentlichen Dateninhalt gehören.
Das Extensionbit (Ext) wird gesetzt, wenn auf dem RTP-Header noch ein zusätzlicher Header folgt. [29]

Sowohl der CSRC-Count, das Marker-Bit, die Synchronization Source (SSRC) als auch die Contributing Source (CSRC) werden in Systemen mit mehreren Sendern innerhalb einer Multicastadresse, wie bspw. einem Konferenzsystem verwendet. Da die AES67-Norm nur einen Sender zulässt, werden diese Bits nicht gesetzt. [29]

Das Payload-Type-Feld gibt das Format und die Codierung der übertragenen Payload an. In der RFC 3551 ist eine Liste[2] über die Zuordnung enthalten. Da die in der AES67-Norm verwendeten Codierungen nicht enthalten sind, werden

[2] RFC 3551, Payload Type Definitions, online unter: https://tools.ietf.org/html/rfc3551#section-6.

als Payload-Type Werte zwischen 96 und 127 verwendet, welche kennzeichnen, dass die Zuordnung dynamisch erfolgt. [29]
Die Sequence-Number wird mit jedem übertragenen Paket erhöht. Mit ihr kann die Reihenfolge der Pakete sichergestellt werden.
Der Timestamp dient dem Empfänger zur Synchronisation und orientiert sich an der Samplerate der Nutzlast.
Im Anschluss folgt die Nutzlast des RTP-Pakets. [29] [31]

Real-Time Control Protocol (RTCP)

Das Real-Time Control Protocol (RTCP) dient zur Steuerung von RTP-Streams und ist im RFC 3550 definiert.
Hierbei übermittelt der Empfänger eines RTP-Streams dem Sender periodisch Nachrichten, um diesen die derzeitige Güte der Übertragung (QoS) mitzuteilen. [29]

2.2.4. Zusammenfassende Darstellung der verwendeten Protokolle

Die zuvor vorgestellten Protokolle werden zur Übertragung von Audio over IP ineinander verschachtelt und bauen jeweils aufeinander auf. Das verdeutlicht auch die folgende Grafik.

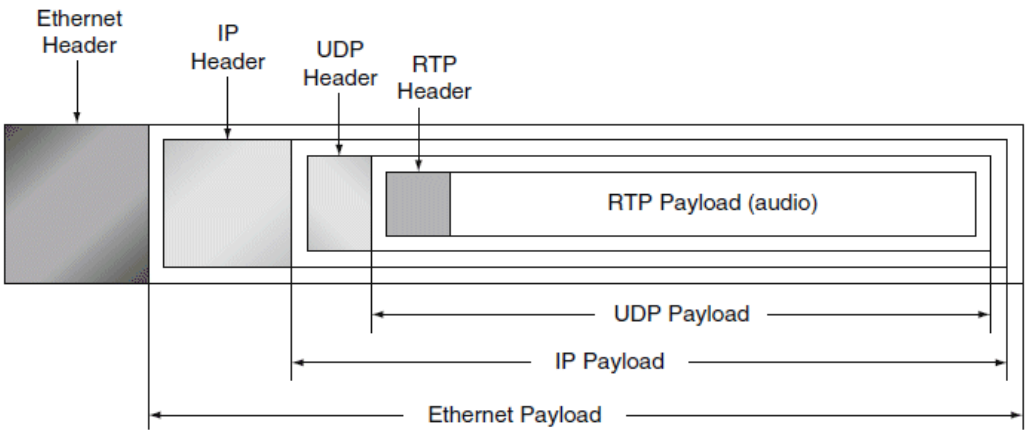

Abbildung 11: Darstellung der verwendeten Protokolle zur Echtzeitübertragung [32]

Die eigentlich zu übertragende Nutzlast (RTP Payload) wird mit einem RTP-Header versehen. Sie ergeben zusammen das RTP-Paket.
Dieses entspricht der Payload des UDP-Pakets. Dieser Vorgang gilt entsprechend für die anderen enthaltenen Protokolle.

Wie hier gut erkennbar ist, dient das Paket einer höheren Schicht der niedrigeren Schicht als Payload.

2.3. AES67-Standard

AES67 ist ein 2013 von der Audio Engineering Society (AES) veröffentlichter Standard, um Audio over IP mit geringer Latenz zu übertragen.
Dabei wurden nicht alle Komponenten neu definiert, sondern verschiedene, bereits bestehende und etablierte Normen und Standards kombiniert, um so einen neuen Standard zu ergeben. [2]

Der Standard ist entwickelt worden, um eine Audiokommunikation zwischen Geräten verschiedener Hersteller zu ermöglichen, die den Anforderungen von professioneller Audioqualität und niedriger Latenz gerecht werden.
Dabei versteht die Norm unter professioneller Audioqualität eine Audiokodierung mit mindestens 16-Bit Abtasttiefe und 44,1 kHz Abtastfrequenz. Die Verzögerungszeit wird mit kleiner als 10 ms angegeben. [3]

Durch die dadurch entstehenden hohen Netzwerkanforderungen sieht der Standard einen Einsatz ausschließlich im LAN und nicht im Internet vor [3].

Als Anwendungsbereich sieht die Norm die Verwendung in professionellen Audioanwendungen sowohl für Festinstallationen (Konzerthäuser etc.) als auch mobile Systeme (Live-Tournee) vor. Aber auch in Rundfunkstudios, Musikproduktionen sowie Postproduktionen kann dieser Standard eingesetzt werden. [3]

Die Anforderungen, die an ein Gerät gestellt werden, welches diesen Standard implementiert, werden im Kapitel Anforderungsanalyse näher erläutert.

2.4. Linux

Im folgenden Abschnitt werden Grundlagen in Bezug auf Linux, insbesondere den Linux-Kernel, erläutert. Dieses Wissen ist essenziell für die Entwicklung eines Kerneltreibers.

2.4.1. Aufbau des Betriebssystems

Ein Betriebssystem besteht aus dem Betriebssystemkern (Kernel) und den sonstigen Betriebssystemkomponenten (Userland).
Den Aufbau eines Linux-Betriebssystems verdeutlicht Abbildung 12 und wird im Folgenden genauer erklärt.

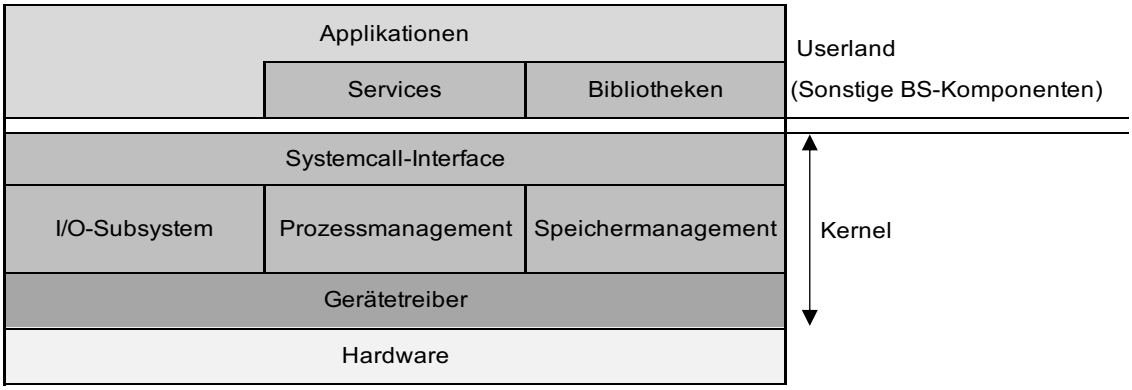

Abbildung 12: Aufbau des Betriebssystems in Anlehnung an [33]

Der Kernel hat unter anderem die Aufgabe, die Abarbeitung von Applikationen zu koordinieren und dabei Betriebsmittel wie Prozesse, Speicher und sonstige Hardware zu verwalten und diesen den Applikationen aus dem Userland durch Dienste zur Verfügung zu stellen. [33]

Der Zugriff auf Hardware und Geräte erfolgt innerhalb des Kernels durch Treiber. Diese Gerätetreiber sind im Kernel und enthalten Funktionen, die dem Betriebssystem ermöglichen, Zugriff auf Hardware-Geräte zu erhalten.
Somit stellen Treiber eine zusätzliche Schicht des Kernels dar, damit sie, je nach verwendeter Hardware, einfach austauschbar sind. [33]

2.4.2. Speicherbereich und Abarbeitungskontext

Programmcode kann in verschiedenen Speicherbereichen und Umgebungen ablaufen. Diese werden im folgenden Abschnitt vorgestellt.

Speicherbereiche

Die Linux-Speicherverwaltung trennt die Speicherbereiche in Linux in zwei Speicherbereiche, den User- und den Kernel Space [33].

User Space

Der Speicherbereich, welcher von Applikationen genutzt wird, wird als User Space bezeichnet. Eine Routine innerhalb des User Spaces hat keinen Zugriff auf den Kernel Space, auch nicht mit Superuser-Rechten. [33] [34]

Kernel Space

Der Speicherbereich, welcher vom Kernel genutzt wird, wird als Kernel Space bezeichnet. Zur Verdeutlichung: Ausschließlich eine Routine, welche innerhalb des Kernels ausgeführt wird, hat direkten Zugriff auf den Kernel Space.
Auch kann vom Kernel Space nicht direkt auf den User Space zugegriffen werden. Ein direkter Zugriff, auch über Zeiger auf den jeweils anderen Speicherbereich, ist grundsätzlich nicht möglich.
Nur durch das Aufrufen von bestimmten Funktionen (bspw. *copy_from_user*) oder das Verwenden von Mmap (siehe Kapitel 5.5.1) wird der Zugriff vom Kernel Space auf den User Space ermöglicht. [33] [34]

Ausführungskontext

Außerdem bestimmen die Umgebungen (auch als Kontext bezeichnet), aus welchem ein Programmcode aufgerufen wird, auf welche Dienste und Ressourcen dieser Zugriff hat [33].

Userkontext

Ein im Userkontext, auch als Applikationsebene bezeichnet, aufgerufener Programmcode hat Zugriff auf alle Dienste, die das Betriebssystem zur Verfügung stellt. Routinen in diesem Kontext können jederzeit durch Software- oder Hardware-Interrupts unterbrochen werden. [33]

Kernelkontext

Als Kernelkontext werden jegliche Routinen bezeichnet, die im Betriebssystemkern enthalten sind. Der Kernel-Kontext wird zusätzlich in einen Prozess- und einen Interrupt-Kontext unterteilt. [33]

Prozesskontext

Prozesskontext-bezogene Programmroutinen werden entweder durch einen zuvor von einer User-Applikation aufgerufenen Software-Interrupt oder einen Systemcall aufgerufen. In diesem Kontext stehen alle Funktionalitäten des Betriebssystems zur Verfügung. Auch ein Austausch mit der gebundenen Applikation ist möglich. Ungebundene Kontexte wie bspw. Kernel-Threads können hingegen keine Daten zwischen dem User- und dem Kernel Space austauschen.
Kernelcode, welcher im Prozesskontext ausgeführt wird, ist unterbrechbar und es besteht auch die Möglichkeit, die Verarbeitung des Codes für einige Zeit anzuhalten und diesen schlafen zu legen. [33] [34]

Interruptkontext

Programmroutinen, welche im Interruptkontext (im Englischen als atomic context bezeichnet) ausgeführt werden, werden stets zusammenhängend, das heißt ohne Unterbrechungen, ausgeführt, außer sie werden durch andere, höherwertigere Interrupts unterbrochen.
Dadurch gibt es jedoch Folgendes zu beachten:

- Die Verarbeitung dieses Programmcodes ist möglichst kurz zu halten, damit reguläre Rechenprozesse nicht unnötig verlangsamt werden.
- Außerdem dürfen diese Routinen weder sich selbst schlafen legen (passives Warten), noch dürfen sie Funktionen (wie z.B. kmalloc()) aufrufen, die sich selbst ggf. schlafen legen.

Typische Routinen, die im Interruptkontext aufgerufen werden, sind Interrupt Service Routinen (ISR) sowie Softirqs (z.B. Timer-Callbacks). [33] [34]

2.5. ALSA - Advanced Linux Sound Architecture

ALSA steht für Advanced Linux Sound Architecture und ist eine moderne Soundarchitektur für Linux und seit der Kernelversion 2.6 offiziell als Standard im Kernel integriert.
Sie ermöglicht unter anderem Applikationen aus dem Userland flexiblen (im Sinne von frei wählbaren Parametern wie Abtastrate etc.) und einfachen Zugriff auf Audiogeräte wie bspw. Soundkarten. Die genauen Implementierungsdetails

der jeweiligen Hardware bleiben der Applikation verborgen, denn es stehen vordefinierte Schnittstellen zur Verfügung. [35]

2.5.1. ALSA-Aufbau

Die ALSA-Bibliothek ist in sieben verschiedene Schnittstellen unterteilt [36]. Die bei der Wiedergabe/Aufnahme verwendete Schnittstelle ist die PCM-Bibliothek.

Bei der Wiedergabe/Aufnahme wird der ALSA-Gerätetreiber nicht direkt durch die Anwendungsapplikation aufgerufen, sondern durch die ALSA-PCM-Library. Somit findet der Kontextwechsel vom User- zum Kernel-Kontext mit Aufrufen des ALSA-Treibers aus der PCM-Bibliothek statt. Dies verdeutlicht folgende Abbildung. [37]

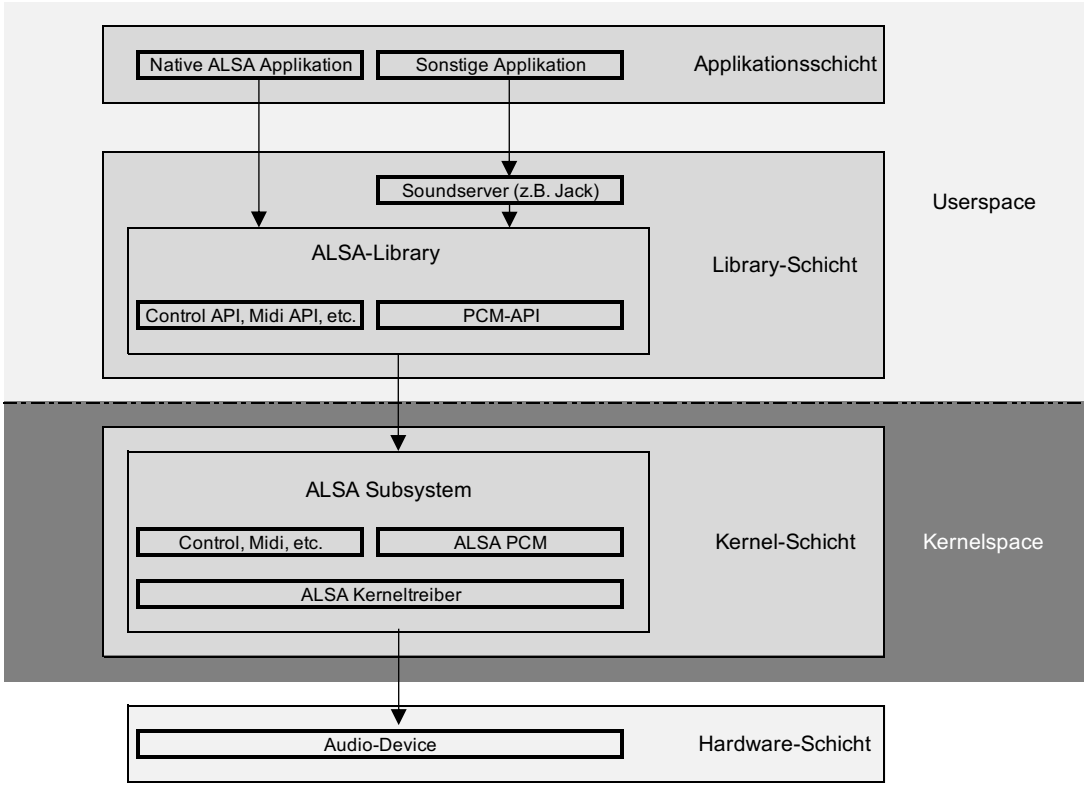

Abbildung 13: Aufbau des ALSA-Systems [Eigene Abbildung in Anlehnung an [38]]

Je nachdem, ob die aufrufende Applikation die ALSA-Schnittstelle direkt implementiert oder einen Soundserver wie Jack oder PulseAudio aufruft (welcher u.a. das parallele Abspielen aus mehreren Applikationen ermöglicht), wird die entsprechende ALSA-Library entweder direkt von der Applikation oder vom Soundserver aus aufgerufen.
Anschließend ruft die jeweilige ALSA-Library das entsprechende ALSA-Subsystem auf, in diesem Fall also den entsprechenden ALSA-PCM-Treiber. [37]

Die Funktionsweise der ALSA-Anwendungsbibliothek (PCM) sowie der genaue Ablauf einer Wiedergabe bzw. Aufnahme werden im nachfolgenden Abschnitt erläutert.

