HTW DRESDEN

BACHELORARBEIT

# Vergleich verschiedener Audiosynchronisationsverfahren

*Ben Schönherr*

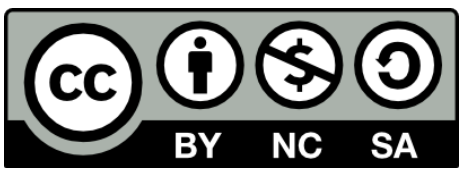
betreut von

Prof. Dr. Jörg VOGT

Zweitgutachter

Prof. Dr.-Ing. Robert BAUMGARTL

10. August 2018



# Danksagung

Ich bin Prof. Dr.-Ing. Jörg Vogt, der diese Arbeit betreute, zu besonderen Dank verpflichtet. Er stand mir mit Ratschlägen und Zeit für Diskussion zu Seite.

Zudem bedanke ich mich bei meinen Korrekturlesern Holger Schönherr, Teresa Schönherr und Paul Gottschald welche diese Arbeit grammatikalisch wesentlich verbesserten.

# Inhaltsverzeichnis

**Abbildungsverzeichnis** **X**

**Tabellenverzeichnis** **XI**

**Abkürzungsverzeichnis** **XII**

**Formelverzeichnis** **XIII**

**1 Einleitung** **1**

**2 Zielstellung** **3**

2.1 Aufgabe . . . . . 3

2.2 Lösungsweg . . . . . 3

**3 Latenz - Verzögerung - Differenz** **5**

3.1 Latenz - Verzögerung . . . . . 5

3.2 Differenz . . . . . 5

**4 Soundkarte** **7**

4.1 Aufbau . . . . . 7

4.1.1 Digitaler Signalprozessor . . . . . 7

4.1.2 Analog-Digital-Wandler . . . . . 8

4.1.3 Digital-Analog-Wandler . . . . . 8

4.2 Sampling . . . . . 8

4.2.1 Sample . . . . . 8

4.2.2 Frame . . . . . 9

4.2.3 Abtastrate . . . . . 9

4.2.4 Abtastratenkonvertierung . . . . . 9

4.2.5 Abtasttiefe . . . . . 10

4.2.6 Datenrate . . . . . 11

4.2.7 Periode . . . . . 11

4.3 Verzögerung . . . . . 11

4.3.1 Puffergröße . . . . . 11
4.3.2 DSP - Architektur . . . . . 12
4.3.3 DSP - Implementation . . . . . 12

**5 Netzwerk** **13**
5.1 OSI-Modell . . . . . 13
5.2 Übertragungswege - Local Area Network . . . . . 14
5.2.1 Ethernet . . . . . 14
5.2.2 Wireless Lan . . . . . 14
5.3 Vermittlung . . . . . 15
5.3.1 Unicast . . . . . 15
5.3.2 Multicast . . . . . 15
5.4 Datenübertragungsprotokolle . . . . . 15
5.4.1 TCP . . . . . 15
5.4.2 UDP . . . . . 16
5.4.3 RTP . . . . . 16
5.4.4 Overhead vs. Payload . . . . . 17
5.5 Round Trip Time . . . . . 17
5.6 Quality of Service . . . . . 18
5.7 Jitterbuffer . . . . . 18

**6 Zeitsynchronisation** **20**
6.1 Network Time Protocol . . . . . 20
6.1.1 Zeitstempel . . . . . 21
6.1.2 Funktionsweise . . . . . 21
6.1.3 Synchronisationsablauf . . . . . 22
6.1.4 SNTP . . . . . 22
6.1.5 Einschätzung . . . . . 22
6.2 Precision Time Protocol . . . . . 23
6.2.1 Best Master Clock Algorithmus . . . . . 23
6.2.2 Funktionsweise . . . . . 23
6.2.3 Einschätzung . . . . . 24

**7 Ausgangspunkt zur Betrachtung von Audiosynchronisationsverfahren 25**

7.1 Übertragungslaufzeit . . . . . 26

7.2 Verarbeitungszeit . . . . . 26

**8 Audiosynchronisationsverfahren 27**

8.1 Mit Offset . . . . . 27

8.1.1 Aufbau . . . . . 27

8.1.2 Variante I . . . . . 28

8.1.3 Variante II . . . . . 30

8.1.4 Wahl des Offsets . . . . . 31

8.1.5 Einschätzung . . . . . 33

8.2 Client als Synchronisationsquelle . . . . . 33

8.2.1 Aufbau . . . . . 34

8.2.2 Synchronisation . . . . . 35

8.2.3 Einschätzung . . . . . 36

8.3 Snapcast . . . . . 37

8.3.1 Funktionsweise . . . . . 37

8.3.2 Einschätzung . . . . . 39

8.4 Verzögertes Senden . . . . . 39

8.4.1 Verzögerung bestimmen . . . . . 39

8.4.2 Synchronisation . . . . . 40

8.4.3 Einschätzung . . . . . 41

**9 Synchronisation während des Streamings 43**

9.1 Pakete wiederholen/überspringen . . . . . 44

9.2 Stream verzögern/beschleunigen . . . . . 45

**10 Paketverlust 46**

**11 Praktische Tests 48**

11.1 Umsetzung . . . . . 48

11.2 Messmöglichkeiten . . . . . 49

11.3 Ideale Bedingungen . . . . . 51
11.4 WLAN . . . . . 51
11.5 Ergebnisse . . . . . 53
11.6 Fehlerquellen / Probleme . . . . . 53

**12 Ergebnisse und Bewertung** **54**

**13 Schlussbemerkungen und Ausblick** **56**

**Literaturverzeichnis** **57**

# Abbildungsverzeichnis


1 Latenzkette . . . . . 5
2 Soundkarte . . . . . 7
3 Alias-Effekt . . . . . 9
4 Abtasttiefe . . . . . 10
5 OSI-Modell . . . . . 13
6 Overhead vs. Payload . . . . . 17
7 Round Trip Time . . . . . 18
8 Network Time Protokoll . . . . . 21
9 Precision Time Protokoll . . . . . 24
10 Streaming ohne Protokoll . . . . . 25
11 Streaming mit Offset . . . . . 28
12 Zeitverlauf (Pakete besitzen ein Offset) . . . . . 29
13 Synchronisation mit Offset und RTCP . . . . . 31
14 Aufbau - Client als Synchronisationsquelle . . . . . 34
15 Client als Synchronisationsquelle . . . . . 36
16 Synchronisation durch mit Puffer . . . . . 38
17 Streaming mit verzögertem Senden . . . . . 41
18 Signalverlauf mit wiederholtem Paket . . . . . 44
19 Signalverlauf mit Nullpaket . . . . . 46
20 Signalverlauf mit sinusförmiger Analyse . . . . . 47
21 Programmablauf eines Threads für eine Verbindung . . . . . 49
22 Versuchsaufbau - Takt . . . . . 50
23 Versuchsaufbau - ansteigender Ton . . . . . 50
24 Alles in einem Rechner . . . . . 51
25 Versuchsaufbau . . . . . 52

# Tabellenverzeichnis

1 Vergleich der Audiosynchronisationsverfahren . . . . . 54

# Abkürzungsverzeichnis

**DSP** Digitaler Signal Prozessor

**ADC** Analog Digital Konverter

**DAC** Digital Analog Konverter

**WLAN** Wirless Local Area Network

**LAN** Local Area Network

**CSMA** Carrier Sense Multiple Access

**FEC** Forward Error Correction

**TCP** Transmission Control Protocol

**UDP** User Datagram Protocol

**DCCP** Datagram Congestion Control Protocol

**RTP** Real-time Transport Protocol

**RTCP** Real-time Control Protocol

**RTT** Round Trip Time

**QoS** Quality of Service

**NTP** Network Time Protocol

**SNTP** Simple Network Time Protocol

**PTP** Precision Time Protocol

**BMCA** Best Master Clock Algorithm

**PLC** Packet Loss Concealment

# Formelverzeichnis

| | | |
|---|---|---|
| $t_p$ | s | Übertragungsverzögerung |
| $t_p$ | s | Ausbreitungsverzögerung |
| $t_u$ | s | Übertragungslaufzeit |
| $t_v$ | s | Verarbeitungszeit |
| $t_o$ | s | Offset |
| $t_{pd}$ | s | Periodendauer |
| $t_l$ | s | Verzögerungszeit |
| $t_{av}$ | s | Abspielverzögerung |
| $t_w$ | s | Wartezeit |
| $t_d$ | s | Differenz |
| $T_s$ | s | Zeit des Servers |
| $T_c$ | s | Zeit des Clients |
| $T_{wg}$ | s | Zeitpunkt der Wiedergabe |

# 1 Einleitung

Heutzutage reicht es oft nicht aus Musik in nur einem Raum eines Gebäudes oder einer Wohnung abzuspielen. Vielmehr steigt die Nachfrage nach vernetzten Räumen, in denen Musik gestreamt werden kann. Dabei werden immer häufiger Multiroomlösungen verwendet. Multiroom beschreibt die Möglichkeit, in verschiedenen Räumen einer Wohnung Musik zu streamen. Es wird dabei unterschieden, ob das jeweilige netzwerkfähige Abspielgerät eigene Musik abspielt oder ob eine bestimmte Auswahl an Geräten denselben Inhalt synchronisiert wiedergeben.

Wo früher Lautsprecherkabel von einem Zimmer zum nächsten gelegt werden mussten, ist es heute möglich, nahezu unendlich viele Abspielgeräte mit einer Quelle zu koppeln und zu verwalten, ohne dass zwingend ein Netzwerkkabel verwendet werden muss. Vorausgesetzt wird ein lokales Netzwerk, in welchem die Geräte miteinander kommunizieren können. Da dieses jedoch nicht für gleiche Übertragungs- und Verarbeitungszeiten der Audiosignale garantiert, benötigt es einen Mechanismus, welcher die unbekannten Einflussgrößen so kompensiert, dass synchrones Audiostreaming auch über das lokale Netzwerk möglich ist.

In der Fachliteratur gibt es zu diesem Thema selten detailliert beschriebene Ansätze. Häufig sind Patente und Ideen zu finden, welche das Problem nur oberflächlich lösen. Es existieren bereits verschiedene kommerzielle Produkte[1], welche Multiroomlösungen mit synchroner Wiedergabe anbieten. Aber auch hier lässt sich die genaue Funktionsweise nur erahnen.

Daher lohnt es sich, tiefer in dieses Thema einzusteigen. Neben der grundsätzlichen Funktionsweise und dem Vergleich verschiedener Ansätze, ein Medium synchron wiederzugeben, darf sich der Frage gestellt werden, wie gut die Systeme funktionieren und ob diese eventuell auch in kommerziellen Lösungen An-

[1] z.B. Sonos, Sony, AirPlay

wendung finden.

# 2 Zielstellung

## 2.1 Aufgabe

In dieser Arbeit sollen unterschiedliche Audiosynchronisationslösungen vorgestellt und verglichen werden. Es wird vorausgesetzt, dass alle Audiodaten über das lokale Netzwerk übertragen werden. Neben der detaillierten Funktionsweise einzelner Lösungen sollen diese nach ihrer Genauigkeit und Verwendung bewertet werden. Zudem müssen grundlegende Probleme der Audiosynchronisation genannt und tiefgehend erörtert werden. Der Erkenntnisgewinn ergibt sich aus der Vorstellung einzelner Audiosynchronisationslösungen, aber auch aus deren Bewertung, Verwendung und eines praxisbezogenen Tests.

## 2.2 Lösungsweg

Im Kapitel drei bis sechs werden alle nötigen Grundlagen für die synchrone Wiedergabe eines Mediums im lokalen Netzwerk beschrieben. Nach einem kurzen Überblick über mögliche Verzögerungen und entstehender Differenzen wird die Funktionsweise der Soundkarte näher betrachtet. Es folgen Grundlagen über die Datenübertragung und den Aufbau eines lokalen Netzwerks. Danach wird sich im Kapitel fünf genauer mit der Zeitsynchronisation verschiedener Systeme auseinander gesetzt. Diese wird für die meisten Systeme vorausgesetzt.

Im Kapitel sieben wird der Ausgangspunkt beschrieben und der Schwerpunkt auf die eigentlichen Probleme beim Abspielen eines Mediums über das Netzwerk an verschiedenen Abspielgeräten gelegt.

Ausgehend von genannten Problemen, die sich vor allem auf Übertragungs- und Verarbeitungslaufzeit beschränken, werden im Kapitel acht vier verschiedene Audiosynchronisationsverfahren beschrieben und bewertet. Diese Verfahren sind nur eine Auswahl aller möglichen Synchronisationslösungen, zeigen aber die grundsätzlichen Unterschiede und Gemeinsamkeiten auf.

Kapitel neun stellt sich der Frage, ob während des Streamings synchronisiert werden muss oder ob ein einmaliges Synchronisieren zu Beginn des Streamings ausreicht. Des Weiteren werden Möglichkeiten für das Synchronisieren während der Wiedergabe betrachtet.
Es folgt ein Kapitel, in dem Lösungen bei Paketverlust diskutiert werden.

Nach den theoretisch beschriebenen Audiosynchronisationslösungen und deren Anwendung wird im Kapitel elf ein Synchronisationsprotokoll, welches ohne Zeitsynchronisation auskommt, praktisch getestet und deren Genauigkeit gemessen. Die daraus gewonnenen Erkenntnisse bewerten das Protokoll.

Im Kapitel 12 und 13 befinden sich gewonnene Ergebnisse und Erkenntnisse der verschiedenen Audiosynchronisationslösungen, sowie ein Ausblick in die Zukunft zum Thema Audiosynchronisation.

# 3 Latenz - Verzögerung - Differenz

## 3.1 Latenz - Verzögerung

Latenzen und Verzögerungen beschreiben grundsätzlich das Gleiche. In der folgenden Arbeit fasst der Begriff Verzögerung auch mehrere Latenzen zusammen. Ganz allgemein beschreiben diese die Zeit, die vom Ende eines Ereignisses bis zum Anfang einer Reaktion vergeht.

Verzögerungen können in vielen verschiedenen Bereichen auftreten. Bezogen auf die Audiowiedergabe beschreibt die Verzögerung die Zeit, die vom Abspielen bis zum Wahrnehmen vergeht. So beinhaltet diese neben der Zeit, die eine Soundkarte benötigt, auch die Zeit, die vergeht bis die Schallwellen vom Lautsprecher zum Empfänger[2] gelangen. Letzteres lässt sich durch die Schallgeschwindigkeit bestimmen. Diese beträgt bei 20 °C Lufttemperatur 343 m/s. Bei einem Meter Entfernung zur Schallquelle beträgt somit die Latenz 2,91ms.

Anhand des typischen Aufbaus, um Musik über das Netzwerk zu streamen, lässt sich eine Latenzkette bestimmen.

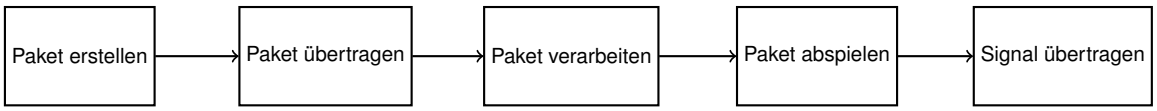



Abbildung 1: Latenzkette

## 3.2 Differenz

Die Differenz beschreibt einen in Zeit gemessenen Unterschied zweier Geräte. Wenn Uhren verschiedener Systeme verglichen werden, beschreibt die Differenz,

[2] eine Person

wie weit diese voneinander abweichen.

Wird ein Medium an verschiedenen Geräten synchron wiedergegeben, beschreibt die Differenz die zeitliche Verschiebung der Ausgangssignale zu einem bestimmten Zeitpunkt. Im Idealfall muss diese gleich Null sein. Bezogen auf synchrones Audiostreaming definiert die Differenz somit den Fehler. Je größer also diese ist, desto weiter sind die Ausgangssignale auseinander, und desto weniger synchron spielen die Geräte das Medium ab.

Nun stellt sich die Frage, wie groß die Differenz, also der Fehler, sein darf, dass beide Ausgangssignale als synchron wahrgenommen werden. Der Präzedenz-Effekt besagt, dass erst ab einer Differenz von mehr als 40 ms die Ausgänge zweier Schallquellen als getrennte Audiosignale wahrgenommen werden [22, vgl.]. Bei einer Differenz unter 40 ms werden durch Echo, veränderte Lautstärke und Klangfarbe die zu unterschiedlichen Zeitpunkten eintreffenden Signale bemerkbar. Ist die Differenz kleiner als 1,5 ms, kann diese beim Empfänger nicht mehr festgestellt werden [23, vgl.].
Aus den angegebenen Werten ergibt sich, dass Audiosynchronisationslösungen, welche Differenzen kleiner als 1,5 ms erreichen, im gleichen Raum verwendet werden können. Audiosynchronisationslösungen mit größeren Differenzen sind als Multiroomlösungen[3] zu empfehlen. So wird das veränderte Klangbild von zwei auseinanderliegenden Audiosignalen nur im Bereich zwischen zwei Räumen wahrgenommen.

[3] Abspielgeräte stehen in unterschiedlichen Räumen

# 4 Soundkarte

Für die Ausgabe eine Audiostreams ist die Soundkarte unverzichtbar. Heutzutage sind Soundkarten in nahezu jedem Computer entweder als PCI-Steckkarte oder im Chipsatz verbaut. Im folgenden Kapitel wird die Funktionsweise der Soundkarte erklärt. Zusätzlich werden die Grundlagen des Samplen behandelt sowie mögliche Gründe für Latenzen der Soundkarte genannt.

## 4.1 Aufbau

Soundkarten bestehen aus mehreren verschiedenen Komponenten, die das Verarbeiten der Audiosignale ermöglichen, indem sie miteinander kommunizieren.

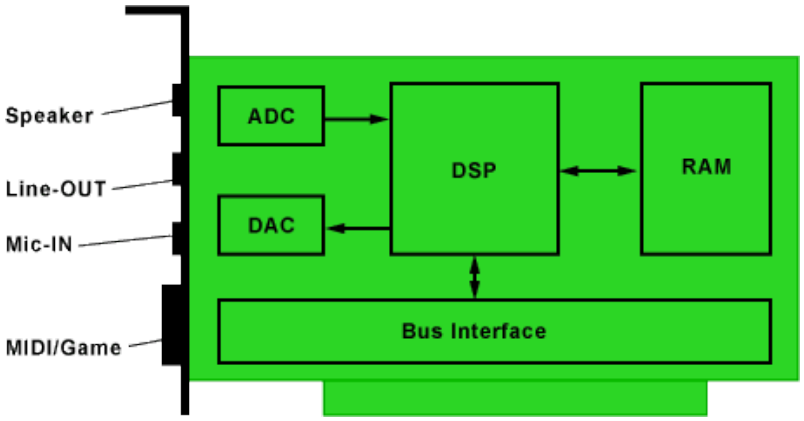



Abbildung 2: Soundkarte

Quelle: [6]

### 4.1.1 Digitaler Signalprozessor

Der digitale Signalprozessor verwendet als Input die Daten des Analog-Digitals-Wandlers oder eines Speichers[4]. Er ist speziell für die Aufgaben der Verarbeitung von Sound-Daten konzipiert [vgl. 24, Kapitel 28 S. 503]. Darunter fällt zum Beispiel das Filtern und Transformieren [5] der digitalen Signale. Für manche Auf-

[4] z.B. ein externes Medium oder die Festplatte
[5] z.B. die kontinuierliche Fourier-Transformation

gaben ist es notwendig auf mehrere Samples zuzugreifen. Der DSP löst dieses Problem mit einem Ringpuffer, in welchem die letzten Frames gespeichert werden. Da die Berechnungen und der Zugriff auf den Speicher wiederkehrend sind, wird der DSP so konzipiert, dass viele Schritte parallel abgearbeitet werden können. In jedem Fall sollte die Eingaberate gleich der Ausgaberate sein.

Bei der Verarbeitung von analogen Signalen wird zwischen Fest- und Gleitkommaarithmetik unterschieden. Hierbei geht es weniger um die Geschwindigkeit. Vielmehr kann das analoge Signal durch eine Gleitkommazahl genauer aufgelöst werden, was sich positiv auf die Rauchintensität auswirkt.

### 4.1.2 Analog-Digital-Wandler

Der Analog-Digital-Wandler wandelt ein analoges Signal in ein digitales Signal um und übergibt dieses dem DSP. Je höher die Abtasttiefe und Abtastrate ist, desto größer ist die zu verarbeitende Bandbreite und desto genauer kann das analoge Signal quantifiziert werden.

### 4.1.3 Digital-Analog-Wandler

Der Digital-Analog-Wandler wandelt das digitale Signal des DSPs in ein analoges Signal um. Aus den einzelnen Samples wird eine Impulsfolge erstellt, die die gleiche Frequenz wie das Eingangssignal besitzt.

## 4.2 Sampling

### 4.2.1 Sample

Ein Sample beschreibt genau einen Datenpunkt eines Signals. Wird eine Abtastrate von 44100Hz vorgegeben, so gibt es 44100 Samples pro Sekunde. Die Samplegröße hängt von der Abtasttiefe ab.

### 4.2.2 Frame

Für jeden Kanal gibt es ein eigenes Sample. Ein Frame wird durch die Anzahl der Samples für die unterschiedlichen Kanäle definiert [3, vgl.].

### 4.2.3 Abtastrate

Die Abtastrate wird auch oft Samplingrate genannt. Sie gibt an wie oft ein Signal in einer Sekunde abgetastet wird. Je höher diese ist, desto besser kann ein analoges Signal quantisiert werden.

Es gilt das Nyquist-Shannon-Abtastheorem, welches besagt, dass die Abtastrate mindestens doppelt so hoch wie die höchste Frequenz eines zu quantifizierten analogen Signals sein muss. Ist dies nicht der Fall werden hohe Frequenzen als niedrige interpretiert und verfälschen so das Audiosignal. Dieser Effekt wird Aliasing genannt [vgl. 24, Kap. 3 S. 40].


Abbildung 3: Alias-Effekt

Das menschliche Gehör nimmt eine Frequenz zwischen 20 Hz und 20.000 Hz wahr. Um alle Frequenzen abzudecken liegt die Abtastrate, unter Einberechnung des NS-Abtasttheorem, bei mindestens 40.000 Hz. Eine handelsübliche Audio-CD besitzt eine Abtastrate von 44.1 kHz [12, vgl.].

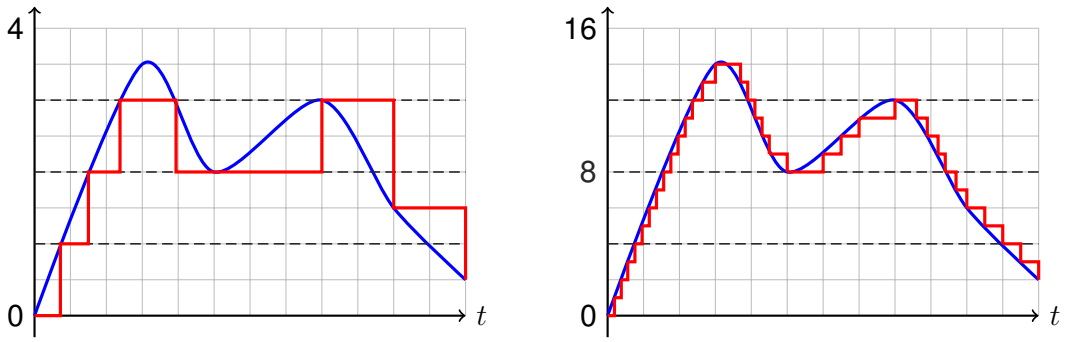
### 4.2.4 Abtastratenkonvertierung

Mit der Abtastratenkonvertierung ist es möglich, die Abtastraten eines Signals zu verändern, ohne dass merkliche Verluste entstehen [26, S. 142]. Es gibt ver-

schiedene Möglichkeiten, um die Samplingrate eines Signals zu konvertieren. Die einfachste Möglichkeit ist, ein digitales Signal in ein analoges umzuwandeln und dieses danach mit der gewünschten Abtastrate neu abzutasten [vgl. 15, S. 288]. Es ist besser, bei Steigerung der Abtastrate das Signal zu interpolieren. Dazu werden für die Berechnung der Zwischenwerte verschiedene digitale Filter verwendet. Soll hingegen die Abtastrate gesenkt werden, wird das Nyquist-Shannon-Abtastheorem vorausgesetzt. Wird zum Beispiel eine Abtastrate von 44.1 kHz vorausgesetzt, kann diese um die Hälfte reduziert werden, ohne dass ein Qualitätsverlust entsteht.