2.5.2. ALSA-Anwendungsbibliothek (PCM)

Hauptbestandteil der ALSA-PCM-Bibliothek ist ein Ringpuffer, welcher zur Bereitstellung oder Übertragung der Audiosamples dient [39]. Dieser Ringpuffer dient als Schnittstelle zwischen der Audioapplikation im User Space und dem Soundkartentreiber im Kernel Space.

Aufbau des PCM-Ringpuffers

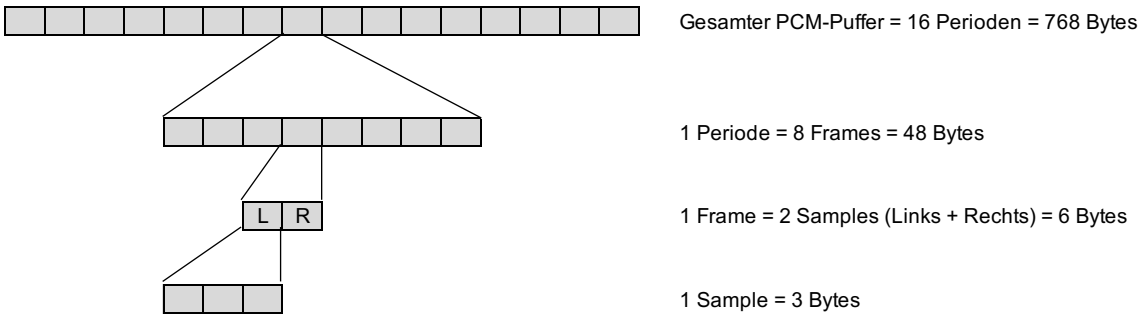

Abbildung 14: Aufbau des PCM-Ringpuffers [Eigene Abbildung in Anlehnung an [40]]

Wie die obenstehende Abbildung verdeutlicht, enthält der Ringpuffer eine Reihe von Samples, wobei in der ALSA-Terminologie die Zusammenfassung eines Samples aller Kanäle als ein Frame und ein Vielfaches von Frames als eine Periode bezeichnet wird. Der Zugriff auf den PCM-Puffer erfolgt immer periodenweise. [41]

Gerätezustände während der Wiedergabe/Aufnahme

Sowohl bei der Wiedergabe als auch bei der Aufnahme durchläuft ein PCM-Gerät mehrere Zustände. Ein PCM-Gerät befindet sich somit stets in einem definierten Status. [41] Eine Liste aller Status findet sich unter [3].

[3] ALSA project - the C library reference, PCM Interface, PCM state, online unter: https://www.alsa-project.org/alsa-doc/alsa-lib/group___p_c_m.html#ga61ac499cb3701ce536d4d83725908860.

Die folgende Abbildung zeigt ein vereinfachtes Zustandsübergangsdiagramm der ALSA-PCM-Bibliothek.

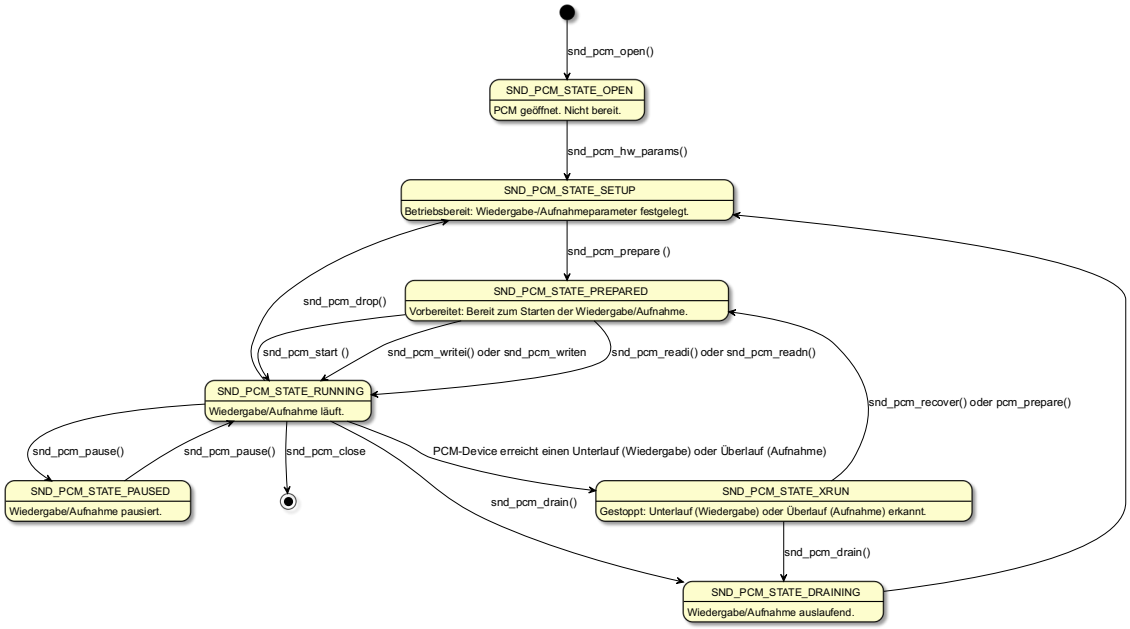

Abbildung 15: Zustandsübergangsdiagramm der ALSA-PCM-Bibliothek [Eigene Darstellung]

Zu Beginn wird das Audiogerät von der Userland-Applikation geöffnet.
Nach dem diese die Eigenschaften wie Sample-Format oder Größe des PCM-Puffers festgelegt hat und diese auch vom Audiogerät unterstützt werden, befindet es sich anschließend im Bereitschaftsmodus (*Setup*).
Soll nun eine Wiedergabe/Aufnahme in Kürze beginnen, wird das Audiogerät für die anstehende Aufgabe vorbereitet und der Betrieb kann aufgenommen werden (*Prepared*). Jetzt können auf dem ALSA-PCM-Ringpuffer Samples gelesen bzw. geschrieben werden und die Aufnahme/Wiedergabe läuft (*Running*). Sollte ein Unterlauf oder Überlauf auftreten, wird dies durch den entsprechenden Zustand (*Xrun*) kenntlich gemacht. Soll der Betrieb dabei nicht abgebrochen werden, muss das Audiogerät zuvor erneut vorbereitet werden (*Prepared*), um danach wieder in den Wiedergabe-/Aufnahmemodus (*Running*) zu wechseln.
Soll die Aufgabe/Wiedergabe beendet werden, kann die Applikation entscheiden, ob der Betrieb sofort unterbrochen werden soll und damit die im PCM-Puffer befindlichen Samples verworfen werden sollen, oder ob diese noch verarbeitet (*Draining*) werden sollen, bevor der Betrieb gestoppt wird. [41]

3. Anforderungsanalyse

3.1. Anforderungen des AES67-Standards

Die folgende Tabelle ist eine Zusammenfassung von Anforderungen für Geräte, die den AES67-2015-Standard adaptieren sollen. Im nachfolgenden Abschnitt wird auf die einzelnen Anforderungen genauer eingegangen.

Lfd Nummer	Kurze Beschreibung	Beschreibung	Verweis auf weitere technische Spezifikation
Allgemeine Anforderungen			
1	Latenz < 10 ms	Erreichen einer Latenzzeit zwischen Sender und Empfänger geringer als 10 ms.	
Synchronisation und Medientakt			
2	Synchronisierung über PTP	Alle Teilnehmer im Netzwerk sind in der Lage, die Wiedergabe und Aufnahme mit einer gemeinsamen Uhr synchronisieren zu können. Diese Synchronisierung sollte über Precision Time Protocol (PTP) erfolgen.	IEEE 1588-2008 (Precision Time Protocol)
3	Ableiten eines Medientaktes	Geräte sind in der Lage, aus der PTP-synchroniserten Netzwerkzeit einen Medientakt abzuleiten. Dabei muss der Medientakt mit der Abtastfrequenz übereinstimmen.	
Netzwerk(Transport-)anforderungen (OSI-Schicht 3+4)			
4	Verwenden des IPv4	Media-Pakete sollten über das IPv4-Protokoll übertragen werden. IP-Pakete sollten, soweit möglich, nicht fragmentiert werden.	RFC 791 (Internet Protocol)
5	Verwenden des UDP	Geräte verwenden zum Transport des Medienstreams das verbindungslose Protokoll UDP.	RFC 768 (User Datagram Protocol) sowie RFC 3551 (RTP Profile for Audio and Video Conferences with Minimal Control)
6	Unterstützen von Multicasting	Geräte können Streams über Multicasting übertragen und empfangen.	RFC 1112 (Host Extensions for IP Multicasting)
7	Unterstützen von IGMP	Zum Managen von Multicast-Gruppen muss das Protokoll IGMPv2 unterstützt werden. IGMPv3 kann optional unterstützt werden.	RFC 2236 (IGMPv2) sowie RFC 3376 (IGMPv3)
8	Unterstützen des RTP	Geräte verwenden zum Übertragen von Media-Streams das Real-Time Transport Protocol.	RFC 3550 (RTP: A Transport Protocol for Real-Time Applications) sowie RFC 3551 (RTP Profile for Audio and Video Conferences with Minimal Control)
9	Unterstützen von QoS	Geräte nutzen QoS, um eine Bevorzugung von echtzeitrelevanten Datenströmen gegenüber anderen Daten zu erreichen.	RFC 2474 (Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers)

Tabelle 2: Anforderungen gemäß AES67 (Teil1) [Eigene Tabelle basiernd auf [3]]

Lfd Nummer	Kurze Beschreibung	Beschreibung	Verweis auf weitere technische Spezifikation
Encoding und Streaming			
10	PCM-Kodierung der Payload	Es sollten 16 Bit und 24 Bit Pulscodemodulation unterstützen werden.	RFC 3551 (RTP Profile for Audio and Video Conferences with Minimal Control) Abschnitt 4.5.11 sowie RFC 3190 (RTP Payload Format for 12-bit DAT Audio and 20- and 24-bit Linear Sampled Audio) Abschnitt 4
11	Abtastrate der Payload	Mindestens eine Unterstützung einer Abtastraterate von 48 kHz. Optional eine Unterstützung der Abtastraten 44,1 kHz und 96 kHz.	
12	Anstreben einer Packet Time von 1 ms	Eine Packet Time von 1ms ist ein idealer Wert, um eine größtmögliche Kompatiblität zwischen verschiedenen Geräten zu erreichen, und sollte daher angestrebt werden.	
13	Senden und Empfangen von Streams mit variabler Anzahl von Känalen	Sender sollten mindestens einen Stream mit 1 bis 8 Kanälen anbieten. Empfänger sollten den Empfang von Streams mit 1-8 Kanälen unterstützen.	
14	Empfangspufferung	Zur Kompensation von Jittern ist ein Empfangspuffer notwendig.	
15	Geringe Schwankung des Sendezeitpunkts des Senders	Der vom Sender regelmäßige Sendezeitpunkt des zu übertragenden Pakets sollte eine nicht zu große Abweichung erfahren.	
16	Nutzung von Session Description	Um wichtige Informationen bezüglich des entsprechenden Streams, wie bspw. das Kodierungsformat, zu übertragen, wird das Session Description Protocol genutzt.	RFC 4566 (Session Description Protocol)

Tabelle 3: Anforderungen gemäß AES67 (Teil2) [Eigene Tabelle basierend auf [3]]

1. Latenz <10ms

Im Rahmen dieser Norm steht diese Anforderung an erster Stelle. Eine geringe Latenzzeit wird durch den Einsatzzweck im professionellen Audiobereich begründet.

Allerdings muss bei paketorientierter Übertragung zwangsläufig eine Pufferung von Audiosamples erfolgen. Hierbei gilt es, einen ausgewogenen Kompromiss zwischen ausreichend dimensionierter Pufferung und akzeptabler Verzögerung zu finden. [3]

2. Synchronisierung über PTP

Der Standard sieht vor, dass alle Geräte, die Audiostreams empfangen und/oder aussenden, in der Lage sind, ihre eigene lokale Uhr (Slave Clock) mit einer

Hauptuhr (Masterclock) zu synchronisieren. Dies wird mittels des IEEE 1588-2008 Standard, auch als PTPv2 bezeichnet, realisiert. [3]
Eine exakte Zeit ist erforderlich, damit Streams auf allen Geräten zur exakt gleichen Zeit phasengleich ausgespielt werde können. Der Standard sieht eine Abweichung von nur ± einem halben Sample vor. Dadurch ist bei einer Abtastrate von 48 kHz eine Zeitgenauigkeit von ±10 µs erforderlich. [42]

3. Ableiten eines Medientaktes

Um eine Phasensynchronität über mehrere Geräte hinweg zu erreichen, ist es erforderlich, aus der synchronisierten lokalen Uhr (siehe vorheriger Punkt) einen Medientakt (auch Media Clock genannt) abzuleiten, der der gewählten Abtastrate entspricht. Dadurch wird erreicht, dass bei allen Geräten die Media Clock sowohl frequenz-, als auch phasengleich ist.
Den Zusammenhang zwischen den 3 verschiedenen Zeitgebern verdeutlicht folgende Abbildung:

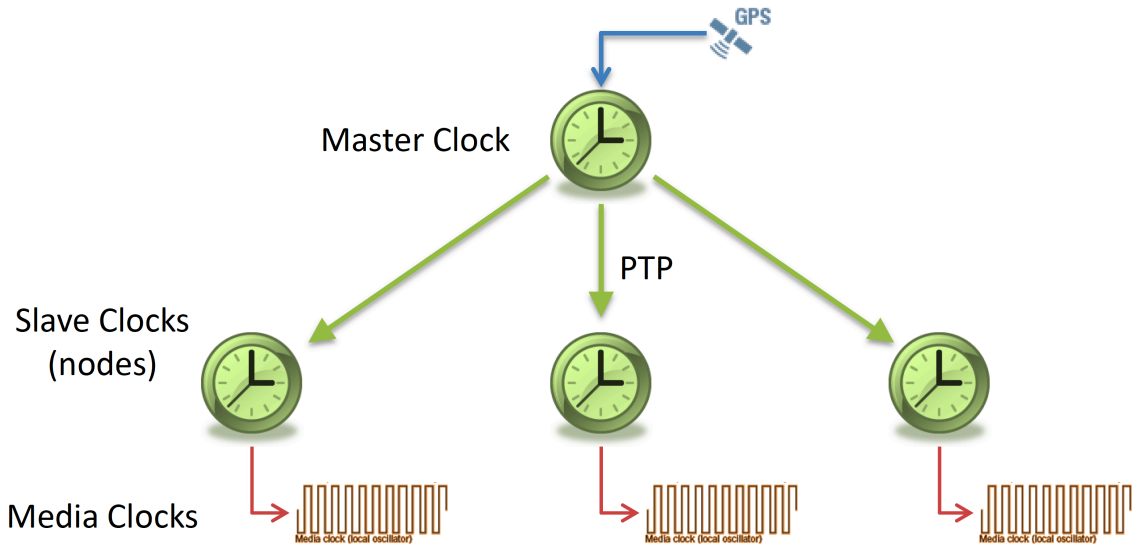

Abbildung 16: Zusammenspiel zwischen den Master-, Slave- und Media Clocks [2]

Die Media Clock wird typischerweise als eine 32-Bit Integer-Zahl ohne Vorzeichen gespeichert. Dieser Integerwert wird bei einer Abtastrate von 48 kHz pro Sekunde genau 48000-mal inkrementiert. Wie die folgende Rechnung zeigt, hat dies zur Folge, dass bei einer 32-Bit-Darstellung der Wert alle 24,86-Stunden überläuft.