### 4.2.5 Abtasttiefe

Die Abtasttiefe, auch Bittiefe genannt, gibt an, mit welcher Genauigkeit die einzelnen Samples quantifiziert werden. Wenn die Abtasttiefe steigt, verbessert sich die Qualität und die Dynamik eines Signals. Ein analoges Signal kann bei einer Abtasttiefe von 8 Bit auf $2^8 = 256$ Quantisierungsstufen abgebildet werden. Bei einer Abtasttiefe von 16 Bit gibt es $2^{16} = 65536$ Stufen.




(a) Abtasttiefe: 4 (b) Abtasttiefe: 16

Abbildung 4: Abtasttiefe

Je geringer die Bittiefe ist, desto wahrscheinlicher sind Quantisierungsfehler. Diese sind durch ein Rauschen wahrnehmbar[vgl. 24, Kap. 3 S. 36][6].

[6]Quantisierungsfehler können durch Ditherung zum Teil reduziert werden.

### 4.2.6 Datenrate

Die Datenrate gibt die Anzahl an Bytes an, die die Soundkarte innerhalb einer Sekunde verarbeiten muss, um ein analoges Signal zu erstellen oder zu digitalisieren. Sie wird aus der Abtastrate, der Abtasttiefe und der Kanalanzahl berechnet, indem die Größen miteinander multipliziert werden [3, vgl.].

$$\text{Datenrate} = \text{Abtastrate} \cdot \text{Kanäle} \cdot (\frac{\text{Abtasttiefe}}{8}) \tag{1}$$

### 4.2.7 Periode

Eine Periode, auch Periodendauer genannt, beschreibt die Zeit, die die Soundkarte benötigt, um einen definiert großen Abschnitt (im engl. „Chunk“) eines Signals zu verarbeiten [3, vgl.]. Dabei wird die Größe des Abschnittes durch die Abtastrate dividiert.

$$\text{Periodendauer} = \frac{\text{Abschnittsgröße}}{\text{Abtastrate}} \tag{2}$$

Häufig festgelegte Werte für die Abschnittsgröße sind 512,1024 und 2048. Mit steigender Größe steigt auch die Periodendauer. Nachfolgend wird diese auch Abschnittsdauer genannt.

## 4.3 Verzögerung

Für eine perfekte Synchronisation unterschiedlicher Clients ist es notwendig, dass jeder Client die Zeit kennt, die von der Übergabe eines Samples an die Soundkarte bis zur tatsächlichen Wiedergabe am Ausgang vergeht. Im Folgenden werden Faktoren genannt, welche, bezogen auf die Soundkarte, einen verhältnismäßig großen Einfluss nehmen können.

### 4.3.1 Puffergröße

Für eine saubere Signalverarbeitung des DSPs sollte der Puffer mindestens der Größe einer Periode entsprechen. Ist die Puffergröße zu klein, sodass kein gan-

zer Abschnitt zwischengespeichert werden kann, kommt es zu Unterbrechungen, die wahrnehmbar sind [7]. Für jeden zwischengespeicherten Abschnitt steigt die Zeit um dessen Periodendauer. Daraus ergibt sich die

$$\text{Latenz} = \text{Periodendauer} \cdot \frac{\text{Puffergröße}}{\text{Abschnittsgröße}} \tag{3}$$

Je mehr Abschnitte ein Puffer zwischenspeichern kann, desto mehr Zeit vergeht bis diese abgespielt werden. Sind an jedem Gerät die Puffer der jeweiligen Soundkarte gleich groß gewählt, spielt dieser Wert bei synchronem Abspielen keine Rolle.

### 4.3.2 DSP - Architektur

Im Laufe einer Signalverarbeitung werden viele Daten verarbeitet. Wichtig ist, dass der DSP schnell genug ist, um ein Sample zu lesen, zu berechnen und zu speichern bevor das nächste Sample kommt. Da dies immer wiederkehrende Aufgaben und Instruktionen sind, eignet sich die Harvard-Architektur besonders gut [vgl. 24, Kap. 28 S. 509].

### 4.3.3 DSP - Implementation

Während Treiber in der Sprache C übersichtlicher und besser wartbar sind, ist die Perfomance von Treibern in der Sprache Assembler höher. Da die Entwicklung in Assembler aufwendiger und somit teurer ist, spiegelt sich dies auch in den Preisen einer guten Soundkarte wieder [vgl. 24, Kap. 28 S. 522].

[7] z.B. durch „Klick“-Geräusche

# 5 Netzwerk

Nun stellt sich die Frage, wie die Audiopakete von einem Server zu mehreren Clients gesendet werden. Das hängt natürlich von den Eigenschaften des Netzwerkes und der verwendeten Protokolle ab. Im Folgenden werden bekannte Übertragungswege, Vermittlungs- und Datenübertragungsprotokolle beschrieben und unterschieden.

## 5.1 OSI-Modell

Um sich die Datenübertragung von einem Endsystem zum anderen besser vorzustellen, folgt das OSI-Modell. Für die Audiosynchronisationsverfahren spielen besonders die Sicherungs-,Vermittlungs- und Transportschicht eine Rolle.

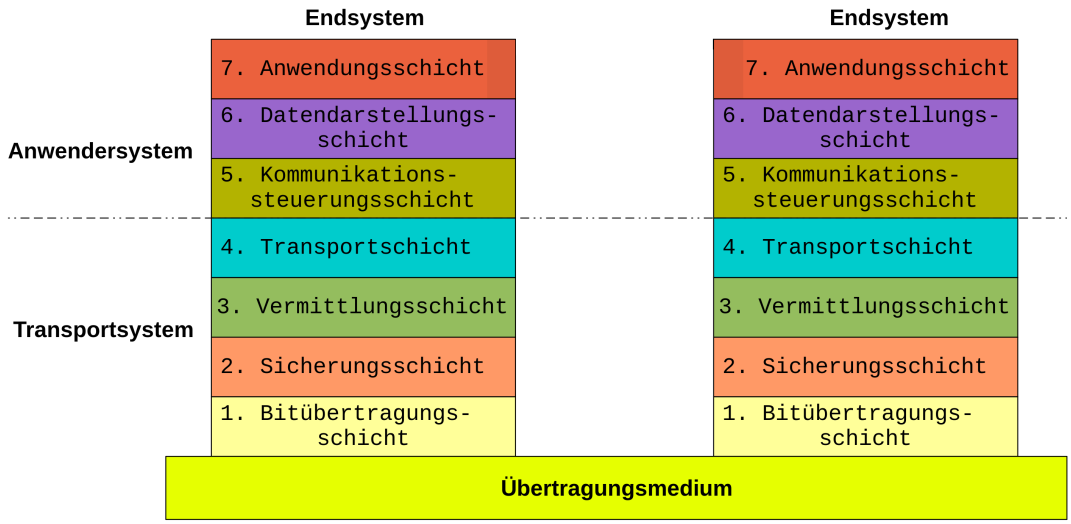



Abbildung 5: OSI-Modell

Quelle: [29]

## 5.2 Übertragungswege - Local Area Network

### 5.2.1 Ethernet

*Ethernet* basiert auf dem Standard IEEE802.3 und beschreibt das Verteilen von Netzwerkpaketen unter Verwendung eines Kabels. Es findet überwiegend in lokalen Netzwerken (LAN) Anwendung.

Die Datenpakete werden ohne festes Zeitraster übertragen. Das bedeutet, dass der Erfolg, dass ein Datenpaket zum richtigen Zeitpunkt beim Empfänger eintrifft, nicht sicher ist. „Er unterliegt nur einer gewissen Wahrscheinlichkeit. Wegen der unzuverlässigen Übertragungstechnik ist Ethernet auf die Intelligenz höherer Protokolle und Anwendungen angewiesen.“[8] Die Wahrscheinlichkeit jedoch, dass ein Paket über ein Kabel korrekt übertragen wird, ist hoch und lässt sich über die Bitfehlerwahrscheinlichkeit ausrechnen.

### 5.2.2 Wireless Lan

*Wireless Lan* basiert auf dem Standard IEEE802.11 und ist für die Kommunikation verschiedener Geräte über das Funknetzwerk zuständig. Auch wenn in der Theorie hohe Übertagungsraten erreicht werden können, wird in der Praxis die Datenrate erheblich durch Gegenstände, störende Sender und dem CSMA - Verfahren[8] gesenkt. Im Gegensatz zu Ethernet ist WLAN wesentlich störanfälliger, was sich besonders bei hohen Qualitätsanforderungen bemerkbar macht.

Gegen Paketverluste können verschiedene Fehlerkorrekturverfahren eingesetzt werden. Eines der Bekanntesten ist *Forward Error Correction* (FEC), welches verlorene Pakete durch XOR-Verknüpfung wiederherstellen kann.

[8] Mehrfachzugriffsverfahren - mehrere WLAN-Teilnehmer kommen mit der selben Frequenz aus

## 5.3 Vermittlung

### 5.3.1 Unicast

Bei *Unicast* sind Sender und Empfänger direkt miteinander verbunden. Falls es nur eine Verbindung gibt, kann die volle Bandbreite im Netzwerk verwendet werden. Für jede hinzukommende Verbindung steigt die Netzlast, womit die Bandbreite der einzelnen Verbindungen sinkt. Im Extremfall können einzelne Pakete nicht im erwarteten Zeitfenster eintreffen. Überträgt ein Sender Informationen an mehrere Empfänger, wird bei *Unicast* mehr Rechenleistung als bei *Multicast* benötigt.

### 5.3.2 Multicast

Bei *Multicast* ist der Sender mit einer Gruppe von Empfängern verbunden. Dafür greifen Empfänger und Sender auf eine bestimmte Adresse[9] zu. Der Vorteil ist, dass trotz neuer Verbindungen die Netzlast gleich bleibt. Für das Verwalten der Empfängergruppen gibt es verschiedene Protokolle[10].
Ein Nachteil ist, dass der Sender nicht zwingend darüber Bescheid wissen muss, welche Empfänger die Informationen erhalten, da die Pakete erst am Router oder Switch vervielfältigt werden.

## 5.4 Datenübertragungsprotokolle

### 5.4.1 TCP

Das *Transmission Control Protocol* ist ein verbindungsorientiertes Datenübertragungsprotokoll und befindet sich auf der Transportschicht. Durch die Bestätigung einzelner Pakete des Clients sollen Übertragungsverluste beseitigt werden. Der Paketheader ist mit 20 Byte relativ groß und besteht unter anderem aus der Sequenznummer. So können Pakete mit unterschiedlicher Reihenfolge am Client

[9] bei IPv4: 224.0.0.0 - 239.255.255.255
[10] z.B. IGMP - RFC2236

sortiert werden.

### 5.4.2 UDP

Das *User Datagram Protocol* ist ein verbindungsloses Datenübertragungsprotokoll und stellt eine Alternative zu TCP dar. Der Server erwartet für die gesendeten Pakete keine Rückmeldung. Zudem ist der Paketheader mit 8 Byte klein und besitzt keine Paketnummerierung, weshalb Pakete in unterschiedlicher Reihenfolge nicht nachträglich im Client sortiert werden können.UDP benötigt weniger Rechenleistung als TCP und wird deshalb in einigen echtzeit-kritischen Anwendungen wie Audio- und Videostreaming verwendet.

### 5.4.3 RTP

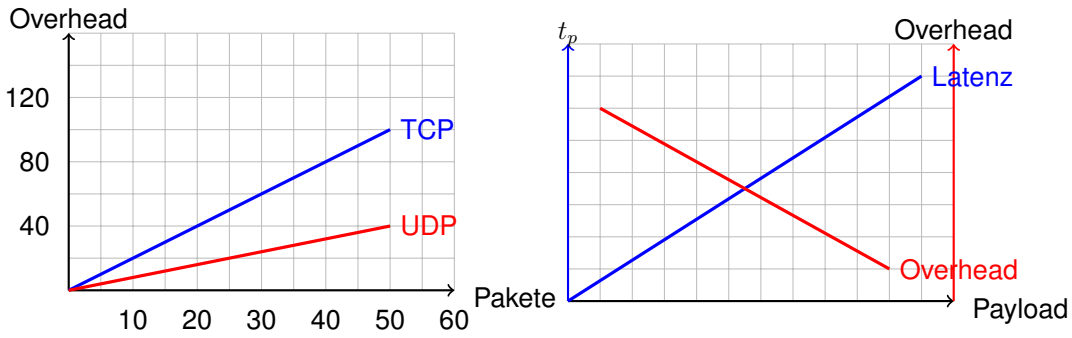
Das *Real-time Transport Protocol* befindet sich im OSI-Modell auf der Anwendungsschicht und baut auf UDP auf. Es wurde für das Übertragen von Audio- und Videodaten mit Echtzeitcharakteristik entwickelt. Zudem sind *Unicast*- und *Multicast*verbind-ungen möglich [9, vgl.]. RTP als solches unterstützt keine Quality-of-Service Vereinbarungen[11]. Vielmehr wird auf eine saubere Übertragung eines Datenstroms in möglichst kurzer Zeit geachtet. Der Paketheader besteht aus mindestens 12 Byte und enthält unter anderem den Zeitstempel des Servers. Mit diesem ist es möglich verschiedene Quellen, zum Beispiel Bild und Ton, am Client zu synchronisieren.

Neben dem *Real-time Transport Protocol* wird im RFC3550 das *Real-time Control Protocol* beschrieben. So können verschiedene Anforderungen des Streams ausgehandelt und ebenso gesteuert werden. Da RTCP-Pakete während der Übertragung der Daten auch kontinuierlich gesendet werden können, sollten diese nicht mehr als 5% der eigentlichen Sitzungsbandbreite ausmachen [vgl. 9, s. 24].

[11] z.B. sollen Informationen nur ohne Fehler übertragen werden

### 5.4.4 Overhead vs. Payload

Je größer der Header ist, desto mehr Overhead wird bei gleichbleibender Paketrate erzeugt. Wird hingegen die Nutzlast (engl. „Payload“) der Pakete vergrößert, sinkt die Paketrate und somit auch gesamte der Overhead. Jedoch steigt die Übertragungsverzögerung $t_p$ der einzelnen Pakete, da diese größer sind.




(a) Overhead

(b) Latenz (qualitativ)

Abbildung 6: Overhead vs. Payload

Die Übertragungsverzögerung $t_p$ lässt sich mit

$$t_p = (\frac{\text{Payload} + \text{Overhead}}{\text{Datenrate}}) \tag{4}$$

und der gesamte Overhead einer zu übertragenden Datei mit

$$\text{Overhead} = \frac{\text{Datei}}{\text{Payload}} \cdot \text{Header} \tag{5}$$

berechnen.

## 5.5 Round Trip Time

Die Paketumlaufszeit (RTT) gibt die Zeit an, die benötigt wird, um ein Paket vom Server zum Client und wieder zurück zu senden.

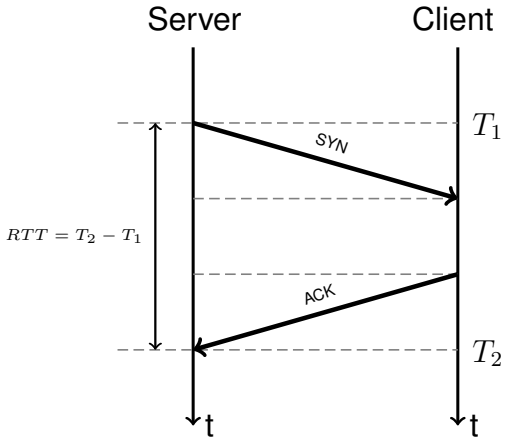



Abbildung 7: Round Trip Time

Die halbe Paketumlaufszeit beschreibt die Übertragungszeit $t_u$ in eine Richtung. Für sie gilt

$$t_u = t_p + t_a \tag{6}$$

Dabei entspricht $t_p$ der Übertragungsverzögerung und $t_a$ der Ausbreitungsverzögerung. In der folgenden Arbeit spielt die Übertragungs- oder Übertragungslaufzeit häufig eine wichtige Rolle. Sie ist aber nicht zu verwechseln mit der Übertragungsverzögerung $t_p$. Da Hin- und Rückübertragungszeit durch bestimmte Faktoren[12] unterschiedlich sein können, ist die halbe Paketumlaufzeit nur eine bedingt gute Näherung der Übertragungszeit in eine Richtung.

## 5.6 Quality of Service

*„QoS beschreibt in der TCP/IP-Welt die Güte eines Kommunikationsdienstes aus Sicht des Anwenders.“*[7] Im Bezug auf Audiostreaming wird besonders auf wenig Paketverluste und geringe Übertragungszeiten Wert gelegt.

## 5.7 Jitterbuffer

Der *Jitterbuffer* wird auf der Clientseite verwendet und gleicht Laufzeitunterschiede der einzelnen Pakete aus. Er funktioniert nach dem FIFO-Prinzip. Je mehr

[12] z.B. asymmetrisches Routing

Pakete in einem Puffer zwischengespeichert werden, desto größer ist die Zeit zwischen Empfangen und Verarbeiten eines Paketes. Wie bei dem Puffer der Soundkarte, sollten diese an allen Clients gleich groß sein um ein synchrones Abspielen zu gewährleisten.

# 6 Zeitsynchronisation

Synchronisation bedeutet, dass auf mindestens zwei voneinander unabhängigen Systemen identische Operationen zum exakt gleichen Zeitpunkt ausgeführt werden müssen. Soll ein Medium an mehreren Geräten synchron wiedergegeben werden, müssen dieselben Audiosamples zum gleichen Zeitpunkt auf allen Geräten verarbeitet werden. Fast alle Audiosynchronisationsprotokolle benötigten dazu exakt synchronisierte Uhren.

In der Praxis werden Zeitsynchronisationsprotokolle verwendet, welche die Uhren der verschiedenen Systeme miteinander synchronisieren. Die Differenzen der Uhren können unter anderem durch die eingebauten Quarzoszillatoren, welche als Taktgeber verwendet werden, entstehen. Diese können jedoch Abweichungen besitzen, welche besonders durch äußere Einflüsse[13] verursacht werden. Bei einer Abweichung von 10 ppm[14] kann eine Uhr schon nach einem Monat bis zu 26s verzögert sein. Es lohnt sich, die Uhren der Geräte regelmäßig neu zu synchronisieren oder zu überwachen.

Es gibt verschiedene Zeitsynchronisationsprotokolle. Die zwei bekanntesten, die Synchronisation über das lokale Netzwerk ermöglichen, werden im Folgenden beschrieben.

## 6.1 Network Time Protocol

Die aktuelle Version 4 des NTPs ist in dem RFC 5905 definiert und beschreibt drei unterschiedliche Synchronisationsvarianten [15]. Das Verteilen der Zeitpakete basiert auf dem verbindungslosen Transportprotokoll UDP und verwendet standardmäßig den Port 123 [5, vgl.]. Wird das *Network Time Protocol* im *Wide Area Network* verwendet, können Genauigkeiten von weniger als 10ms erreicht wer-

[13] z.B. Temperatur
[14] Parts per million $10^{-6}$, 10 ppm = 1 zu 100.000 $\approx$ 0.86 Sekunde pro Tag
[15] Symmetrisch, Client/Server und Broadcast

den.

### 6.1.1 Zeitstempel

Der NTP-Zeitstempel besteht aus 64 Bits. Dabei stehen die ersten 32 Bits für die Sekunden ab dem 1. Januar 1900 und die restlichen 32 Bits für die Sekundenbruchteile. Somit besitzt NTP eine theoretische Auflösung von 0.23 Nanosekunden [vgl. 27, S. 3].

### 6.1.2 Funktionsweise

Alle Geräte werden, abhängig von der Anzahl der Zwischenstationen zur Referenzuhr, in jeweilige Schichten (engl. „Stratum“) eingeteilt. Die Schicht Null ist eine Atom- oder Funkuhr.

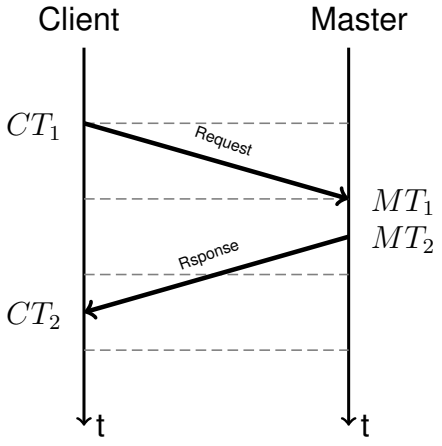



Abbildung 8: Network Time Protokoll

Soll eine Uhr synchronisiert werden, können mehrere Geräte der übergeordneten Schicht abgefragt werden. Dafür sendet der Client seinen Zeitstempel $CT_1$ an verschiedene Zeitserver. Diese erstellen einen Zeitstempel $MT_1$ beim Empfang, für welchen $MT_1 = CT_1 + t_u$ gilt, und senden diesen inklusive eines neuen Zeitstempels $MT_2$ zurück an den Client. Beim Empfang erstellt dieser einen letzten Zeitstempel $CT_2$ mit $CT_2 = MT_2 + t_u$.
Aus diesen vier Zeitstempeln lässt sich die zeitliche Differenz $\theta$ zwischen Client

und Master berechnen [vgl. 25, S. 318].

$$\theta = \frac{(MT_1 - CT_1) + (MT_2 - CT_2)}{2} \tag{7}$$

Die Übertragungsverzögerung $t_u$, in eine Richtung, lässt sich mit

$$t_u = \frac{(CT_2 - CT_1) - (MT_2 - MT_1)}{2} \tag{8}$$

berechnen. Es wird davon ausgegangen, dass die Verarbeitungszeit, die Client und Server benötigen, identisch sind.

### 6.1.3 Synchronisationsablauf

Das Netzwerk-Zeit-Protokoll NTP teilt die Synchronisation, je nach Größe der zeitlichen Differenz, in verschiedene Bereiche ein. Beträgt diese mehr als 1000s, beendet das NTP die Synchronisation und die Uhr muss vorerst manuell eingestellt werden. Ist hingegen die zeitliche Differenz zweier Uhren weniger als 125ms, geht die Zeit des Clients schneller oder langsamer, um keine großen Sprünge zuzulassen [vgl. 5, S. 48].

### 6.1.4 SNTP

Das *Simple Network Time Protocol* ist eine vereinfachte Version des NTPs. Es ist ungenauer benötigt aber weniger Rechenleistung. Die Synchronisation wird immer nur mit einem übergeordneten Gerät durchgeführt [vgl. 18, S. 2].