Größtmöglich darstellbare Zahl $2^{32} - 1 = 4294967295$

$$\frac{4294967295}{48000 * 60\ sek * 60\ min} \approx 24{,}855\ std$$

Dieser Überlauf muss bei der Implementierung berücksichtigt werden.
Der Ursprung der Media Clock liegt der IEEE 1588 Epoche (1. Januar 1970 00:00:00 TAI) zu Grunde. [3]

4. Verwenden des IPv4

Unter Fragmentierung versteht man die Teilung von IP-Paketen in mehrere kleine IP-Pakete. Durch diesen Vorgang darf kein Verlust an Informationen entstehen. Obwohl die RFC 791 besagt, dass Empfänger in der Lage sein sollen, fragmentierte IP-Pakete wiederzusammenzusetzen, klammert die Norm diese Fähigkeit explizit aus. Sie müssen lediglich in der Lage sein, eine Fragmentierung zu erkennen und den Sender entsprechend darüber zu benachrichtigen.
Damit Pakete im Netzwerk nicht fragmentiert werden, kann im IP-Header ein Don't Fragment-Flag gesetzt werden. Durch Setzen dieses Flags zeigt der Absender an, dass es nicht gewünscht ist, das IP-Paket in mehreren Teilen zu übertragen.
Sollte es dennoch zu einer Fragmentierung kommen, ist dies dem Sender anzuzeigen und dieser sollte daraufhin die Übertragung einstellen und ggf. mit einer anderen Packet-Size fortsetzen. [3]

5. Verwenden des UDP

Die Übertragung der Medienpakete sollte mithilfe des User Datagram Protocols erfolgen. Durch die große anfallende Datenmenge bei der Übertragung von unkomprimierten Audiosignalen über das Netzwerk ist eine hohe Nutzlast vorteilhaft. Durch die Wahl des UDP entsteht im Vergleich zum TCP ein geringerer Overhead. Außerdem ist nur mit UDP ein Streaming nach dem Multicasting-Verfahren möglich. [3]

6. Unterstützung von Multicasting

Die Verwendung von Multicasting hat den Vorteil, dass der Empfang eines Streams von mehreren Geräten möglich ist, ohne dass sich die Bandbreite mit

der Zahl der Empfänger erhöht. Obwohl nach RFC 1112 vorgesehen, schließt AES67 das parallele Senden explizit aus.
Je Multicast-Adresse darf nur ein Sender zurzeit übertragen. Dafür müssen Sender vor dem Übertragen in eine Multicastgruppe sicherstellen, dass diese noch nicht belegt ist, indem der Sender zuvor dieser Multicastgruppe beitritt.
Abweichend zum RFC ist lediglich der IP-Bereich von 239.0.0.0 bis 239.255.255.255 vorgesehen. [3]

7. Unterstützen von IGMP

Zum Beitreten und Verlassen von Multicastgruppen ist IGMP ab der Version 2 zu unterstützen. Optional kann IGMPv3 verwendet werden. [3]

8. Unterstützen des RTP

Geräte müssen zum Übertragen von Media-Streams das Real-Time Transport Protocol verwenden.
Um sowohl Fragmentierung der Datenpakete zu verhindern als auch zukünftig Kompatibilität mit dem Standard IPv6 sicherzustellen, ist die maximal zulässige RTP-Nutzdatengröße auf 1440 Byte festgelegt.
Für Kommunikation über das RTP wird üblicherweise der Port 5004 verwendet.
Die Angabe des Formats und der Abtastrate, die für einen bestimmten Stream verwendet werden, erfolgt durch eine Kombination aus dem Feld Nutzlasttyp im RTP-Header (RFC 3550, Abschnitt 5.1) und den mit dem Stream verknüpften Beschreibungsinformationen. [3]

9. Unterstützen von QoS

Um eine QoS im Netzwerk zu implementieren, müssen die Geräte die Differentiated Services (DiffServ) implementieren.
Mithilfe von DiffServ ist es möglich, IP-Pakete zu klassifizieren. Damit können IP-Pakete mit einer höheren Priorität vorrangig zugestellt werden.
Im AES67-Standard sind mindestens die drei folgenden Verkehrsklassen zu unterstützen: [3]

- Clock (DiffServ-Klasse EF)
- Media (DiffServ-Klasse AF41)
- Best effort (DiffServ-Klasse DF)

10. PCM-Kodierung der Payload

Der Standard sieht vor, dass die einzelnen Audiosamples unkomprimiert als Pulscodemodulation übertragen werden sollen.
Zur Wahl steht die 16-Bit Codierung nach RFC 3551 Abschnitt 4.5.10. Diese sieht eine Zahlenrepräsentation als 16-Bit vorzeichenbehaftete Zahl in Big-Endian-Reihenfolge vor.
Des Weiteren kann die 24-Bit vorzeichenbehaftete Zahlenrepräsentation in Big-Endian-Reihenfolge, welche in der RFC 3190 im Abschnitt 4 beschrieben ist, verwendet werden. [3]

11. Abtastrate der Payload

Die folgenden Abtastraten können unterstützt werden:

- 44,1 kHz
- **48 kHz**
- 96 kHz

Dabei müssen alle Geräte mindestens die Abtastrate von 48 kHz unterstützen, die übrigen beiden können optional unterstützt werden. [3]

12. Anstreben einer Packet Time von 1 ms

Die Packet Time bezeichnet die Echtzeitdauer, der in einem IP-Packet enthalten Samples.
Eine Packet Time von einer Millisekunde ist als ein guter Kompromiss zwischen geringer Latenz, die beim Paketieren entsteht, und für das Netzwerk handelbares Paketaufkommen.
Je nach Abtastrate enthält ein IP-Paket somit 48 oder 96 Samples je Kanal. [3]

13. Senden und Empfangen von Streams mit einer variablen Anzahl von Kanälen

Die Anzahl der zu übertragenden Kanäle eines Streams ist variabel zu gestalten. Es sollten 1 – 8 Kanäle unterstützt werden. Eine höhere Anzahl von Kanälen ist möglich.

Es sollten nur Kanäle gebündelt werden, die auch verwandt sind. Als Beispiel sei hier das Bündeln von einem Stereo-Signal mit 2 Kanälen oder die Übertragung von Surround-Sound genannt. [3]

14. Empfangspufferung

Jeder Empfänger benötigt einen Empfangspuffer, um bei der Übertragung entstehende Jitter abfangen zu können. Als Jitter bezeichnet man in einem Netzwerk die Schwankung der Übertragungszeit [13].
Als Richtwert gibt die Norm vor, dass der Puffer mindestens dreimal so groß wie die Packet Time sein soll. Empfohlen wird die Verwendung eines Puffers, der mindestens 20-mal so groß wie die Packet Time ist bzw. mindestens 20 ms Samples zwischenspeichern kann. [3]

15. Geringe Schwankung des Sendezeitpunkts des Senders

Die Sender senden während der Übertragung einen kontinuierlichen Strom an IP-Paketen.
Dieser Datenstrom sollte eine durchschnittliche Abweichung von maximal 17 Paketen bzw. 17 ms nicht überschreiten. Optional können Werte wie 1 Paket bzw. 1 ms Packet Time Schwankung erreicht werden. [3]

16. Nutzung von Session Description

Sitzungsbeschreibungen (Session Descriptions) enthalten wichtige Informationen zu jedem Stream wie Netzwerkadressierung oder das Codierungsformat.
Dazu wird das Session Description Protocol (SDP) nach RFC 4566 verwendet. [3]

Unterstützen eines Discovery-Services

Discovery-Services können genutzt werden, um innerhalb des Netzwerks verfügbare Teilnehmer und Sitzungen bekannt zu machen. Dadurch ist es für den Nutzer einfacher, Verbindungen zu verwalten und bspw. einen Stream zu abonnieren, ohne eine manuelle Konfiguration der Session Description vorzunehmen.
Das Unterstützen von einem Discovery-Service sieht der Standard nicht explizit vor, jedoch werden Vorschläge zu möglichen Erkennungsdiensten gemacht.

Darin werden die Services Bonjour und Session Announcement Protocol (SAP) genannt. [3]

3.2. Anforderungen des Stakeholders

Wie in der Einleitung bereits erwähnt, stand der Bayerische Rundfunk (BR) als Projektpartner zur Ermittlung der Bedürfnisse zur Verfügung.
Der BR setzt in der Produktion sowohl für Hörfunk als auch Fernsehen eine Vielzahl von eigenentwickelten, maßgeschneiderten Lösungen auf Linuxbasis ein.
Um diese Lösungen auch zukünftig an das derzeitig entstehende Media-IP-System im neuen Funkhaus anbinden zu können, kam der Wunsch nach einem AES67-fähigen Audiotreiber für Linux auf.

3.2.1. Einsatzzweck

Als Einsatzzweck sieht der BR einerseits den Einsatz als Abhörsystem an Arbeitsplätzen vor. Andererseits ist die Anbindung an das vom BR eigens entwickelte OpenLogger vorgesehen.
Das Mittschnittsystem OpenLogger ist eine Eigenentwicklung des BR, welches automatisiert alle ausgespielten Sendungen aufzeichnet und für einen gewissen Zeitraum speichert, um den Anforderungen des gesetzlichen Mitschnitts gerecht zu werden.
Dieses System wird seit einigen Jahren beim BR und seit 2019 auch in anderen öffentlichen Rundfunkanstalten wie bspw. im Norddeutschen Rundfunk eingesetzt.

3.2.2. Technische Anforderungen

Der Einsatz ist vorwiegend auf eingebetteten Systemen vorgesehen. Diese werden von erfahrenen Technikern eingerichtet und gewartet.
Damit nimmt der Anwender in Bezug auf den zu entwickelnden Treiber selbst keine direkten Einstellungen vor.
Somit entfällt vorerst das Bedürfnis nach einer grafischen Benutzeroberfläche bzw. ist diese im Einsatz von eingebetteten Systemen sowieso nicht möglich.
Zusätzlich kam seitens des Bayerischen Rundfunks der Wunsch auf, möglichst viele Teile der Implementierung im Kernel Space zu erledigen.

Es sollte, soweit möglich, ein virtueller ALSA-Treiber implementiert werden, der in der Lage ist, insbesondere Multicast-Audiostreams zu empfangen und ggf. auch abzusetzen.
Aufgrund dieser Anforderungen erscheint eine Konfiguration der Parameter (IP-Adresse etc.) des Treibers durch die Linux-Shell sinnvoll.
Einerseits wird eine Konfiguration so auch auf eingebetteten Systemen ermöglicht, andererseits ist auch der automatisierte Zugriff aus Skripten oder anderen Programmen möglich, sodass weitere Erweiterungen seitens des BR durch nachfolgende Weiterentwicklung möglich sind.

Gegen das vollständige Verarbeiten einer SDP-Datei innerhalb des Treibers als Alternative zur Konfiguration der einzelnen Parameter spricht, dass ein Treiber nur so viel wie nötig im Kernel Space erledigen sollte. Daher ist die Nutzung und die Bereitstellung des Session Description Protocols eher dem User Space zuzuordnen und auch dort zu implementieren.

3.3. Auswertung der Anforderungsanalyse

Einen vollständig AES67-kompatiblen Treiber zu entwickeln, der alle oben genannten Anforderungen vollständig erfüllt, ist sehr aufwendig und im Rahmen dieser Arbeit nur begrenzt leistbar.
Insbesondere die Generierung des Medientaktes mit Bezug zur PTP-Synchronisierung stellt eine große Herausforderung dar.
Insofern ist dieser Treiber als ein Prototyp zu verstehen, der die passend für den Einsatzzweck notwendigen obenstehenden Anforderungen erfüllt.

3.3.1. Entwicklungsziele und Abgrenzung

Um einen ersten Ansatzpunkt geben zu können und zu evaluieren, ob ein solches Vorgehen sinnvoll ist, sind vorrangig die folgenden Anforderungen berücksichtigt worden:
Entwicklung eines zuverlässigen und stabilen ALSA-kompatiblen Treibers, der in der Lage ist, RTP-Streams zu abonnieren und diese mittels eines kommandozeilenbasierenden Audiorecorders wie *arecord* aufzuzeichnen.

Außerdem sollte eine Möglichkeit zur Konfiguration der streamspezifischen Parameter geschaffen werden, um einen flexiblen Einsatz zur Einbindung in Media-IP-Umgebungen zu ermöglichen.

Dabei wurde explizit die normgerechte Erzeugung des Medientaktes vorerst ausgeklammert. Stattdessen sollte untersucht werden, ob auch ein zuverlässiger Betrieb im syntonen Modus, das heißt, Synchronisierung auf den ankommenden Streams, möglich ist.

4. Systementwurf und Design

4.1. Architekturdesign des Treibers

Grundsätzlicher Aufbau des Treibers

Der Treiber lässt sich in die folgenden Bestandteile einteilen:

1. **Allgemeine Treiberarchitektur**: Implementierung aller notwendigen Callbacks für einen Plattform-Modultreiber
2. **Sysfs-Interface**: Konfigurationsschnittstelle des Treibers zum Userland
3. **ALSA-Treiberinterface**: Implementierung der ALSA-Callbacks sowie Timercallbacks
4. **Asynchrones Netzwerkinterface**: Ermöglicht das Senden und Empfangen von RTP-Paketen.

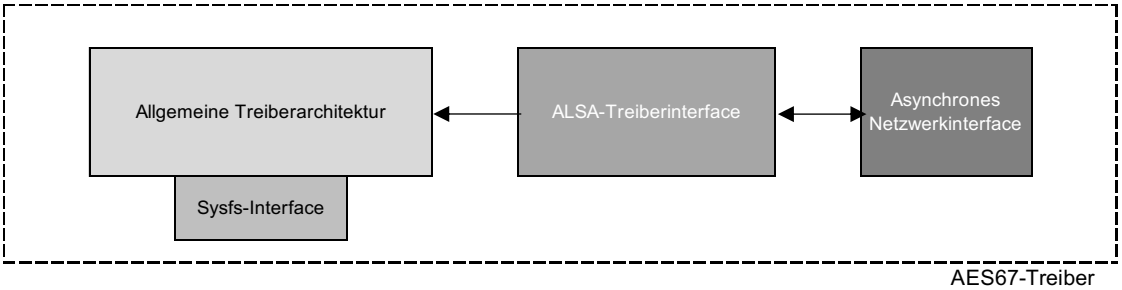

Abbildung 17: Aufbau des AES67-Treibers [Eigene Abbildung]

4.1.1. Allgemeine Treiberarchitektur

Stacked driver

Häufig bestehen Treiber aus mehreren Schichten, wie beispielsweise aus einer Low-Level-Schicht, welche direkt auf die Hardware zugreift und einer High-Level-Schicht, welche die Schnittstellen zu den Applikationen sicherstellt. Man spricht hierbei von sogenannten stacked driver [33]. Auch in der ALSA-Architektur ist dieser Aufbau vorgesehen.