### 6.1.5 Einschätzung

Das *Network Time Protocol* ist nur unter bestimmten Bedingungen für synchrones Audiostreaming anwendbar. Es muss von gleichen Verarbeitungs- und Übertragungszeiten ausgegangen werden. Ist dies nicht der Fall, steigt die Abweichung [25]. Je weiter die Clients von der Referenz entfernt sind, desto größer wird der Fehler. Ursache dafür können unterschiedliche Ausbreitungsgeschwindigkeiten und die wachsende Zahl an Zwischenstationen sein. Wird das NTP im

lokalen Netzwerk verwendet und alle Geräte synchronisieren sich mit dem selben Master, ist dieses Protokoll für synchrones Audiostreaming durchaus geeignet.

## 6.2 Precision Time Protocol

Das PTP ist im IEEE 1588 beschrieben. Anders als bei dem *Network Time Protocol* gibt es nur einen Vorgänger der als Referenz dient. Diese wird nach dem *Best Master Clock Algorithm* ausgewählt. Das *Precision Time Protocol* ist für lokale Netzwerke entwickelt und kann durch Hardware oder Software realisiert werden.

### 6.2.1 Best Master Clock Algorithmus

Der BMCA Algorithmus wählt die Uhr als Referenz aus, die am meisten stabil und exakt funktioniert [16, vgl.]. Dabei werden verschiedene Eigenschaften, unter anderem deren Genauigkeit und Klassifikation, miteinander verglichen.

### 6.2.2 Funktionsweise

Nachdem alle Geräte mit den jeweiligen zu synchronisierenden Uhren in eine Hierarchie gebracht wurden, können die Slaves mit dem Master synchronisiert werden. Dazu beherrscht das Protokoll vier verschiedene Nachrichtentypen [vgl. 25, S. 320]. Als Erstes sendet der Server über Multicast in einem bestimmten Zeitintervall eine Synchronisationsnachricht (Sync) mit seinem Zeitstempel $MT_1$ an alle Clients. „Zum anderen wird zusätzlich T1' an der Ethernetschnittstelle gemessen und in einem separaten Paket nachgeschickt, um die in der Pipeline des Stack zur Hardware auftretende Zeitspanne zu eliminieren."[11] Jeder Client antwortet indem er seine Empfangszeit $CT_1$ und Sendezeit $CT_2$ zurückschickt. Beim Empfangen der Antwort erstellt der Server einen Zeitstempel $MT_2$. Aus allen vier Zeitstempeln lässt sich nun die zeitliche Differenz $\theta$ wie bei dem Network Time Protokoll berechnen. Diese wird zurück an den Client gesendet, der mit $t_u = ST_3 - MT3 + \theta$ die Übertragungszeit errechnet. Die korrekte Zeit wird dann mit

$$NewST = OldST - t_u - \theta \tag{9}$$

ermittelt [vgl. 25, S. 320].

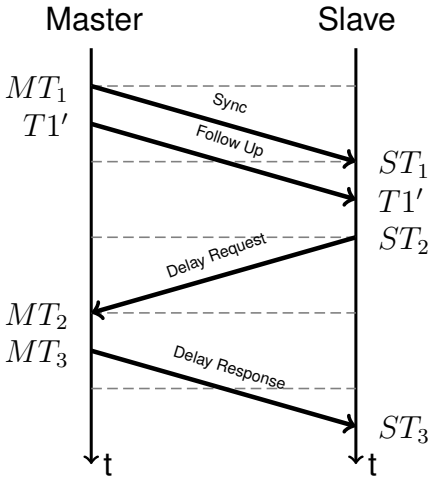



Abbildung 9: Precision Time Protokoll

### 6.2.3 Einschätzung

PTP ist bezüglich der Probleme vergleichbar mit dem Network Time Protokoll. Für gute Ergebnisse müssen ebenfalls gleiche Paketlaufzeiten sowohl zum Client als auch zum Master vorausgesetzt werden. Wird das *Precision Time Protocol* mit Hardware realisiert, sind durch hohe Auflösungen und gleiche Antwort- und Verarbeitungszeiten sehr gute Ergebnisse möglich. Es lassen sich mit Softwarelösungen Differenzen unter 200 µs erreichen. Diese Werte sind aber im Wesentlichen von Plattform, Rechenleistung und Netzlast abhängig [vgl. 2, S. 16].

# 7 Ausgangspunkt zur Betrachtung von Audiosynchronisationsverfahren

Ausgehend von den vorherigen Kapiteln ist es möglich, Audiodaten von einem Server an mehrere Clients zu senden und diese auch abzuspielen. Zudem können die Uhren der einzelnen Geräte synchronisiert werden.

Nun stellt sich die Frage, welche Parameter in den Synchronisationsprotokollen näher betrachtet werden müssen, damit alle Clients ein Medium synchron abspielen. Dazu wird als Nächstes ein Ansatz ohne Synchronisationsprotokoll verfolgt.

Der Server sendet alle Pakete in einem lokalen Netzwerk, im Abstand der Periodendauer an alle Clients. Für die Übertragung kann das verbindungslose Übertragungsprotokoll UDP mit Multicast verwendet werden. Empfängt einer der Clients ein Paket, spielt er dieses sofort ab. Ziel ist es, dass die Pakete, ausgehend vom Server, so schnell wie möglich zu den Empfängern gelangen und abgespielt werden. Die Jitterbuffer der Empfänger und die Puffer der Soundkarten sind gleich groß.

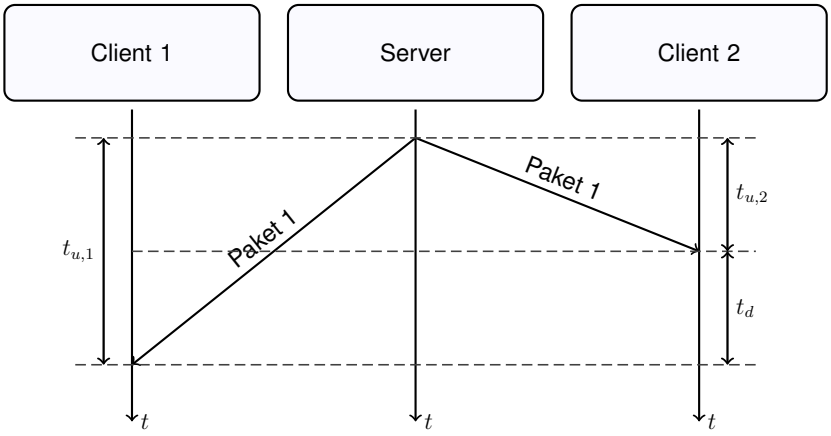



Abbildung 10: Streaming ohne Protokoll

In der Praxis gibt es jedoch unbekannte Einflussgrößen, welche synchrones Audiostreaming nicht ermöglichen. Dazu zählen die verschiedenen Übertragungs-

laufzeiten $t_u$ zu den Empfängern sowie die unterschiedlichen Verarbeitungszeiten $t_v$ der einzelnen Clients.

## 7.1 Übertragungslaufzeit

Das lokale Netzwerk kann für gleiche Paketübertragungszeiten der einzelnen Verbindungen nicht garantieren. Das gilt besonders bei Netzwerken, die aus Ethernet und WLAN-Verbindungen bestehen. Die Wahrscheinlichkeit, dass die Pakete zu unterschiedlichen Zeitpunkten bei den Clients eintreffen und der Inhalt nicht synchron abgespielt wird, ist sehr hoch. Unter bestimmten Bedingungen[16] lassen sich die Abweichungen zwar minimieren, aber im alltäglichen Gebrauch unterscheiden sich die Übertragungszeiten zum Teil deutlich.

## 7.2 Verarbeitungszeit

Die Verarbeitungszeit $t_v$ hängt von vielen Faktoren ab. Dazu gehören Rechenleistung, Implementierung und Digital-Analog-Wandler. Auch die Priorisierung der Prozesse eines Betriebssystems kann eine Rolle spielen. Für jeden Client variiert somit die Zeit die vom Empfangen eines Paketes bis zum tatsächlichen Wahrnehmen eines Tones am Ausgabegerät vergeht.

[16] z.B. kabelgebundene Verbindungen

# 8 Audiosynchronisationsverfahren

Ziel ist es, den Inhalt der Pakete zum exakt gleichen Zeitpunkt an verschiedenen Clients wiederzugeben, trotz dass es unterschiedliche Übertragungs- und Verarbeitungszeiten gibt. Verschiedene Ansätze werden durch die folgenden Synchronisationsverfahren beschrieben und verglichen. Dabei wird in diesem Abschnitt auf die erste Synchronisation, bevor die Clients mit der Wiedergabe beginnen, Wert gelegt. Ob später weitere kontinuierliche Synchronisationen notwendig sind, wird im Kapitel Synchronisationsablauf während des Streamings untersucht.

## 8.1 Mit Offset

Um die unterschiedlichen Übertragungs- und Verarbeitungszeiten zu vernachlässigen, kann ein Offset verwendet werden, welches mindestens so groß wie die längste Abspielverzögerung ist, welches aus Übertragungs- und Verarbeitungszeit besteht. Dieses wird auf den jeweiligen Zeitstempel der einzelnen Audiopakete addiert. Wird ein in der Zukunft liegender Zeitstempel eines Paketes erreicht, muss dieses wiedergegeben werden. Die folgenden Varianten unterscheiden sich darin, ob das Offset der Client oder der Server einberechnet [vgl. 19, S. 1].

### 8.1.1 Aufbau

Es wird davon ausgegangen, dass die Pakete mit dem Echtzeittransportprotokoll RTP basierend auf UDP, übertragen werden. Alle Clients sind über eine Multicastadresse erreichbar. Die Uhren aller Geräte, also aller Empfänger und die des Servers, müssen miteinander synchronisiert werden. Je nach Anforderung bezüglich der Genauigkeit kann dies mit dem *Network Time Protocol* oder *Precision Time Protocol* umgesetzt werden.

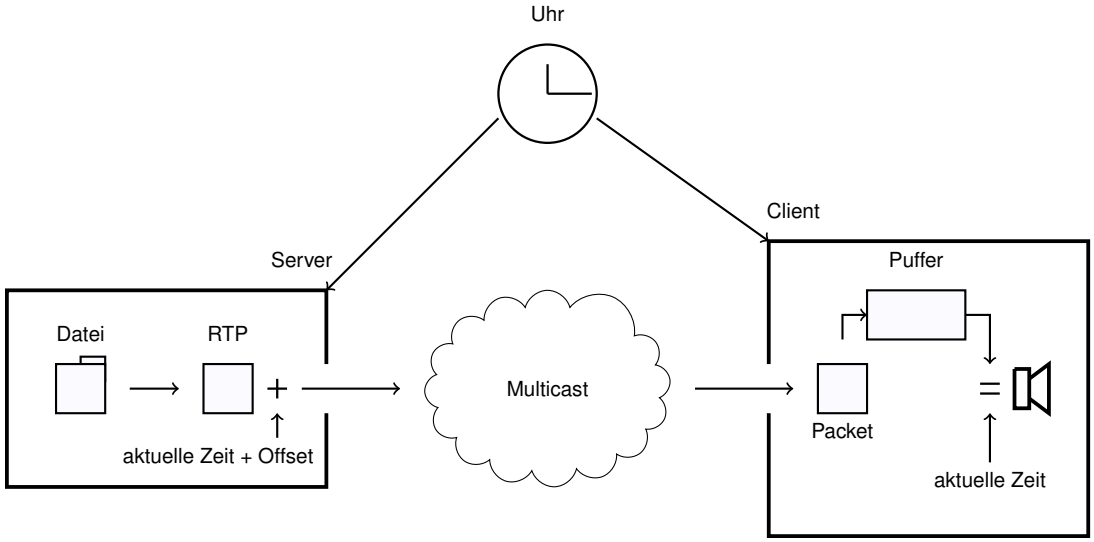



Abbildung 11: Streaming mit Offset

### 8.1.2 Variante I

Der Server öffnet die Quelle, also eine Datei oder einen Stream, und erstellt aus den einzelnen Samples RTP-Pakete. Diese besitzen neben den typischen Parametern den Zeitstempel, welcher den Zeitpunkt $T_s$ beschreibt, zu welchem das Paket vom Server abgesendet wird. Der Abstand einzelner Zeitstempel ist immer gleich und lässt sich durch die Periodendauer $t_{pd}$ bestimmen. Bevor die einzelnen RTP-Pakete an die Multicastadresse gesendet werden, wird auf dem jeweiligen Zeitstempel der Pakete ein Offset $t_o$ addiert, welches größer als die Summe aus der höchsten Übertragungslaufzeit $t_u$ zu einem der Clients und der dazugehörigen längsten Verarbeitungszeit $t_v$ der jeweiligen Empfänger ist.

Die Abspielgeräte empfangen nun zu verschiedenen Zeitpunkten die Pakete mit dem Zeitstempel $T_{wg}$, welcher durch das addierte Offset $t_o$ auf dem Zeitstempel des Servers $T_s$ in der Zukunft liegt und größer als der lokale aktuelle Zeitpunkt $T_c$ des Clients ist. Nun warten diese solange bis die aktuelle Zeit $T_c$ gleich dem Zeitstempel $T_{wg}$ ist. Ist dies der Fall, kann das Paket abgespielt werden. Da alle Uhren der Clients gleich gehen, wird der Inhalt des Paketes überall zum exakt gleichen Zeitpunkt abgespielt. Dafür muss jedoch zusätzlich vorausgesetzt werden, dass,

nachdem die beiden Zeitstempel übereinstimmen, die Daten sofort ausgegeben werden. So wird gewährleistet, dass alle vorherigen Verarbeitungsschritte und die damit verbundenen Zeiten durch das Offset vernachlässigt werden können. Das bedeutet auch, dass bei Verwendung eines Puffers dessen Länge $t_b$ in dieser Variante mit in der Verarbeitungszeit $t_v$ enthalten ist und beim Bestimmen des Offsets nicht vergessen werden darf.

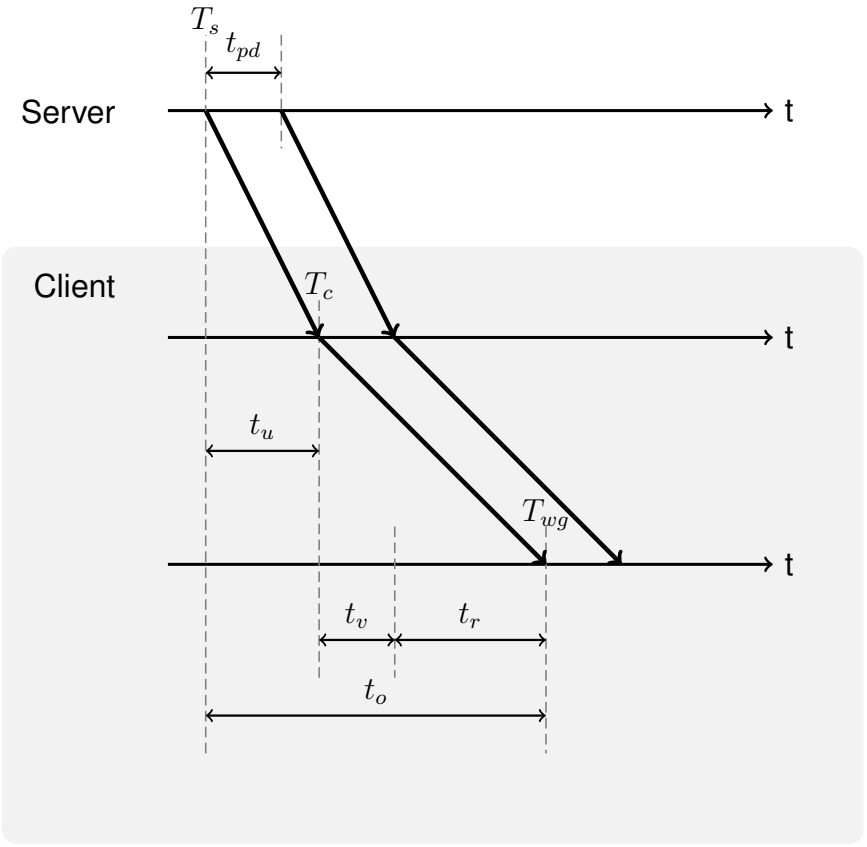



Abbildung 12: Zeitverlauf (Pakete besitzen ein Offset)

Die restliche Zeit

$$t_r = t_o - t_u - t_v \tag{10}$$

ist nach dem Synchronisieren des ersten Paketes bei jedem Empfänger durch die unterschiedlichen Übertragungslaufzeiten $t_u$ und Verarbeitungszeiten $t_v$ verschieden. Sie gibt an, wie lange die einzelnen Clients warten bevor der Inhalt abgespielt werden muss. Ist $t_r$ negativ, kommt das Paket zu spät an, was wiederum eine Synchronisation unmöglich macht. Der Grund dafür ist ein zu klein

gewähltes Offset.

Microsoft erreichte 2004 mit diesem Protokoll Latenzen von weniger als 50 µs [vgl. 10, S. 13] der einzelnen Clients. Mit diesen Werten können problemlos mehrere Wiedergabegeräte in einem Raum ohne hörbare Unterschiede verwendet werden. Diese wurden vor allem durch eine hardwarebasierte Zeitsynchronisation erreicht.

### 8.1.3 Variante II

Das Grundprinzip dieser Lösung ist der ersten Variante sehr ähnlich. Diesmal wird das Offset jedoch nicht im Server sondern im Client addiert. Der Server muss lediglich die einzelnen RTP-Pakete mit dem passenden Zeitstempel $T_s$ vor dem Absenden an die Multicastadresse senden und eventuell ein Offset $t_o$ an alle Empfänger im Voraus verteilen. Die Clients empfangen die RTP-Pakete, addieren das Offset auf den Zeitstempel der Pakete, und warten solange bis ihre eigene Zeit der des ausgerechneten Zeitstempels entspricht. Dann wird der jeweilige Inhalt der Pakete abgespielt.

Anders als bei Variante I müssen der Server und die Clients neben den RTP-Paketen auch Pakete, die das Offset enthalten, senden und empfangen können. Dafür kann das *Real Time Control Protocol* verwendet werden.

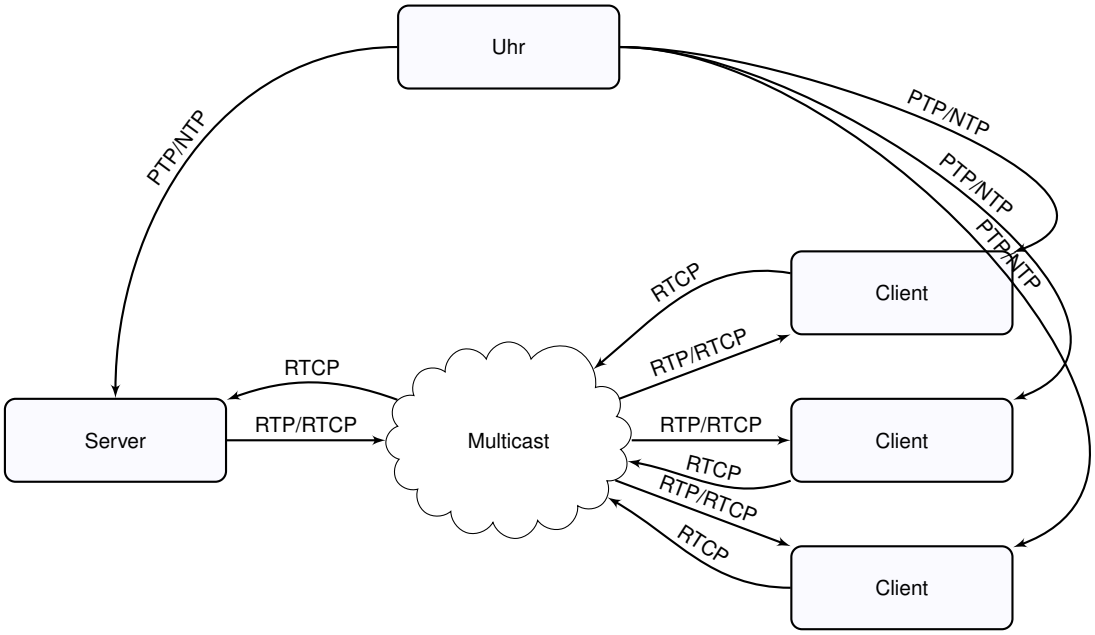



Abbildung 13: Synchronisation mit Offset und RTCP

Vorteil dieser Variante ist, dass der Server nicht mehr zwingend mit dem Offset in Berührung kommt. Dieses kann der Server entweder zusätzlich senden oder die Clients handeln untereinander ein Offset aus. Auch wenn der Server somit weniger Aufwand besitzt, kommen dennoch eine Reihe an zusätzlichen Aufgaben hinzu, die die Empfänger umso mehr auslasten.

Die grundsätzlichen Funktionen dieser Variante wurden auch von der Firma Sonos näher untersucht. Dabei werden die Uhren aller Geräte mit dem *Simple Network Time Protokoll* synchronisiert [vgl. 28, S. 18]. Alle Audiopakete besitzen ebenfalls einen Zeitstempel und werden über eine Multicastadresse an die Clients ausgehend von einem Master verteilt [vgl. 28, S. 18]. Ob diese Variante jedoch in den Produkten von Sonos eingesetzt wird, lässt sich nicht genau sagen.

### 8.1.4 Wahl des Offsets

Das Offset $t_o$ kann auf drei verschiedenen Wegen bestimmt werden. Dabei sind neben den Übertragungslaufzeiten auch die unterschiedlichen Verarbeitungszei-

ten zu beachten.

Die einfachste Möglichkeit ist das Offset zu schätzen. Es wird eine Zeit angenommen, die mit einer hohen Wahrscheinlichkeit größer als die Summe aus der längsten Übertragungs- und Verarbeitungszeit ist. Vorteil dieser Lösung ist der geringe Implementierungsaufwand. Jedoch kann nur mit einer gewissen Wahrscheinlichkeit davon ausgegangen werden, dass das Offset groß genug ist. Je größer es ist, desto größer ist auch die Wahrscheinlichkeit, dass alle Clients die Audiopakete rechtzeitig abspielen. Typische Werte liegen zwischen 0.5 und 2 Sekunden.

Eine andere Möglichkeit wäre, das Offset anhand der größten *Round Trip Time* inklusive der jeweiligen Verarbeitungszeit zu bestimmen. Bei Variante I wird die Übertragungslaufzeit vom Server ausgehend bestimmt. Um die Verarbeitungszeit eines Clients mit einzukalkulieren, kann dieser dessen doppelte Verarbeitungszeit warten, bevor eine Antwort für die RTT-Berechnung zurückgesendet wird. Die gemessene *Round Trip Time* am Server besteht dann aus $RTT = 2 \cdot (t_u + t_v)$. Wird diese Zeit durch zwei dividiert, besitzt der Server das minimal mögliche Offset dieser Verbindung. Das allgemeine Offset richtet sich nach der höchsten benötigten Zeit der einzelnen Verbindungen.
Die maximale Verarbeitungszeit der einzelnen Clients muss durch eine Initialisierungsfunktion berechnet werden. Es ist wichtig, dass auch Puffergrößen beachtet werden. Handeln bei Variante II die Clients das Offset aus, so müssen sie auch ihre Übertragungs- und Verarbeitungszeit messen und mitteilen.
Diese Lösung ist wesentlich genauer als eine Schätzung und lässt auch keine falschen Offsets zu. Es steigt neben der Komplexität auch der Implementierungsaufwand.