Im Rahmen dieser Bachelorthesis ist ein virtueller Treiber, d.h. ein Treiber ohne physikalischen Hardwarezugriff, entwickelt worden. Daher entfällt ein Treiber auf der Low-Level-Schicht. Stattdessen ist ausschließlich ein High-Level-Treiber entwickelt worden.

Für die Entwicklung eines Low-Level-ALSA-Treibers gibt es ein eigenes ALSA-Framework. Weiterführende Informationen hierzu lassen sich unter dem Begriff ALSA System on Chip (ASoC) finden.

Plattformtreiber

Aufgrund des fehlenden dazugehörigen physikalischen Geräts wurde ein Plattformtreiber entwickelt. Solche Plattformtreiber zeichnen sich dadurch aus, dass es kein physikalisches Gerät gibt, welches durch das Betriebssystem erkennbar ist und dadurch das Laden eines Treibers automatisch initiiert werden könnte. Stattdessen wird das Anmelden des Plattform-Gerätes direkt nach dem Laden des Treibers vorgenommen. [43]

Modultreiber

Normalerweise sind Treiber integraler Bestandteil des Kernels. Sogenannte Kerneltreiber erfordern es, dass bei Austausch eines Treibers der gesamte Kernel neu kompiliert werden muss.
Als Alternative dazu können Treiber auch als ladbares Modul realisiert werden. Sie können entweder während der Laufzeit des Betriebssystems mit dem Befehl *insmod* geladen und mit dem Befehl *rmmod* entladen werden.
Außerdem werden sie sehr häufig bei Desktop-Distributionen eingesetzt, da somit nur eine Version ausgeliefert werden muss. Stattdessen werden sie automatisch während des Startvorgangs des Linux geladen und beim Herunterfahren wieder entladen. [33]

Dies ist besonders während der Entwicklung, aber auch in der späteren Verwendung praktisch, weshalb sich für einen Modultreiber entschieden wurde.
Nichtdestotrotz muss der Modultreiber zur verwendeten Kernelversion passen, da der Treiber auf Schnittstellen des Kernels aufbaut, die sich geändert haben könnten. [33]

4.1.2. Sysfs-Interface

Das Sys-Filesystem (Sysfs) ist eine Schnittstelle zwischen dem Anwender und dem Treiber. Es kann mittels der Bash unter dem Pfad /sys/ aufgerufen werden.

Das Sys-Filesystem ist ein virtuelles Filesystem, denn die Verzeichnisse und Dateien werden dynamisch erzeugt. In diesem können Informationen über Treiber abgerufen und Einstellungen des zugehörigen Geräts vorgenommen werden. [33]

Um den Treiber möglichst flexibel in den unterschiedlichsten Installationen verwenden zu können, lassen sich sowohl für den Transmitter als auch den Receiver gleichermaßen folgende Einstellungen über das Sysfs konfigurieren:

- IP-Adresse
- IP-Port
- Anzahl der enthaltenen Samples pro Kanal je Paket

Weitere Einstellungen wie die Samplerate und -frequenz werden direkt über das ALSA-Interface durch die entsprechende Audioapplikation vorgenommen.

4.1.3. ALSA-Treiberinterface

User-Applikationen, die auf ein ALSA-Gerät zugreifen wollen, rufen, wie in Kapitel 2.5.2. bereits beschrieben, die ALSA-PCM auf und erzeugen bei jedem Aufruf einen Software-Interrupt.
Anschließend wird auf der Kernelebene im zugehörigen Gerätetreiber der entsprechende ALSA-Callback vom ALSA-Framework aufgerufen und abgearbeitet.

ALSA-Callbacks

Eine Liste aller möglichen Callbacks sowie deren Erklärung ist unter [4] zu finden. Es können für die Aufnahme und das Abspielen unterschiedliche Callbacks definiert werden.
Damit ein ALSA-Treiber lauffähig ist, müssen mindestens die folgenden 8 Callbacks definiert sein:

- *open*
- *close*
- *ioctl*

[4] The Linux Kernel, Writing an ALSA Driver, PCM open callback, online unter: https://www.kernel.org/doc/html/v4.17/sound/kernel-api/writing-an-alsa-driver.html#operators.

- *hw_params*
- *hw_free*
- *prepare*
- *trigger*
- *pointer*

Die Callbacks *trigger* und *pointer* werden im Interrupt-Kontext ausgeführt. Die Funktionen und die genaue Implementierung der einzelnen Callbacks werden im Kapitel 5.5.1. vorgestellt.

Timercallbacks

Normalerweise ruft die physikalische Hardware wie bspw. ein Soundchip die Daten aus dem PCM-Puffer ab bzw. befüllt diesen und teilt anschließend dem ALSA-Framework durch ein Hardware-Interrupt mit, dass entweder im PCM-Puffer neue Samples bereitstehen (Wiedergabemodus) bzw. Samples aus dem PCM-Puffer konsumiert worden sind (Aufnahmemodus).
Dies ist in dem Treiber aufgrund der fehlenden physikalischen Hardware durch einen Timer ersetzt worden.
Der Timer hat im Wiedergabemodus die Aufgabe, die Daten regelmäßig aus dem PCM-Puffer abzurufen und über einen FIFO-Puffer an das Netzwerk-Interface zu übergeben. Im Aufnahmemodus werden die Daten aus einem FIFO-Puffer entnommen und im PCM-Puffer abgelegt. Im Anschluss erfolgt in beiden Fällen die Benachrichtigung des ALSA-Frameworks.
Die Funktionsweise des FIFO-Puffers wird im folgenden Abschnitt näher erläutert.

Da Timerinterrupts innerhalb eines atomaren Kontextes ausgeführt werden und die Callback-Routine deshalb möglichst kurz gehalten werden sollte, ist ein asynchrones Netzwerkinterface implementiert worden, welches die Audiosamples über das Netzwerk verschickt bzw. empfängt.

4.1.4. Asynchrones Netzwerkinterface

Das Netzwerkinterface dient dazu, die Übertragung bzw. den Empfang von Audiosamples, welche in Datenpakete verpackt über das Netzwerk übertragen werden, sicherzustellen.

Socket

In vielen Betriebssystemen wie auch in Linux dienen Sockets als Schnittstelle zur Netzwerkimplementierung [44].
Der Linux-Kernel-Netzwerk-Stack verwaltet analog zum ISO-OSI-Modell die Schicht 2 (Netzwerkgerätetreiber), Schicht 3 (Ipv4) sowie die Schicht 4 (UDP). Die Verwaltung darüberliegender Schichten ist Aufgabe der aufrufenden Applikationen [45].
Daher muss sich das Netzwerkinterface des Treibers ausschließlich um die Implementierung des RTP (Schicht 5) kümmern. Die darunterliegenden Protokolle werden durch den Netzwerk-Socket verwaltet.
Empfangene Datenpakete werden seit der Verwendung der New API im Socket gepuffert, sodass die Applikation diese von Zeit zu Zeit abrufen kann.
Konkret bedeutet das, dass die Verarbeitung der anliegenden Datenpakete im Netzwerksocket nicht allzu zeitkritisch ist, solange der Puffer nicht überschrieben wird [45].
Zu Beginn der Aufnahme/Wiedergabe muss der Socket stets mit den entsprechenden Daten initialisiert werden. Dies erfolgt in dem ALSA-Callback *Hw-params*.

Workqueue

Damit die Timercallback-Routinen möglichst schnell abgearbeitet werden können, ist das Netzwerkinterface nicht in diesen Callbackroutinen enthalten.
Stattdessen ist das Netzwerkinterface asynchron ausgeführt, was bedeutet, dass es in einem eigenständigen Thread parallel zu dem Kerneltreiber läuft.
Zur Verwaltung wurde eine Workqueue verwendet.
Eine Workqueue ermöglicht die sequenzielle Abarbeitung der in der Workqueue enthaltenen Aufgaben [33] .
In dieser wird, eigens für jede Senderichtung, die entsprechende Arbeit in die Schlange angehängt. Dadurch ist das parallele Nutzen der Aufnahme- und Wiedergabefunktion möglich.
Nachdem das Ende der Arbeit erreicht worden ist, das heißt, entweder es sind alle anstehenden Samples verschickt worden (Wiedergabemodus) oder es liegen keine weiteren Datenpakete im Socket-Buffer vor (Aufnahmemodus), wird

die Aufgabe mit einer Verzögerungszeit von wenigen Millisekunden erneut wieder in die Warteschlange aufgenommen.

FIFO-Puffer

Zum Austausch von Audiosamples zwischen dem Netzwerkinterface und dem ALSA-Timercallback dient ein FIFO-Puffer, in welchem, je nach Richtung, die empfangen Samples bzw. die zum Senden bereitstehenden Samples zwischengespeichert werden.

Diese Struktur ist an das Erzeuger-Verbraucher-Muster (Engl.: producer–consumer pattern) angelehnt. Dieses Entwurfsmuster ist ein Muster für Nebenläufigkeit und kann angewendet werden, um zwischen zwei oder mehreren voneinander unabhängigen Threads Daten austauschen zu lassen [46].
Dabei werden von einem Thread (Erzeuger) Daten erzeugt und anschließend in einem gemeinsamen Puffer gespeichert.
Der andere Thread (Verbraucher) liest den Puffer aus, der nach dem First In – First out (FIFO)-Prinzip arbeitet. Bei diesem Prinzip werden die Daten, die zuerst in diesem Puffer gespeichert worden sind, auch zuerst wieder ausgelesen. In der Regel werden hierfür sogenannte Ringpuffer eingesetzt.

Das Entwurfsmuster hat einerseits die Aufgabe, dafür zu sorgen, dass der gleichzeitige Zugriff auf den Puffer verhindert wird und damit Race Conditions vermieden werden.
Zum anderen stellt es sicher, dass die Daten in der korrekten Reihenfolge ausgelesen werden, indem die Daten nach dem FIFO-Prinzip zwischengespeichert werden.

4.2. Datenstruktur

Die folgende Abbildung zeigt den vereinfachten Aufbau der Datenstruktur des Treibers.

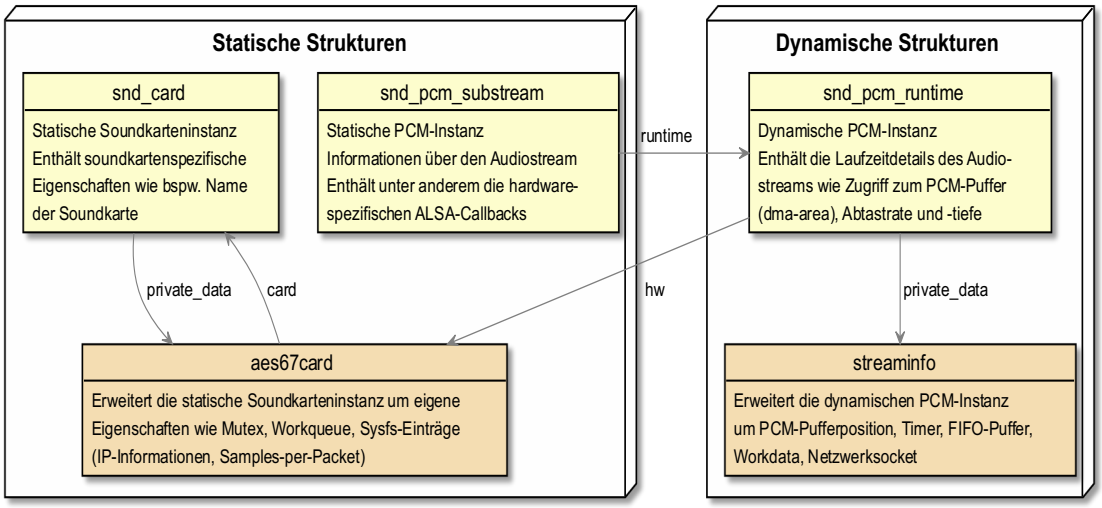

Abbildung 18: Vereinfachter Aufbau der Datenstruktur im Treiber [Eigene Abbildung]

Die Datenstruktur unterteilt sich in statische sowie dynamische Strukturen.
Die statischen Strukturen werden beim Erzeugen der Geräteinstanz erstellt und erst wieder mit deren Entfernen freigegeben.
Die dynamischen Datenstrukturen werden erst zum Beginn einer Wiedergabe/Aufnahme erzeugt und nach deren Beenden wieder freigegeben.

Dabei ist die statische Soundkarteninstanz (*snd_card*) um eine eigene Struktur erweitert (*aes67card*) worden, um eigene Funktionalitäten ermöglichen zu können.
Die nur während der Laufzeit benötigten Daten sind aus dieser Datenstruktur getrennt und in eine eigene Struktur (*streaminfo*) ausgelagert worden, welche die dynamische PCM-Instanz (*snd_pcm_runtime*) erweitert.
Da diese Datenstrukturen, wie oben bereits erläutert, erst zur Laufzeit erzeugt werden, kann hiermit der Speicherbedarf des Treibers in Bereitschaftsphasen reduziert werden.

Das nachfolgende Kapitel beschreibt nun konkret, welche Funktionen implementiert worden sind.

5. Entwicklung eines ALSA-Treibers

5.1. Allgemeines zur Treiberentwicklung

Da bei der Entwicklung eines Treibers einige Besonderheiten bei der Programmierung beachtet werden müssen, werden diese in diesem Abschnitt vorgestellt.

Linux Kernel API

Systembibliotheken aus dem User Space-Bereich wie die C-Standardbibliothek stehen im Kernel nicht zur Verfügung, da sie selbst Kernelfunktionen (Syscalls) aufrufen. Sie sind eigens für den Zugriff aus dem User Space entwickelt worden und können auch nur dort aufgerufen werden.
Stattdessen steht eine eigene Linux Kernel API[5] zur Verfügung, welche an die C-Bibliothek angelehnt ist, jedoch aus dem Kernel Space aufgerufen werden kann. [33]

Kein Speicherschutz

Außerdem ist zu beachten, dass Treiber vollen Zugriff auf den Kernel Space haben. Daher ist eine sorgsame Programmierung notwendig, da es ansonsten zur Kernelpanic und somit zum Systemabsturz führen kann. [33]

Begrenzte Stackgröße

Die Stackgröße von Threads im Kernel Space ist architekturabhängig. Sie beträgt meist etwa 4 bis 8 kByte [47]. Aufgrund der begrenzten Stack-Größe ist die Verwendung von rekursiven Funktionen nicht erlaubt.
Auch sollten größeren Datenbereich nicht vorab reserviert werden. Stattdessen sollte Speicher besser über kmalloc dynamisch allokiert werden, da dieser dann nicht auf dem Stack, sondern dem Heap liegt [48].
Dabei gilt es, wie auch schon bei der normalen Programmierung, darauf zu achten, dass allokierter Speicher stets nach dem Verwenden wieder freigegeben wird. Das gilt insbesondere auch dann, wenn der Treiber aufgrund eines

[5] The Linux Kernel API, online unter: https://www.kernel.org/doc/htmldocs/kernel-api/.

aufgetretenen Fehlers wieder geschlossen wird. Alle bis dahin allokierten Speicher müssen auch wieder freigegeben werden.

Vermeidung von Race Conditions

Race Condtions können auftreten, wenn aus mindestens zwei Prozessen der gleichzeitige Zugriff auf ein Betriebsmittel wie z.B. eine Variable stattfindet und dabei mindestens ein Prozess dieses verändert. Man spricht hierbei von zwei konkurrierenden Rechenprozessen.
Race Conditions können durch Mehrkernprozessoren, aber auch durch einen präemptiven Kernel, welcher die Unterbrechung von Prozessen auch im Kernelkontext ermöglicht, auftreten. [33]
Um Race Conditions zu vermeiden, muss sichergestellt werden, dass ein kritischer Abschnitt nur von einem Prozess zur Zeit betreten wird. Dazu lassen sich in Linux Gerätetreibern unter anderem folgende Synchronisations-Mechanismen verwenden: [33]

- Atomare Operationen
- Semaphore / Mutex
- Spin-Locks

Atomare Operationen

Durch atomare Operationen wird sichergestellt, dass das Auslesen der Variable, die anzuwendende Operation und das Zurückspeichern des neuen Wertes ohne Unterbrechung ausgeführt werden.
Es stehen diverse Operationen wie bspw. Inkrementieren, Addieren oder auch Bit-Operationen zur Verfügung. [33]
Für atomare Operationen spricht vor allem der geringe Aufwand in der Verwendung.

Semaphore und Mutex

Semaphore ermöglichen es, das gleichzeitige Betreten von kritischen Abschnitten auf eine definierte Anzahl von Prozessen zu begrenzen.
Bei Mutexes (Engl. Mutal exclusion) ist dieser Wert auf 1 festgelegt, sodass nur ein Prozess zur Zeit auf einen kritischen Abschnitt zugreifen kann. [33]
Bei der Verwendung von Mutexes gilt zu beachten:

- Vor der erstmaligen Verwendung eines Mutexes muss dieser initialisiert werden.
- Es ist darauf zu achten, dass der gesperrte Bereich stets wieder freigegeben wird. Häufig wird dazu bei auftretenden Fehlern mittels goto-Befehl zum Freigeben gesprungen. [33]
- Wenn auf einen gesperrten Bereich zugegriffen werden soll, gibt ein Mutex die CPU frei und legt den Prozess schlafen, solange dieser gesperrt ist und weckt diesen wieder, sobald der Bereich wieder frei ist. Daher dürfen Mutexes nicht im Interruptkontext (z.B. Timer (Softirq)) eingesetzt werden. Hier kann stattdessen ein Spin-Lock verwendet werden. [34]

Spin-Lock

Im Gegensatz zu Mutexes wird bei Spin-Locks auf eine Freigabe aktiv gewartet. Das heißt, dass der Prozess während der Wartezeit nicht freigegeben wird. Dies wird als sog. Spinning bezeichnet. [33]
Bei der Verwendung von Spin-Locks gilt zu beachten:

- Der kritische Abschnitt sollte so kurz wie möglich gehalten werden [33].
- Spin-Locks dürfen nicht verwendet werden, wenn der ausführende Code im kritischen Abschnitt den Prozess ggf. schlafen legen könnte [34].

5.2. Coding Style

An der Entwicklung des Linuxkernels sind sehr viele verschiedene Entwickler beteiligt, welche nicht alle einem gemeinsamen Unternehmen angehören.
Um dennoch einen einheitlichen Coding Style zu ermöglichen, wurde ein Linux Coding Style[6] entworfen, welcher auf dem „Kernighan und Ritchie (K&R)“- Stil, benannt nach den Autoren eines bekannten Fachbuches über die Programmiersprache C, basiert. Für Kommentare besteht ebenfalls ein Styleguide [7].

[6] The Linux Kernel, Linux kernel coding style, online unter: https://www.kernel.org/doc/html/v4.10/process/coding-style.html.
[7] The Linux Kernel, Writing kernel-doc comments, online unter: https://www.kernel.org/doc/html/v4.10/doc-guide/kernel-doc.html#writing-kernel-doc-comments.

5.3. Objektbasierte Programmierung

Fast alle Teile des Linuxkernels sind in der Programmiersprache C geschrieben. Zwar zählt diese Programmiersprache nicht zu den objektorientierten Sprachen. Nichtdestotrotz lassen sich mehrere Eigenschaften mittels des struct-Elements zu einem Objekt bündeln. Diese Eigenschaft wird auch in der Kernelprogrammierung dazu eingesetzt, um Objekte und Funktionen zusammenzufassen. Damit dienen Objekte zur Kapselung von Daten und Funktionalität.
Funktionen können durch Funktionszeiger an den Kernel übergeben werden.
Dadurch ist der Kernel in der Lage, die treibereigenen Funktionen aufrufen zu können. Erwartet der Kernel eine Funktionsübergabe, wird dies als Callback-Funktion bezeichnet.
Daten und Funktionen sind strikt zu trennen, was heißt, dass benötigte Daten innerhalb einer Funktion nicht in der Funktion selbst, sondern im zugehörigen Objekt gespeichert sind, welches der Funktion übergeben wird.
Dadurch soll erreicht werden, dass eine Funktion zustandslos ist. [33]

Objekte müssen vor Verwendung alloziiert und initialisiert werden. Anschließend können diese dem Kernel zur Verwendung übergegeben werden. Nach der Verwendung müssen diese wieder wie oben beschrieben deinitialisiert (freigegeben) werden. [33]

5.4. Plattform-Modultreiber

Bei der Entwicklung des ALSA-Treibers wurde sich zu Beginn hauptsächlich an der wissenschaftlichen Ausarbeitung von Smilen Dimitrov und Stefania Serafin orientiert [49]. In deren Ausarbeitung wurde ein virtueller Oszillator als ALSA-Kernelmodul entwickelt und dokumentiert.
Weitere Einstiegspunkte, welche bei der Entwicklung hilfreich waren: [8] [9]

[8] B. Collins, „Writing an ALSA driver“, Ben Collins, Apr. 30, 2010, online unter: http://ben-collins.blogspot.com/2010/04/writing-alsa-driver.html.
[9] I. Takashi, „Writing an ALSA Driver — The Linux Kernel documentation“, online unter: https://www.kernel.org/doc/html/v4.17/sound/kernel-api/writing-an-alsa-driver.html.

Um einen Modultreiber zu entwickeln, müssen die folgenden zwei Funktionen implementiert werden:

1. Der Treibereinstiegspunkt *module_init*, der beim Laden des Modultreibers aufgerufen wird.
2. Der Ausstiegspunkt *module_exit*, der beim Entfernen des Treibers aufgerufen wird.

Außerdem können optional Metainformationen wie Modulautor etc. festgelegt werden. Hierfür stehen vordefinierte Makros zur Verfügung.
Zwingend muss das Makro *MODULE_LICENSE* aufgerufen werden, um die zugrunde liegende Lizenz festzulegen.
Nur wenn ein Treibermodul unter der GNU Public License (GPL) steht, können sämtliche Funktionen des Kernels innerhalb des Treibers genutzt werden [33].

Kompilieren des Moduls

Anschließend muss ein *Makefile* erstellt werden, welches dem Compiler Anweisungen über den Kompilierungsprozess gibt. Ein zum Treiber passendes Makefile lässt sich im Anhang finden.
Anschließend lässt sich das Modul mit dem Befehl *make* kompilieren.

Laden und Entladen des Moduls

Nach erfolgreichem Kompilieren kann das Kernelmodul mit *insmod* geladen werden. Dabei wird die *modul_init*-Routine abgearbeitet.
Mittels *lsmod* lassen sich alle geladenen Module auflisten.
Mit dem Befehl *rmmod* kann ein geladenes Modul wieder entfernt werden. Dabei wird die *modul_exit*-Routine aufgerufen.

5.4.1. Konkrete Implementierung

Die konkrete Implementierung wird im folgenden Abschnitt erläutert. Die folgenden beiden Abbildungen verdeutlichen den Ablauf des Lade- und Entladevorgangs des Treibers.

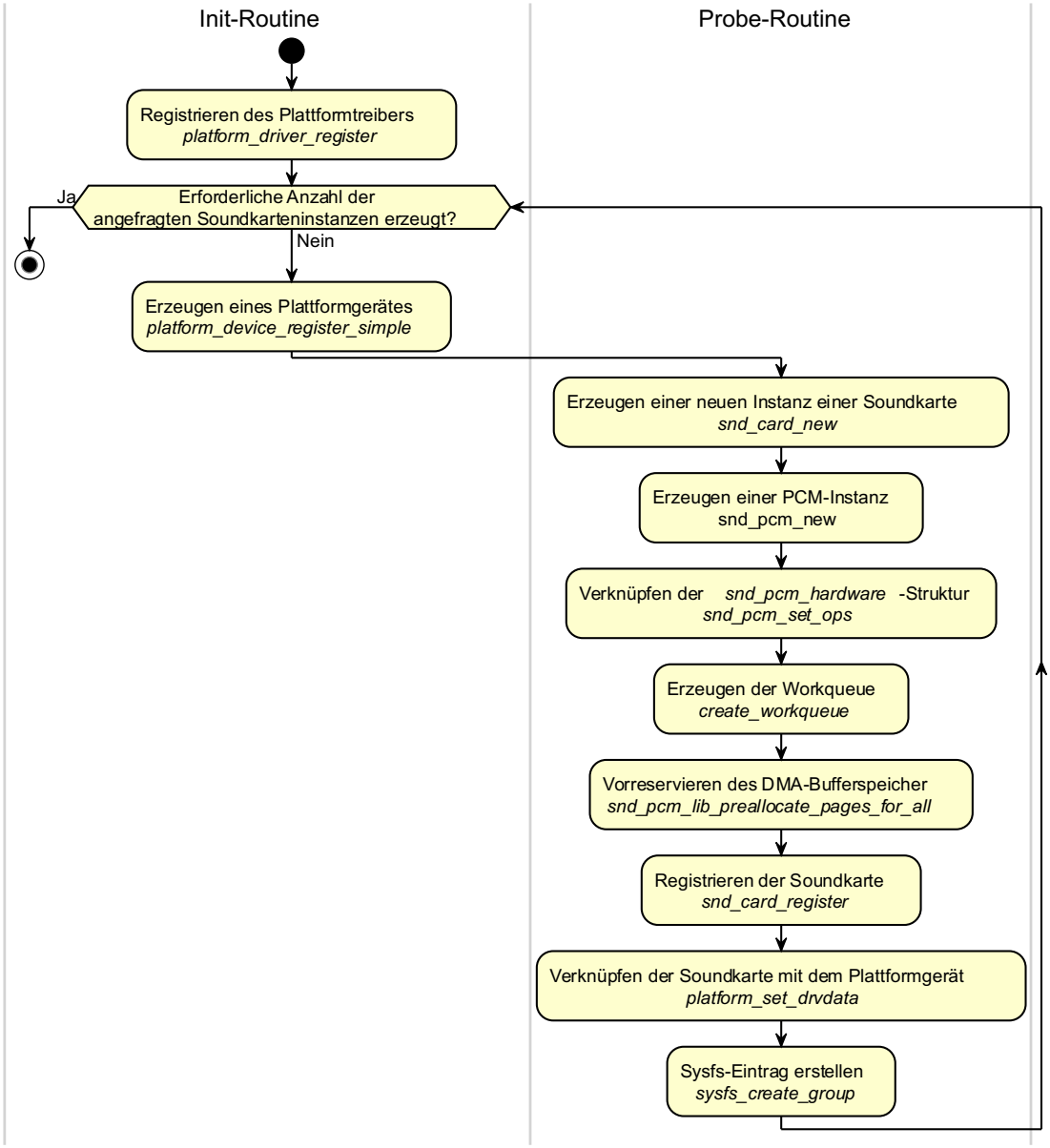

Abbildung 19: Vereinfachtes Ablaufdiagramm: Laden des Treibers [Eigene Darstellung]

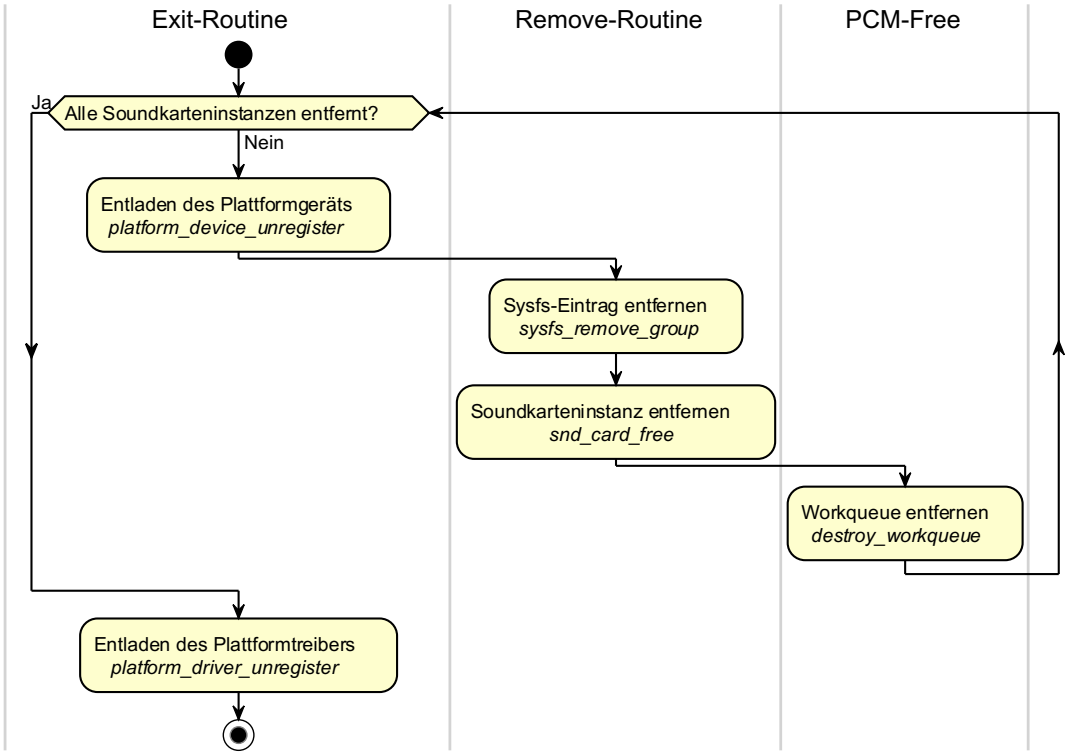

Abbildung 20: Vereinfachtes Ablaufdiagramm: Entladen des Treibers [Eigene Darstellung]

Der Modultreiber

Implementierung der Init-Routine des Modultreibers

Zuerst wird innerhalb der Init-Routine mittels des Befehls *platform_driver_register* das **Treibermodul** im Kernel registriert.
Hierbei wird als Argument eine platform_driver-Struktur übergeben, welche mittels eines Funktionszeigers auf die *probe*- und *remove*-Funktion des Plattformtreibers verweist sowie den Treibernamen bestimmt.

Aufgrund des Plattformtreibers muss innerhalb der *module_init* nicht nur das Treibermodul im Kernel registriert werden, sondern auch das Registrieren von **konkreten Geräteinstanzen** muss dort ausgelöst werden, indem die Funktion *platform_device_register_simple* aufgerufen wird.
Hierbei werden die folgenden Argumente übergeben:

- Name des Gerätes
- Instanz-ID
- Ggf. zusätzlich zu allokierender Speicher
- Anzahl der Ressourcen

Dabei muss der Name des Gerätes mit dem Treibernamen übereinstimmen, da hiermit die Zuordnung des dazugehörigen Treibers erfolgt [50].