Da alle Uhren der Geräte gleich gehen, kann die Übertragungslaufzeit anstatt mit der *Round Trip Time* auch über die jeweiligen Zeitstempel errechnet werden. Dazu wird der Zeitstempel des empfangenen Paketes von der Ankunftszeit am

Client subtrahiert. Der Client kann auf die Übertragungslaufzeit seine Verarbeitungszeit addieren und das Ergebnis als Abspielverzögerung an den Server oder die Clients senden, welche dann die maximale Verzögerung als Offset annehmen.

### 8.1.5 Einschätzung

Das beschriebene Verfahren, welches ein Offset verwendet, besitzt den Vorteil, dass alle Clients über eine Multicastadresse angesprochen werden und, je nach dem wie das Offset bestimmt wird, kein Rückkanal erforderlich ist. Somit können viele Clients ohne steigernde Netzlast mit Audiopaketen bedient werden. Das bedeutet aber auch, dass der Server seine Empfänger nicht zwingend kennen muss. Geht ein Paket bereits auf dem Weg zum Router, der diese verteilt, verloren, empfängt dieses keiner der Clients.

Alle Übertragungs- und Verarbeitungsschwankungen können mit diesem Prinzip abgedeckt werden. Der Fehler dieser Variante wird durch den Fehler des Zeitprotokolls beeinflusst. Umso weiter die Uhren auseinander laufen, desto größer ist auch die Differenz der einzelnen Streams.

## 8.2 Client als Synchronisationsquelle

Das nächste Verfahren wurde von der Universität Oulu [17] in Finnland entwickelt und ist so genau, dass mehrere Wiedergabegeräte in einem Raum stehen können, welche verschiedene Kanäle abspielen, ohne dass Differenzen der Ausgangssignale wahrnehmbar sind. Die unterschiedlichen Übertragungs- und Verarbeitungslaufzeiten können vernachlässigt werden, indem jeder Client einen Puffer besitzt, welcher vor dem Abspielen mit Audiodaten gefüllt wird. Nach der Synchronisation greift jeder Client auf die Stelle seines Puffers zu, welche sich aus Übertragungs- und Verarbeitungszeit ergibt.

### 8.2.1 Aufbau

Alle Uhren der Geräte werden mit dem *Precision Time Protocol* synchronisiert. Alle Audiodaten werden über *Multicast* an die Empfänger übertragen. Zudem besitzt der Server zu jedem Client eine zusätzliche TCP-Verbindung, auch Kontrollkanal genannt, um individuelle Einstellungen vorzunehmen (wie zum Beispiel Kanalnummer, Lautstärke) und periodische Lebenszeichen vom Client zu erhalten.

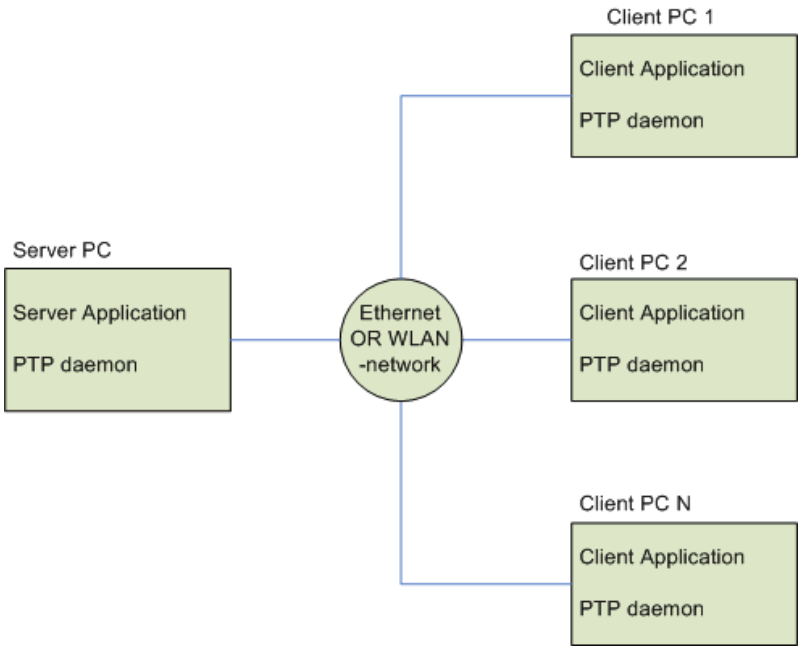



Abbildung 14: Aufbau - Client als Synchronisationsquelle

Quelle: [17]

Die Daten werden basierend auf dem *Real Time Protocol* übertragen. Dieses wurde unter anderem durch Herausnehmen des *Real Time Control Protocol* angepasst. Da die Gefahr relativ hoch ist Pakete zu verlieren, verwenden die Entwickler zusätzlich *Forward Error Correction* [vgl. 17, S. 787]. Alle Clients besitzen gleich große Puffer, die gefüllt werden müssen, bevor mit der Wiedergabe begonnen werden kann. Die Puffergröße muss mindestens der längsten Übertragungslaufzeit eines Paketes vom Server zum Client entsprechen.

### 8.2.2 Synchronisation

Aufgabe ist es, den Audiostream an allen Geräten synchron abzuspielen. Dazu sendet jeder Client über *Multicast* eine Synchronisationsnachricht an den Server. Diese werden periodisch beginnend von unterschiedlichen Startzeiten, um Paketverlust zu vermeiden, gesendet. Sie enthält den lokalen Zeitstempel, die aktuelle Samplenummer und die Nettoverbrauchsrate. Letzteres wird durch die Empfangsrate und der Abspielrate bestimmt. Der Client, der die höchste Nettoverbrauchsrate besitzt und somit die empfangenen Pakete am schnellsten abspielt, wird als Synchronisationsquelle verwendet. Alle Clients orientieren sich an diesem, indem sie den eigenen Zeitstempel und die Samplenummern mit denen der Synchronisationsquelle abgleichen. Dadurch kann die Differenz zum tatsächlichen Abspielzeitpunkt der Synchronisationsquelle für ein bestimmtes Audiopaket berechnet werden.
Nun können die Clients Pakete löschen, hinzufügen oder ihre Wiedergabegeschwindigkeit anpassen, um den tatsächlichen Abspielzeitpunkt der Synchronisationsquelle zu erreichen [vgl. 17, S. 787-788].

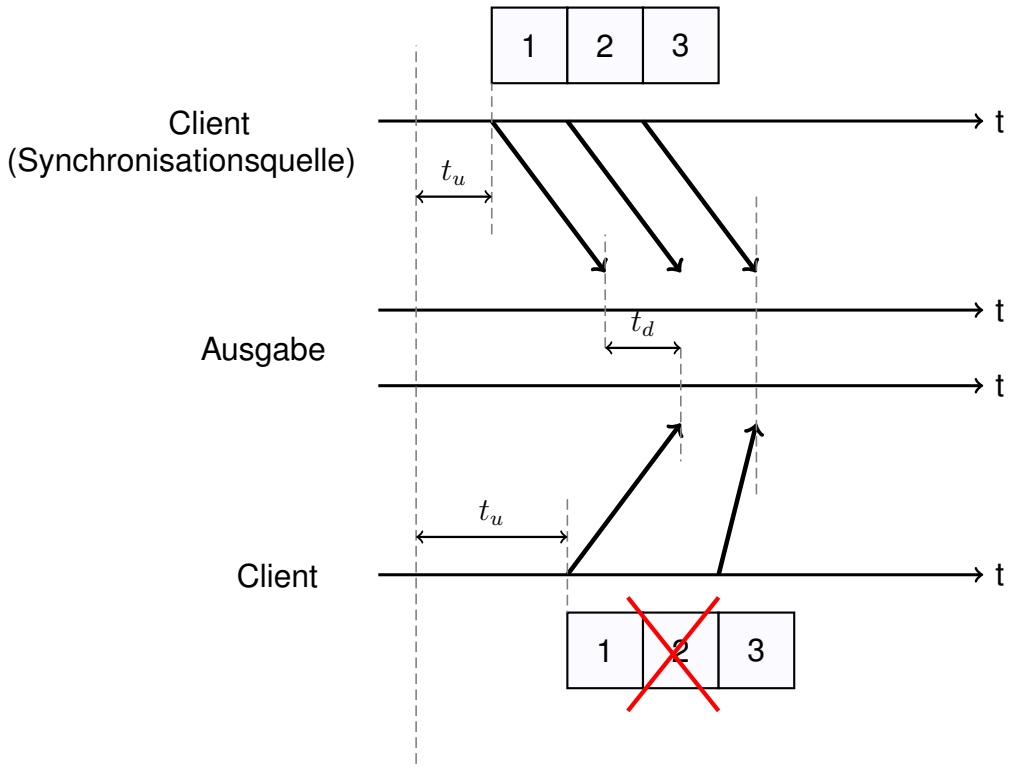



Abbildung 15: Client als Synchronisationsquelle

### 8.2.3 Einschätzung

Mit diesem Audiosynchronisationsverfahren können Schwankungen in den Übertragungs- und Verarbeitungszeiten vernachlässigt werden. Die Universität von Oulu erreichte in Tests mit 4 Clients bei LAN eine Standardabweichung von 19.8 µs und bei WLAN eine Standardabweichung von 60.6 µs [17, S. 788]. Wenngleich die Genauigkeit unter Verwendung von WLAN etwas schlechter ist, können sich mit den erzielten Werten mehrere Clients in einem Raum befinden, ohne dass eine Differenz wahrnehmbar ist. Durch die ständigen Synchronisationsnachrichten und dem Aushandeln der Synchronisationsquelle steigt die Netzlast nur minimal.

## 8.3 Snapcast

Snapcast ist ein Opensourceprojekt, welches für viele verschiedene Streamingdienste[17] eine synchronisierte Multiroomunterstützung anbietet. Dabei werden die vorher behandelten Verfahren kombiniert. Ein erster wesentlicher Unterschied ist, dass die Daten nicht basierend auf UDP mit RTP gesendet werden, sondern mit TCP und somit Unicast an die einzelnen Clients. Für den Entwickler Johannes Pohl stand die Audioqualität im Vordergrund [20, vgl.]. Diese wird jedoch mit UDP durch die Paketverluste besonders unter Verwendung von WLAN negativ beeinflusst. Wird TCP als Übertragungsprotokoll verwendet, gibt es keine Paketverluste. Dennoch bleibt die Gefahr, dass die Pakete nicht zur richtigen Zeit ankommen, bestehen. Hierfür ist ein Prüfmechanismus sinnvoll, welcher sicherstellt, ob ein Paket noch abgespielt werden kann oder nicht. Zudem kann TCP nur auf Basis von Unicast verwendet werden, was sich bei mehreren Verbindungen negativ auf die Netzlast auswirkt. Laut Johannes Pohl [20, vgl.] werden in der Regel nicht mehr als 10 Geräte miteinander verbunden.