Der endgültige Name des jeweiligen Gerätes setzt sich aus dem Namen und der Instanz-ID zusammen [50]. Verdeutlicht anhand eines Beispiels:
Der Name des Geräts „snd_aes67“ sowie die Instanz-ID „0“ wird zusammengesetzt zu „snd_aes67.0“.
Somit können auch mehrere Geräteinstanzen desselben Typs erzeugt werden, die sich dann durch ihre Instanz-ID unterscheiden. Bis zu 8 Soundkarteninstanzen können erzeugt werden.

Um alle registrierten Geräte in der Exitroutine wieder freigeben zu können, werden diese in einer *platform_device*-Arraystruktur gespeichert.

Implementierung Exit-Routine des Modultreibers

Die Exitroutine des Modultreibers entfernt in einer Schleife mit *platform_device_unregister* jede zuvor registrierte und in der *platform_device*-Arraystruktur gespeicherte Geräteinstanz. Hierdurch wird automatisch die Remove-Routine des Plattformtreibers aufgerufen.
Anschließend wird der Modultreiber selbst mit der Funktion *platform_driver_unregister* aus dem Kernel entfernt.

Eine Zusammenfassung über den Kompilierungs- sowie Ladevorgang des Treibers lässt sich im Anhang finden.

Der Plattformtreiber

Implementierung der Probe-Routine des Plattformtreibers und Initalisierung des Plattformtreibers

Die Probe-Routine des Plattformtreibers wird für jede zu erzeugende Geräteinstanz, genauer gesagt durch das Aufrufen von *platform_device_register_simple,* ausgeführt.
In der Probe-Funktion wird zunächst eine neue Instanz einer Soundkarte erzeugt. Hierbei wird auch der notwendige Speicher für die statische *aes67*-Struktur automatisch unter *private_data* alloziert.

Anschließend wird mit *snd_pcm_new* eine PCM-Instanz erzeugt.
Dort wird mit zweimaligen Aufrufen der Funktion *snd_pcm_set_ops jeweils* eine *snd_pcm_ops*-Struktur übergeben, welche durch Funktionszeiger zuerst auf die Wiedergabecallbacks, dann auf die Aufnahmecallbacks des ALSA-Treibers verweist.
Außerdem erfolgen noch einige Zuordnungen wie der Verweis auf die Destruktor-Funktion der PCM-Instanz und der Name der Soundkarte. Auch werden Initialisierungen der Workqueue sowie der notwendigen Mutexes vorgenommen.
Mit Aufrufen der Funktion *snd_pcm_lib_preallocate_pages_for_all* wird der PCM-Puffer vorbereitet und Speicher vorreserviert.

Zuletzt erfolgen die Registrierung der nun vollständig vorbereiteten Soundkarte im Betriebssystem mit der Funktion *snd_card_register* und die Verknüpfung der eben erstellten Soundkarte und der vom Betriebssystem erstellten Plattform-Geräteinstanz mit der Funktion *platform_set_drvdata*. Außerdem wird mit sysfs_create_group der entsprechende Eintrag im Sysfs hinzugefügt.

Implementierung der Remove-Routine des Plattformtreibers und Aufräumen des Plattformtreibers

Im Vergleich zur Probe-Routine fällt die Remove-Routine kurz aus. Sie wird ebenfalls für jede zu entfernende Geräteinstanz einzeln aufgerufen.
Es müssen lediglich die Einträge aus dem Sysfs mit Aufrufen der Funktion *sysfs_remove_group* entfernt werden.
Anschließend wird die Soundkarteninstanz mittels Aufrufes von *snd_card_free* entfernt. Dabei werden viele Destrukturfunktionen automatisch aufgerufen und müssen nicht explizit aufgerufen werden.
Durch die zuvor erfolgte Verknüpfung der Plattform-Geräteinstanz mit der Soundkarte wird somit auch die PCM-Destruktorfunktion automatisch aufgerufen.
Innerhalb dieser Destruktorfunktion wird die zuvor erzeugte Workqueue entfernt. Außerdem wird automatisch der Speicher von der statischen aes67card-Instanz (*private-Data*) freigegeben.

5.5. Implementierung des ALSA-Interfaces

Die nachfolgende Abbildung verdeutlicht das Zusammenspiel zwischen einer nativen ALSA-Userapplikation, dem ALSA-Subsystem und dem Treiber.

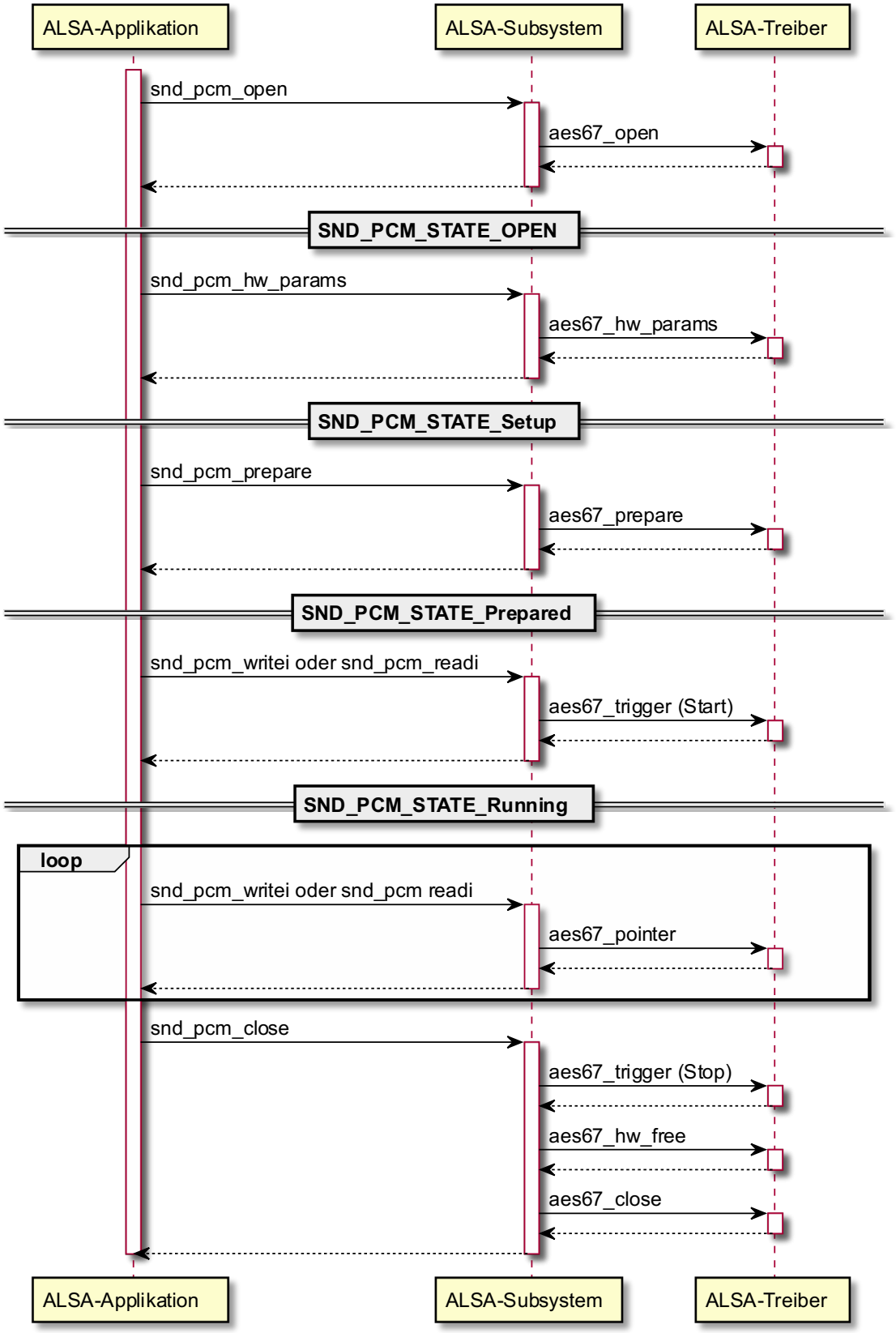

Abbildung 21: Sequenzdiagramm, welches den Ablauf einer Wiedergabe/Aufnahme zeigt [Eigene Darstellung]

Bevor bei einer Wiedergabe/Aufnahme die Übertragung von Audiosamples zwischen der ALSA-Applikation und dem Treiber möglich ist, werden zuvor, wie bereits im Kapitel 2.5.2 erläutert, mehrere Schritte durchlaufen.
Dazu ruft die Applikation Funktionen des Subsystems auf, wodurch daraufhin die Callbacks des Treibers aufgerufen werden.
Nach jedem Aufruf der Applikation erfolgt i.d.R. eine Zustandsänderung.
Der folgende Abschnitt erläutert nun die Implementierung der einzelnen Callbacks.

5.5.1. ALSA-Callbacks

Im folgenden Abschnitt werden die 8 zu definierenden ALSA-Callbacks beschrieben.
Wie oben bereits beschrieben, werden diese innerhalb der *Probe*-Routine mittels der *snd_pcm_ops*-Struktur über Funktionszeiger an das ALSA-Subsystem übergeben.

Alle Callbackfunktionen, die außerhalb des Interruptkontextes stehen (*open, close, ioctl, hw_params, hw_free und prepare*), werden mittels Mutexes vor Race Conditions geschützt.

Sowohl für die Wiedergabe als auch für die Aufnahme werden dieselben Callbackfunktionen aufgerufen, da zum größten Teil die gleichen Aktionen und Vorbereitungen sowohl bei der Wiedergabe als auch bei der Aufnahme erledigt werden müssen.

Open-Callback

Nachdem ein Programm aus dem User Space eine Wiedergabe/Aufnahme gestartet hat, erzeugt das ALSA-Subsystem zuerst eine PCM-Laufzeitinstanz (*snd_pcm_runtime*) und ruft die Open-Callback-Funktion auf.
Dort wird die eigene dynamische Datenstruktur (*Streaminfo*-Instanz) mittels *kzalloc* allokiert, wodurch der Speicherbereich auf 0 initialisiert und anschließend mit der PCM-Laufzeitinstanz verbunden wird (*private_data*).
Dabei wird auch ein Funktionszeiger der Destruktur-Funktion übergeben.

Ebenso wird die zuvor definierte statische Hardwarebeschreibung der PCM-Instanz zugeordnet, welche die Wiedergabe- und Aufnahmeeigenschaften festlegt. [51]

Aufnahme- und Wiedergabeeigenschaften definieren

Um die Wiedergabe- und Aufnahmeeigenschaften einer Soundkarte festzulegen, wird eine statische Struktur vom Typ *snd_pcm_hardware* erstellt. Es ist möglich, für Aufnahme und Wiedergabe unterschiedliche Eigenschaften zu deklarieren.
Viele Einstellungen werden über Define-Anweisungen aus der *asound.h* angegeben.
Die verschiedenen Einstellungsmöglichkeiten werden im Folgenden vorgestellt:

Info-Attribut

In dem Infofeld wird angegeben, wie die Samples in den PCM-Ringpuffer geschrieben werden.

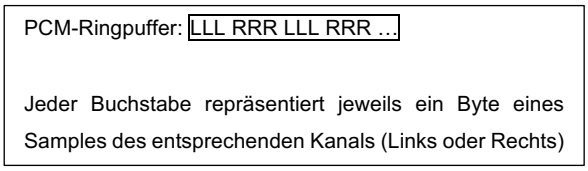

Abbildung 22: Darstellung des PCM-Ringpuffers nach dem Interleaving-Verfahren [Eigene Abbildung in Anlehnung an [52]]

In diesem Treiber werden die einzelnen Samples verschachtelt (Engl. Interleaving) übertragen. Dies verdeutlicht die Abb. 22.

Außerdem muss definiert werden, ob ein Zugriff auf den PCM-Ringpuffer über mmap (Memory Mapped) unterstützt wird.
Wie bereits in den Grundlagen erläutert, wird normalerweise der Speicherbereich des User Space und des Kernel Space strikt getrennt.
Es ist jedoch möglich, mittels mmap den Anwenderprogrammen direkten Zugriff auf den PCM-Ringpuffer zu geben, ohne dass dieser zuvor in den User Space-Speicherbereich kopiert werden muss.

Aus diesen beiden Optionen ergeben sich für Audioapplikationen insgesamt vier Zugriffsmöglichkeiten, welche unter [10] zu finden sind.
Außerdem können zusätzliche Eigenschaften wie eine Pause-Funktionalität hier definiert werden. Eine Liste aller möglichen Eigenschaften ist unter [11] zu finden.

Format-Attribut

In diesem Feld werden alle unterstützten Formate wie Abtasttiefe und Byte-Reihenfolge angegeben. Die Liste aller möglichen Formate ist unter [12] zu finden.
Nach der AES67-Norm werden die Abtastwerte in Big Endian-Reihenfolge übertragen [3].

Rate-Attribut sowie Rate_min und Rate_max

Die unterstützten Abtastraten werden in diesem Feld angegeben. Eine Liste aller möglichen Abtastraten ist unter [13] zu finden.
Die kleinst- und größtmögliche Abtastrate sollte mit den Angaben des *rate*-Attributs übereinstimmen. [51]

Channels_min und Channels_max

Hier wird die Anzahl der unterstützten Kanäle angegeben.

buffer_bytes_max

Die maximale Größe des PCM-Puffers wird hier angegeben.

period_bytes_min und period_bytes_max

Die Periodengröße gibt die Anzahl der Bytes an, die zwischen einem Interrupt übertragen wird. Kleine Perioden führen zu einer geringeren Verzögerungszeit

[10] ALSA project - the C library reference, PCM Interface, PCM access type, online unter: https://www.alsa-project.org/alsa-doc/alsa-lib/group___p_c_m.html#ga661221ba5e8f1d6eaf4ab8e2da57cc1a.
[11] Elixir Cross Referencer, Linux/include/uapi/sound/asound.h, online unter: https://elixir.bootlin.com/linux/v4.3/source/include/uapi/sound/asound.h#L254.
[12] ALSA project - the C library reference, PCM Interface, PCM sample format, online unter: https://www.alsa-project.org/alsa-doc/alsa-lib/group___p_c_m.html#gaa14b7f26877a812acbb39811364177f8.
[13] Elixir Cross Referencer, Linux/include/uapi/sound/asound.h, online unter: https://elixir.bootlin.com/linux/latest/source/include/sound/pcm.h#L108.

(Latenz), jedoch werden mehr Interrupts und dadurch mehr Prozessorlast erzeugt.

Close-Callback

Dieser Callback wird aufgerufen, wenn die Applikation die Soundkarte wieder freigibt. Durch die zuvor erfolgte Zuordnung des Destruktur-Funktionszeigers wird der Destruktur automatisch aufgerufen, der die entsprechende *Streaminfo*-Instanz freigibt.
Anschließend wird automatisch die PCM-Laufzeitinstanz (*snd_pcm_runtime*) wieder freigegeben.

Ioctl-Callback

Input-Output-Control sind Funktionsaufrufe mit einem fest definierten Befehlssatz.
Wenn, wie in diesem Treiber, keine Ioctl-Aufrufe abgefangen werden müssen, kann auf die vordefinierte ALSA-Funktion *snd_pcm_lib_ioctl* verwiesen werden.

Hw_params-Callback

Nachdem das Programm aus dem User Space eine Option aus den verfügbaren Hardware-Parameter ausgewählt und definiert hat, ruft das ALSA-Subsystem den *hw_params*-Callback auf.
In diesem Rückruf werden Initialisierungen und Vorbereitungen für die anstehende Wiedergabe/Aufnahme durchgeführt.

Initalisierung des Netzwerksockets

Es wird ein Netzwerksocket mit Aufrufen der *socket_init* initialisiert und vorbereitet.
Dabei wird festgelegt, dass das IPv4-Protokoll (*AF_INET*) verwendet werden soll und eine verbindungslose Kommunikation (*SOCK_DGRAM*) über UDP (*IPPROTO_UDP*) erfolgt.
Außerdem werden die IP-Informationen aus der statischen Karteninstanz *aes67card* übernommen.
Gehört die IP-Adresse dabei zum Multicast-Adressbereich, wird mittels *kernel_setsockopt*-Funktion der Beitritt zur Multicastgruppe ausgelöst.

Außerdem wird die anstehende Workqueue-Arbeit vorbereitet, indem eine Workdata-Instanz erzeugt wird. Diese enthält einen Verweis auf die Soundkarteninstanz sowie einen Worktype, welcher je nach Modus kenntlich macht, ob Daten empfangen (Aufnahmemodus) oder versendet werden sollen (Wiedergabemodus).
Ebenfalls wird der Timer vorbereitet und die FIFO-Struktur initialisiert.

Zum Schluss wird eine Funktion des ALSA-Subsystems aufgerufen, um den PCM-Puffer zu allokieren.

Prepare-Callback

Innerhalb des Prepare-Callbacks werden die PCM-Eigenschaften wie Abtastrate und -tiefe aus der *snd_pcm_runtime*-Struktur abgerufen.
Dadurch kann auch die Größe der Packetsize und somit auch die benötigte Größe des Netzwerkpuffers nach dieser Formel berechnet werden:

$$Nutzlastgr\ddot{o}\ss e = Abtastwerte\ pro\ Kanal\ \cdot Anzahl\ Kan\ddot{a}le\ \cdot \frac{Abtasttiefe}{8}$$

$$Paketgr\ddot{o}\ss e = Nutzlastgr\ddot{o}\ss e + UDP\_Header\_Gr\ddot{o}\ss e\ (12\ Byte)$$

Zu beachten ist, dass dieser Callback im Vergleich zum *Hw_params*-Callback nicht nur zu Beginn der Wiedergabe/Aufnahme, sondern auch nach einem ggf. aufgetretenen Unter- bzw. Überlauf aufgerufen wird.

Hw_free-Callback

Der Hw_free-Callback gibt alle zuvor im Hw_params-Callback sowie Prepare-Callback erzeugten Ressourcen wieder frei.
Der PCM-Puffer wird automatisch freigegeben.

Trigger-Callback

Der Trigger-Callback signalisiert über ein Kommando eine startende oder stoppende Wiedergabe/Aufnahme.
Mit Aufrufen dieses Callbacks werden der Timer und die Workqueue-Arbeit gestartet bzw. wieder angehalten.

Pointer-Callback

Der Pointer-Callback wird vom ALSA-Subsystem aufgerufen, um die aktuelle Position des PCM-Puffers anzufragen.
Zuvor wird innerhalb des Callbacks die Funktion zum Verarbeiten (Aufnahme) oder Bereitstellen (Wiedergabe) von Samples aufgerufen. Dies ist die gleiche Funktion, welche auch der Timer aufruft und wird im nachfolgenden Abschnitt beschrieben.
Anschließend gibt dieser Callback stets die aktuelle PCM-Pufferposition zurück.

5.5.2. Timercallbacks

Der Ablauf des Timercallbacks kann wie folgt beschrieben werden:

1. Berechnung der benötigten Frames im Bezug zur Zeitdifferenz zwischen dem jetzigen und dem vorherigen Aufruf.
2. Im Wiedergabemodus: Kopieren von Audiosamples des PCM-Puffers in den FIFO-Puffer.
   Im Aufnahmemodus: Kopieren von Audiosamples vom FIFO-Puffer in den PCM-Puffer.
3. Timerzeit berechnen und Timer erneut starten.
4. Benachrichtigung des ALSA-Subsystems, falls im PCM-Puffer mindestens eine Periode ausgelesen wurde bzw. neu zur Verfügung steht. Dies geschieht durch Aufrufen des PCM-Interrupt-Handlers *snd_pcm_period_elapsed*.

5.6. Implementierung des asynchronen Netzwerkinterface

Das asynchrone Netzwerkinterface wird durch eine Workqueue regelmäßig aufgerufen. Der Ablauf dieser Arbeit geschieht innerhalb des Interrupt-Kontextes und wird in der folgenden Grafik verdeutlicht.

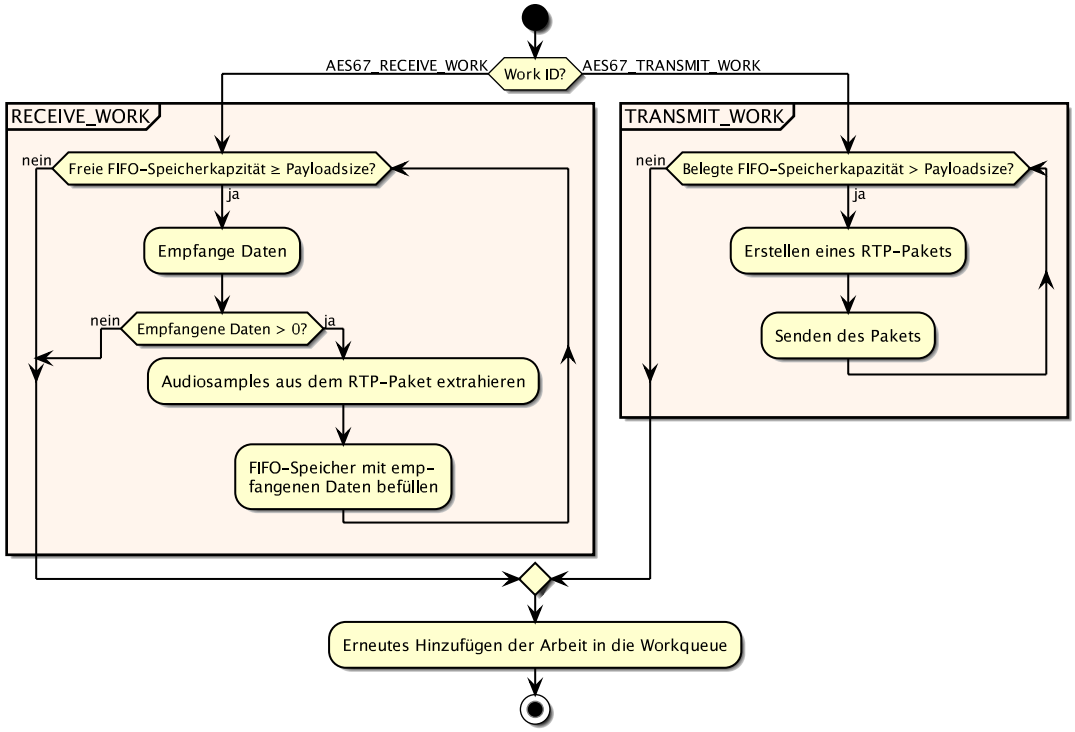

Abbildung 23: Vereinfachtes Ablaufdiagramm des Netzwerkinterfaces [Eigene Abbildung]

Zu Beginn des Callbacks wird die übergebene *work*-Struktur ausgelesen und die anfallende Arbeit anhand der Work ID zugeordnet.
Außerdem wird überprüft, ob das *Running*-Flag gesetzt ist. Ist dies nicht der Fall, ist die zugehörige Wiedergabe/Aufnahme beendet worden und das Netzwerkinterface ist zu beenden.

Receive-Work

Innerhalb der Receive-Work wird zunächst geprüft, ob der FIFO-Speicher ausreichend freien Speicherplatz enthält, um die Audiosamples aufnehmen zu können. Wenn diese Bedingung erfüllt ist, werden die Daten mittels des zuvor erzeugten Netzwerk-Sockets empfangen.
Durch das Setzen des *MSG_DONTWAIT*-Flags kehrt die aufgerufene Empfangsfunktion sofort zurück, selbst dann, wenn keine Pakete empfangen worden sind und der Puffer leer ist.

Wenn ein Datenpaket erfolgreich empfangen worden ist, müssen zunächst die Audiosamples (Payload) aus dem RTP-Paket extrahiert werden. Anschließend können diese dann in den FIFO-Speicher kopiert werden.
Dieser Ablauf wird so oft wiederholt, bis entweder der FIFO-Speicher voll ist oder keine Daten mehr im Socket-Puffer vorliegen.

Transmit-Work

Bei der Transmit-Work wird zunächst geprüft, ob im FIFO-Puffer genügend Audiosamples für die Übertragung vorliegen.
Sofern dies der Fall ist, werden diese aus dem Puffer ausgelesen, ein RTP-Header erstellt und zu einem RTP-Paket zusammengesetzt. Anschließend wird dieses Paket über das Netzwerksocket verschickt.
Dieser Vorgang wird solange wiederholt, bis nicht mehr genügend Audiosamples im FIFO-Speicher vorliegen, um ein RTP-Packet mit der entsprechenden Payload-Größe zu übertragen.

Nach Beendigung der jeweiligen Arbeit wird diese mit einer geringen Verzögerungszeit von einer Millisekunde erneut der Warteschlange hinzugefügt.

FIFO-Puffer

Für den FIFO-Puffer wird das *kfifo*-Interface aus der Linux Kernel API verwendet. Das Interface ist bei der Verwendung von nur einem Erzeuger und einem Verbraucher bereits synchronisiert und somit threadsicher. Daher sind keine weitere Vorgehensmaßnahmen bezügliche der Vermeidung von Race Conditions erforderlich.

Die Größe des FIFO-Puffers wurde so dimensioniert, sodass etwa 100 Millisekunden Audiosamples zwischengespeichert werden können.
Nach der Formel aus Kapitel 2.1. ergibt sich für 100 Millisekunden eine erforderliche Puffergröße von 28800 Bytes. Da die Größe des Puffers eine Potenz von 2 darstellen muss, wurde eine Größe von $2^{12} = 32768$ *Bytes* gewählt.

6. Ergebnisanalyse und Bewertung

6.1. Integrationstest

Um das Zusammenspiel des Treibers zusammen mit anderen AES67-kompatiblen Geräten zu testen, wurde ein Integrationstest im Labor des Norddeutschen Rundfunks (NDR) vorgenommen.

6.1.1. Testaufbau

Zum Einsatz kamen die folgenden Komponenten:

- Aes67-Treiber
- Ravenna COMi.MX bare
- Lawo Ruby Power Core Super Audio Node

6.1.2. Konfiguration

Im Lawo Ruby Power Core ist ein AES67-Stream an eine feste Multicast-Adresse eingerichtet worden. Als Paketlänge wurden hier 48 Samples pro Paket festgelegt.
Die RAVENNA COMi.MX hat eine Einstellung von 64 Samples pro Paket.
In beiden Systemen ist eine 24 bit PCM-Codierung mit einer 48 kHz Abtastrate verwendet worden.

6.1.3. Messergebnisse

Das Abonnieren und anschließende Aufzeichnen von Multicast-Streams mittels eines Aufnahmerekorders wie *arecord* oder *Kwave* ist ohne weitere Probleme durchführbar gewesen. Es ließen sich keine Aussetzer oder Ähnliches feststellen.
Dabei hat sich insbesondere die flexible Konfiguration der RTP-Payload als vorteilhaft erwiesen, da im Testaufbau selbst zwei verschiedene Einstellungen getestet worden sind.

Anders hingegen verhielt es sich mit dem Bereitstellen eines Medienstreams, welcher von anderen AES67-kompatiblen Geräten empfangen werden sollte.

Nach Eingabe der entsprechenden Konfigurationsdaten ließ sich zwar in der Streamübersicht des Ruby Power Core feststellen, dass eine entsprechende Übertragung des Streams stattfand. Doch aufgrund des abweichenden RTP-Zeitstempels konnte sich der AES67-Empfänger nicht auf den ankommenden Stream synchronisieren und somit wurde der Stream auch nicht ausgespielt. Damit ist der Einsatz des Treibers als AES67-Zuspieler in diesem Ausbaustadium derzeit nicht möglich.

6.2. Evaluation

Bezugnehmend zu den ermittelten Anforderungen an ein AES67-kompatibles Gerät (siehe Tabelle 2 und 3 in Kapitel 3.1.) sind folgende Anforderungen erfüllt worden:

Lfd Nummer	Kurze Beschreibung	Erfüllt?	Weitere Erläuterungen
Allgemeine Anforderungen			
1	Latenz < 10 ms	Ja	
Synchronisation und Medientakt			
2	Synchronisierung über PTP	Nein	
3	Ableiten eines Medientaktes	Nein	
Netzwerk(Transport-)anforderungen (OSI-Schicht 3+4)			
4	Verwenden des IPv4	Ja	Ist im Linux Netzwerk-Stack (Socket) implementiert.
5	Verwenden des UDP	Ja	Ist im Linux Netzwerk-Stack (Socket) implementiert.
6	Unterstützen von Multicasting	Ja	Ist im Linux Netzwerk-Stack (Socket) implementiert.
7	Unterstützen von IGMP	Ja	Wird durch Setzen von Socket-Options erreicht.
8	Unterstützen des RTP	Ja	Das Real-Time Transport Protocol wird innerhalb des Treibers implementiert.
9	Unterstützen von QoS	Nein	

Tabelle 4: Erfüllte Anforderungen gemäß AES67 (Teil 1)

Lfd Nummer	Kurze Beschreibung	Erfüllt?	Weitere Erläuterung
Encoding und Streaming			
10	PCM-Kodierung der Payload	Ja	Wird mittels der Flags *SNDRV_PCM_FMTBIT_S16_BE* sowie *SNDRV_PCM_FMTBIT_S24_BE* in der *snd_pcm_hardware* -Struktur sichergestellt.
11	Abtastrate der Payload	Ja	Die Abtastrate kann durch die Audioapplikation eingestellt werden.
12	Anstreben einer Packet Time von 1 ms	Ja	Die Packet Time kann über das Sysfs-Interface induviduell eingestellt werden.
13	Senden und Empfangen von Streams mit variabler Anzahl von Känalen	Teilweise	Es lassen sich Mono- und Stereostreams übertragen. Für den anvisierten Einsatzzweck derzeit ausreichend.
14	Empfangspufferung	Ja	Die Empfangspufferung wird mittels des ALSA-PCM-Puffers sichergestellt.
15	Geringe Schwankung des Sendezeitpunkts des Senders		Ließ sich aufgrund der nicht vollständigen Implementierung des Senders nicht überprüfen.
16	Nutzung von Session Description	Nein	Die Implementierung des Session Description Protocols kann in einer weiteren Entwicklung als User Space Applikation erfolgen.

Tabelle 5: Erfüllte Anforderungen gemäß AES67 (Teil 2)

Es hat sich gezeigt, dass sich der Treiber für den geplanten Einsatzzweck, dem Aufzeichnen von Sendestreams für den gesetzlichen Mitschnittservice, grundsätzlich einsetzen lässt. Die fehlende PTP-Synchronisation macht sich hierbei nicht bemerkbar, da die Ausspielung der Aufnahme naturgemäß sowieso zeitversetzt erfolgt. Auch die derzeit noch nicht implementierte Quality of Service (QoS)-Klassifizierung über DiffServ hat in der Aufnahmefunktion keine Relevanz.

7. Fazit und Ausblick

Es hat sich gezeigt, dass die Entwicklung eines nativen Audiotreibers, welcher ohne einen zugehörigen User Space Daemon auskommt, möglich ist.
Besondere Herausforderungen bei der Entwicklung des Treibers waren zum einen, dass umfassende technische Betriebssystem-Kenntnisse zur Treiberentwicklung notwendig sind. Scheduling-Fehler wie das Verwenden von *kmalloc* innerhalb eines Interrupt-Kontextes treten nicht zwangsläufig jedes Mal auf, was die Fehlersuche schwierig macht.
Außerdem stehen für die Entwicklung eines ALSA-Kerneltreibers nur wenige Dokumentationen zur Verfügung, was die Entwicklungszeit sehr verlängert hat.
Bei der Entwicklung war es hingegen von Vorteil, die Möglichkeit zu haben, bestehende AES67-Systeme zu testen und dabei die Kommunikation mittels *Wireshark* abzuhören.

Zu den nächsten Schritten gehört einerseits die Implementierung des *Precision Time Protocol (PTP)* im Kernel Treiber, welche erforderlich ist, um den Treiber als Sender einsetzen zu können.
Außerdem wäre eine User Space Applikation, welche die Verwaltung und das Verbindungsmanagement übernimmt, sinnvoll, indem sie das in der AES67-Norm geforderte *Session Description Protocol (SDP)* unterstützt.
Für die Anwenderfreundlichkeit wäre es außerdem wünschenswert, wenn diese Applikation die Unterstützung von mindestens einem Discovery-Protokoll wie das *Session Announcement Protocol (SAP)* oder das *Real-Time Streaming Protocol (RTSP)* unterstützt.

Anhang

Anleitung: Inbetriebnahme des Treibers

Makefile

Das Makefile sollte folgendes enthalten:

```
obj-m += snd-aes67.o
snd-aes67-objs  := aes67.o
all:
        make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd) modules
clean:
        make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd) clean
```

Es sollte sich zusammen mit den Dateien *aes.c* sowie *aes.h* in einem Ordner befinden.

Kompilieren und Laden des Treibers

```
make
sudo insmod snd-aes67.ko
```

Entladen des Treibers

```
sudo rmmod snd-aes67
```

Alle Soundkarten auflisten (Wiedergabe)

```
aplay -l
```

Die Ausgabe der Soundkarte sollte dann wie folgt aussehen:

```
**** Liste der Hardware-Geräte (PLAYBACK) ****
Karte 1: AES67Soundcard [AES67-Soundcard], Gerät 0: AES67 PCM [AES67 PCM]
  Sub-Geräte: 1/1
  Sub-Gerät #0: subdevice #0
```

Alle Soundkarten auflisten (Aufnahme)

```
arecord -l
```

Die Ausgabe der Soundkarte sollte dann wie folgt aussehen:

```
*** Liste der Hardware-Geräte (CAPTURE) ****
Karte 1: AES67Soundcard [AES67-Soundcard], Gerät 0: AES67 PCM [AES67 PCM]
  Sub-Geräte: 1/1
  Sub-Gerät #0: subdevice #0
```

Starten einer Wiedergabe bzw. Aufnahme

```
sudo aplay -D hw:1,0 -f S24_BE -r 48000 -c 2 -t raw soundfile.raw
```

```
sudo arecord -D hw:1,0 -f S24_BE -r 48000 -c 2 -t raw soundfile.raw
```

Zu definierende Parameter:

-D	Dieser Parameter ist an die zuvor mittels *aplay -l bzw. arecord -l* ermittelte Soundkarteninstanz anzupassen. Bsp.: *hw:1,0*
-f	Gibt das Abtastformat an. Zur Auswahl stehen S16_BE (16-Bit) sowie S24_BE (24-Bit).
-r	Gibt die Abtastrate an. Zur Auswahl stehen 44100 Hz sowie 48000 Hz
-c	Gibt die Anzahl der Kanäle an. Zur Auswahl stehen 1 oder 2 Kanäle
-t	Gibt den Typ der Audiodatei an. Hinweis: In dem gewählten Abtastformat (Big Endian) wird nur den Typ Raw unterstützt.

Literaturverzeichnis

[3] AUDIO ENGINEERING SOCIETY, INC., „AES67-2015: AES standard for audio applications of networks - High-performance streaming audio-over-IP interoperability“.

[7] M. Meyer, *Signalverarbeitung: Analoge und digitale Signale, Systeme und Filter*. Wiesbaden: Vieweg+Teubner, 2011.

[8] *Digitale Kommunikationssysteme 1 Grundlagen der Basisband-Übertragungstechnik*. Wiesbaden: Vieweg+Teubner Verlag, 2004.

[9] J.-R. Ohm und H. D. Lüke, *Signalübertragung: Grundlagen der digitalen und analogen Nachrichtenübertragungssysteme*, 11., neu Bearbeitete und Erweiterte Auflage. Berlin Heidelberg Dordrecht London New York: Springer, 2010.

[10] T. Görne, *Tontechnik: Hören, Schallwandler, Impulsantwort und Faltung, digitale Signale, Mehrkanaltechnik, tontechnische Praxis*, 4., aktualisierte Auflage. München: Hanser, 2015.

[11] *Meyers Großes Taschenlexikon in 24 Bänden: plus DVD-ROM*, 10., neu Bearb. u. erw. Aufl. Mannheim: Bibliogr. Inst, 2006.

[12] D. Stotz, *Computergestützte Audio- und Videotechnik: Multimediatechnik in der Anwendung*, 2. Aufl. Berlin: Springer, 2011.

[13] A. S. Tanenbaum, *Computernetzwerke*, 3., überarb. Aufl., [3. Nachdr.]. München: Pearson Studium, 2002.

[26] C. Benvenuti, *Understanding Linux network internals*. Sebastapol, CA: O’Reilly, 2006.

[33] J. Quade und E.-K. Kunst, *Linux-Treiber entwickeln: eine systematische Einführung in die Gerätetreiber- und Kernelprogrammierung - jetzt auch für Raspberry Pie*, 4., aktualisierte und erweiterte Auflage. Heidelberg: dpunkt.verlag, 2016.

[34] S. Venkateswaran, *Essential Linux device drivers*. Upper Saddle River, NJ: Prentice Hall, 2008.

[37] J. Newmarch, *Linux sound programming*. New York, NY: Apress, 2017.

[38] F. Meier, „Low-Latency Audio over IP on Embedded Systems“, Technische Universität Hamburg-Harburg, 2013.

[44] J. Wolf, *Linux-Unix-Programmierung: das umfassende Handbuch ; [Einstieg, Praxisbeispiele, Referenz ; System- und Netzwerkprogrammierung ; X Window, GTK+, SDL, Werkzeuge, Sicherheit ; inkl. Openbooks zu C, Unix und Debian]*, 2., aktualisierte und erw. Aufl., 1. Nachdr. Bonn: Galileo Computing, 2007.

[45] R. Rosen, *Linux Kernel Networking: implementation and theory*. New York, NY: Apress, 2014.

[46] J. Goll und M. Dausmann, *Architektur- und Entwurfsmuster der Softwaretechnik: mit lauffähigen Beispielen in Java*. Wiesbaden: Springer Vieweg, 2013.

[48] J. Wolf, *C von A bis Z: das umfassende Handbuch*, 3., aktualisierte und erweiterte Auflage 2009, 7. korrigierter Nachdruck 2019. Bonn: Rheinwerk Verlag, 2019.

[49] S. Dimitrov und S. Serafin, „Minivosc - a minimal virtual oscillator driver for ALSA (Advanced Linux Sound Architecture)“, *Proc. Linux Audio Conf. 2012*, S. 175–182, Apr. 2012.

Internetquellenverzeichnis

[1] „Audio over IP (AoIP) verständlich erklärt :: bonedo.de“, *bonedo.de - Das Musikerportal*. https://www.bonedo.de/artikel/einzelansicht/audio-over-ip-aoip-verstaendlich-erklaert.html (zugegriffen Aug. 06, 2020).

[2] A. Hildebrand, „Overview on IP Audio Networking“, University of Applied Sciences in Rüsselsheim, Jan. 22, 2018, [Online]. Verfügbar unter: https://www.hs-rm.de/fileadmin/persons/khofmann/Gastvortraege/Vortragsfolien/20180126-Hildebrand-IP_Audio_Standards.pdf.

[4] „AES Standard » AES67-2018: AES standard for audio applications of networks - High-performance streaming audio-over-IP interoperability“. https://www.aes.org/publications/standards/search.cfm?docID=96 (zugegriffen Aug. 06, 2020).

[5] von C. Bangert, „Was meinen Experten zur Zukunft der Audionetzwerke?“, *Professional System*, Okt. 13, 2019. https://www.professional-system.de/basics/audionetzwerke-wo-liegt-die-zukunft/ (zugegriffen Aug. 06, 2020).

[6] „AES67 Open Source Software Wishlist“, *Gist*. https://gist.github.com/njh/-c9196c465ea33ae9f97db782870464ef (zugegriffen Aug. 06, 2020).

[14] „Ethernet Frame Aufbau“. https://www.easy-network.de/ethernet-frame.html (zugegriffen Juli 12, 2020).

[15] „Was ist das Internet Protocol? – Definition von IP & Co.“, *IONOS Digitalguide*. https://www.ionos.de/digitalguide/server/knowhow/was-ist-das-internet-protocol-definition-von-ip-co/ (zugegriffen Juli 12, 2020).

[16] „Unicast“, *Wikipedia*. Sep. 07, 2018, Zugegriffen: Juli 13, 2020. [Online]. Verfügbar unter: https://de.wikipedia.org/w/index.php?title=Unicast&oldid-=180725064.

[17] „Multicast“, *Wikipedia*. Sep. 07, 2018, Zugegriffen: Juli 13, 2020. [Online]. Verfügbar unter: https://de.wikipedia.org/w/index.php?title=Unicast&old-id=180725064.

[18] „Broadcast“, *Wikipedia*. Juli 07, 2020, Zugegriffen: Juli 14, 2020. [Online]. Verfügbar unter: https://de.wikipedia.org/w/index.php?title=Multicast&old-id=201646232.

[19] B. Messer und C. Müller, „Wirtschaftsinformatik (Bachelor-Studiengang): Rechnernetze/Onlinedienste (2. Semester)“. http://www.it-infothek.de/-

wirtschaftsinformatik/semester-2/rechnernetze-und-onlinedienste-10.html (zugegriffen Juli 12, 2020).

[20] S. E. Deering, „Host extensions for IP multicasting“, *RFC 1112*, Aug. 1989. https://tools.ietf.org/html/rfc1112 (zugegriffen Juli 04, 2020).

[21] W. Fenner, „Internet Group Management Protocol, Version 2“, *RFC2236*, Nov. 1997. https://tools.ietf.org/html/rfc2236 (zugegriffen Juli 04, 2020).

[22] I. Kouvelas, B. Cain, B. Fenner, S. Deering, und A. Thyagarajan, „Internet Group Management Protocol, Version 3“, *RFC3376*, Okt. 2002. https://tools.ietf.org/html/rfc3376 (zugegriffen Juli 04, 2020).

[23] S. Luber und A. Donner, „Was ist IGMP (Internet Group Management Protocol)?“ https://www.ip-insider.de/was-ist-igmp-internet-group-management-protocol-a-905663/ (zugegriffen Aug. 07, 2020).

[24] W. C. Fenner, S. E. Deering, und B. Haberman, „Multicast Listener Discovery (MLD) for IPv6“, *RFC 2710*, Okt. 1999. https://tools.ietf.org/-html/rfc2710 (zugegriffen Juli 04, 2020).

[25] L. H. M. K. Costa und R. Vida, „Multicast Listener Discovery Version 2 (MLDv2) for IPv6“, *RFC 3810*. https://tools.ietf.org/html/rfc3810 (zugegriffen Juli 04, 2020).

[27] „UDP“. http://www.informatik.uni-hamburg.de/TKRN/world/lernmodule/L-Mint/Popup/UDP.htm (zugegriffen Juli 13, 2020).

[28] J. Postel, „User Datagram Protocol“, *RFC 768*. https://tools.ietf.org/-html/rfc768 (zugegriffen Aug. 07, 2020).

[29] V. Jacobson, R. Frederick, S. Casner, und H. Schulzrinne, „RTP: A Transport Protocol for Real-Time Applications“, *RFC 3550*. https://tools.ietf.org/html/rfc3550 (zugegriffen Aug. 07, 2020).

[30] D. Irwin und J. Slay, „Extracting Evidence Related to VoIP Calls“, *ResearchGate*. https://www.researchgate.net/figure/presents-the-RTP-packet-header-format-In-RTP-the-Synchronization-Source-SSRC-field_fig2_221352750 (zugegriffen Juli 13, 2020).

[31] H. Schulzrinne und S. Casner, „RTP Profile for Audio and Video Conferences with Minimal Control“, *RFC 3551*, Juli 2003. https://tools.ietf.org/-html/rfc3551 (zugegriffen Juli 14, 2020).

[32] D. Witberg, „RTP packet streaming - Hackerspace ACKspace“. https://ackspace.nl/wiki/RTP_packet_streaming (zugegriffen Juli 13,

2020).

[35] R. Rebe, „Alsa - Advanced Linux Sound Architecture“, *Linux-Magazin*. https://www.linux-magazin.de/ausgaben/2005/09/neue-klangarchitektur/ (zugegriffen Juni 07, 2020).

[36] „ALSA Library API - AlsaProject“. https://www.alsa-project.org/main/index.php/ALSA_Library_API (zugegriffen Juni 08, 2020).

[39] „ALSA project - the C library reference: PCM (digital audio) interface“. http://epic-alfa.kavli.tudelft.nl/share/doc/alsa-lib-devel-1.0.22/doxygen/html/pcm.html (zugegriffen Juni 08, 2020).

[40] J. Tranter, „Introduction to Sound Programming with ALSA | Linux Journal“. https://www.linuxjournal.com/article/6735 (zugegriffen Juni 08, 2020).

[41] „ALSA project - the C library reference: PCM (digital audio) interface“. https://www.alsa-project.org/alsa-doc/alsa-lib/pcm.html (zugegriffen Aug. 07, 2020).

[42] A. Hildebrand und D. Boldt, „RAVENNA Webinar ‚AES67 SMPTE ST 2110 Timing and Synchronization‘“, Juni 09, 2020, [Online]. Verfügbar unter: https://www.youtube.com/watch?v=XYxeL1uD0SQ.

[43] „git.kernel.org, Documentation/driver-model“. https://www.kernel.org/doc/-Documentation/driver-model/platform.txt (zugegriffen Juni 08, 2020).

[47] „4. Kernel Stacks — The Linux Kernel documentation“. https://www.kernel.org/doc/html/latest/x86/kernel-stacks.html (zugegriffen Juli 17, 2020).

[50] „Platform Devices and Drivers — The Linux Kernel documentation“. https://www.kernel.org/doc/html/latest/driver-api/driver-model/platform.html (zugegriffen Juli 28, 2020).

[51] I. Takashi, „Writing an ALSA Driver — The Linux Kernel documentation“. https://www.kernel.org/doc/html/v4.17/sound/kernel-api/writing-an-alsa-driver.html (zugegriffen Juni 08, 2020).

[52] „PCM Ring Buffer - AlsaProject Wiki“. https://www.alsa-project.org/wiki/PCM_Ring_Buffer (zugegriffen Juni 08, 2020).

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Werken wörtlich oder sinngemäß übernommenen Gedanken sind unter Angabe der Quellen gekennzeichnet.
Ich versichere, dass ich bisher keine Prüfungsarbeit mit gleichem oder ähnlichem Thema bei einer Prüfungsbehörde oder anderen Hochschule vorgelegt habe.

....

Ort, Datum Unterschrift