### 8.3.1 Funktionsweise

Alle Uhren der einzelnen Geräte sind miteinander synchronisiert. Die Audiopakete enthalten immer einen Zeitstempel $T_s$, welcher den Sendezeitpunkt des Servers beschreibt und im Folgenden auch Abspielzeitpunkt genannt wird. Am Server kann die Pufferlänge $t_b$ des Puffers, welcher nach dem LIFO-Prinzip[18] funktioniert, eingestellt werden. Die Pufferlänge beschreibt die Zeit, die alle Clients warten bis sie mit dem Abspielen beginnen dürfen. Sie sollte länger als die Summe der längsten Übertragungslaufzeit und der Verarbeitungszeit sein. In der Regel ist eine Pufferlänge von einer Sekunde ausreichend. Wenn das erste Paket vom Puffer genommen wird, lässt sich aus der Puffergröße $t_b$, dem Zeitstempel des Paketes $T_s$ sowie der aktuellen Zeit des Clients $T_c$ die Abspielverzögerung $t_{av}$ berechnen. Diese Verzögerung beschreibt, wie weit die Wiedergabe des Clients

[17] z.B. Mopidy, AirPlay, MPlay, Spotify
[18] Last In First Out

vom tatsächlichen Abspielzeitpunkt des Servers entfernt ist.

$$t_{av} = t_v + t_u = (T_c - t_b) - T_s \tag{11}$$

Wurden noch keine Korrekturen vorgenommen, besteht die Abspielverzögerung $t_{av}$ aus der Übertragungslaufzeit $t_u$ und der Verarbeitungszeit $t_v$. Nun kann vom aktuellen Zeitpunkt $T_c$ die Abspielverzögerung abgezogen werden. So wird der Punkt im Puffer erreicht, den der Client gerade abspielen sollte. Ziel ist es nun, die Wiedergabe so zu beschleunigen oder abzubremsen, dass die Clients sich immer an diesem Punkt aufhalten. Dazu können Pakete gelöscht, hinzugefügt oder deren Wiedergabegeschwindigkeit angepasst werden.

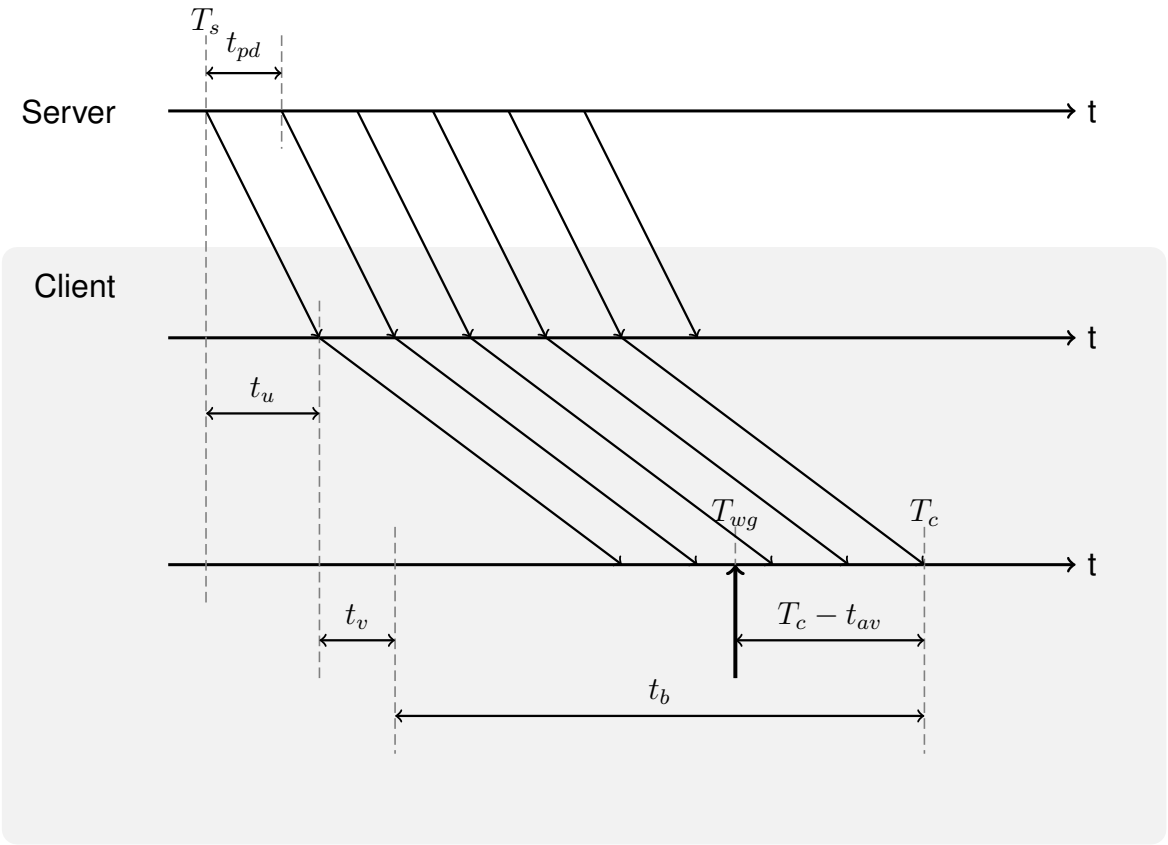



Abbildung 16: Synchronisation durch mit Puffer

### 8.3.2 Einschätzung

Snapcast erreicht typische Differenzen der einzelnen Clients von unter 1 ms [21] und ist somit für Multiroom-Anwendungen geeignet. Jedoch sind für mehrere Abspielgeräte in einem Raum die Verzögerungen grenzwertig. Ähnlich wie bei dem vorherigen Protokoll wird ein Puffer verwendet, welcher die unterschiedlichen Übertragungs- und Verarbeitungszeiten verarbeitet. Je größer die Differenz der einzelnen Uhren der Geräte ist, desto größer wird auch die wahrnehmbare Differenz zwischen den Wiedergabegeräten sein.

## 8.4 Verzögertes Senden

Alle vorher beschriebenen Verfahren setzen gleich gehende Uhren der Clients und zum Teil auch der des Servers voraus. Nun stellt sich die Frage, ob es eine Variante gibt, die synchrones Streaming an verschiedenen Geräten ohne Zeitsynchronisation ermöglicht. Dazu wird der folgende Ansatz näher untersucht.

Grundsätzlich sollen die Pakete zeitlich so versetzt zu den einzelnen Empfängern gesendet werden, dass diese zum gleichen Zeitpunkt abgespielt werden können. Durch das zeitlich versetzte Senden gleicht der Server die unterschiedlichen Übertragungs- und Verarbeitungszeiten der Clients aus.

Es ist notwendig, dass der Server alle Clients kennt und diese separat ansprechen kann. Dazu können auf Basis von *Unicast* die Audiopakete in Form von RTP-Paketen an die einzelnen Clients gesendet werden. Bevor die Übertragung beginnen kann, müssen Übertragungslaufzeit und Verarbeitungszeit bestimmt werden. Jeder Client kann seine eigene Verarbeitungszeit durch eine Initialisierungsfunktion bestimmen.

### 8.4.1 Verzögerung bestimmen

Nun kann, wie im Audiosynchronisationsprotokoll mit einem Offset, der Client bei einer *Rount Trip Time* Messung doppelt solange warten wie dessen Verar-

beitungszeit ist und sendet erst dann eine Antwort zurück. Der Server dividiert den gemessenen Wert durch zwei und erhält die Verzögerung bis ein Paket ausgehend vom Server am Client abgespielt wird. Für die Abspielverzögerung gilt $t_{av} = t_u + t_v$.

Eine andere Möglichkeit ist, dass der Server nur die Übertragungslaufzeit zu den einzelnen Clients bestimmt und somit die Verzögerung erhält bis ein Paket ausgehend vom Server am Client angekommen ist. Die unterschiedlichen Verarbeitungszeiten können ähnlich wie bei dem Snapcast-Protokoll vernachlässigt werden. Dazu besitzen alle Clients einen gleich großen Puffer und kennen ihre jeweilige Verarbeitungszeit. Wurde das Streaming gestartet und der Puffer gefüllt, kann auf das Paket im Puffer zugegriffen werden, welches ohne Verarbeitungszeit im Moment abgespielt werden würde.

### 8.4.2 Synchronisation

Der Server kennt nun alle Abspielverzögerungen, welche aus Übertragungs- und Verarbeitungszeit bestehen. Ausgehend von der längsten Abspielverzögerung werden die Pakete an die Clients gesendet. Ist eine Abspielverzögerung $t_{av,i}$ kleiner als die größte gemessene Abspielverzögerung $t_{av,max}$ zu einem der Empfänger, kann eine Wartezeit $t_{w,i}$, bevor das Paket den Server verlässt, mit

$$t_{w,i} = t_{av,max} - t_{av,i} \tag{12}$$

bestimmt werden. Sobald am Client der Jitterbuffer mit den empfangenen Paketen gefüllt wurde, beginnt dieser die Pakete abzuspielen.

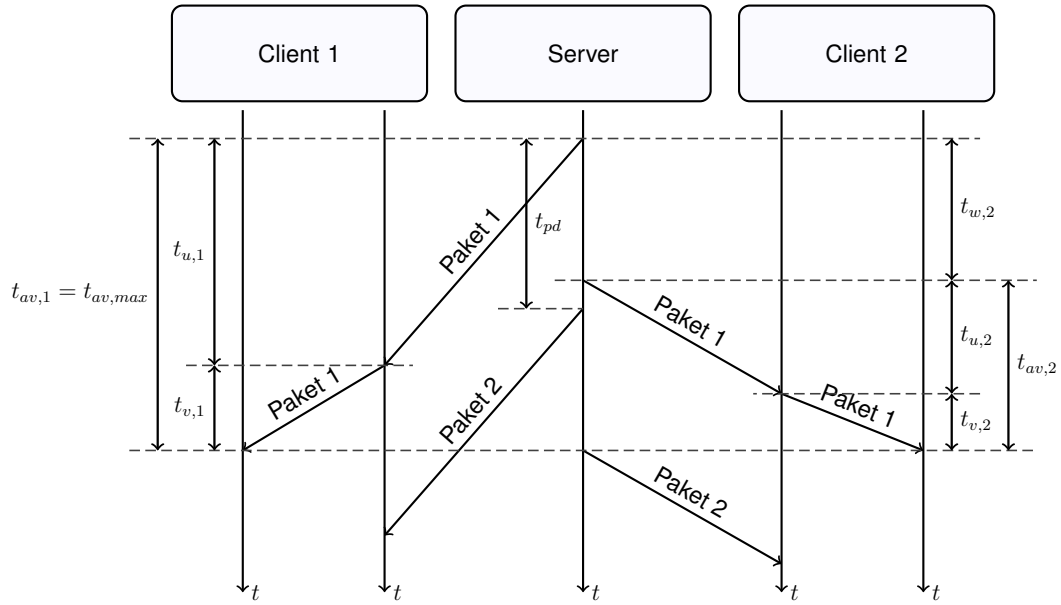



Abbildung 17: Streaming mit verzögertem Senden

Wurde das erste Audiopaket mit den richtigen Abständen an alle Empfänger gesendet, können ab dann die restlichen Pakete im Abstand der Periodendauer an die jeweiligen Clients gesendet werden.

### 8.4.3 Einschätzung

Werden für diese Variante gleichbleibende Übertragungslaufzeiten vorausgesetzt, ist eine Synchronisation des Streams möglich. Leider kann das Netzwerk in der Realität nicht für gleichbleibende Übertragungszeiten garantieren. Speziell für diesen Ansatz bedeutet das, dass sich die im Voraus gemessenen Abspielverzögerungen von denen beim Senden des ersten Audiopaketes unterscheiden können. Je kleiner die Schwankungen der Übertragungslaufzeiten eines Netzwerkes sind, desto kleiner ist auch die Differenz zwischen den Streams mehrerer Clients.

Der aktuell beschriebene Ansatz zeigt jedoch, dass Streams auch ohne Zeitsynchronisationsprotokolle synchronisiert werden können. Die Differenz zwischen den verschiedenen Ausgaben lässt sich durch die schwankenden Übertragungszeiten im Netzwerk bestimmen. Bei schlechten Netzwerkbedingungen können

diese erheblich steigen.

Ein Vorteil ist, dass die hauptsächliche Synchronisationsarbeit am Server geschieht. Die Clients müssen die Audiopakete nur Empfangen und Abspielen. Durch die Unicastverbindungen sind alle Clients vom Server aus erkennbar und individuell steuerbar. So können die berechneten Wartezeiten $t_w$ für die einzelnen Clients individuell angepasst werden. Das ist vor allem dann sinnvoll, wenn Abspielgeräte weit entfernt sind und der Ausbreitungsverzögerung des Audiosignals in der Luft entgegen gewirkt werden soll.

Einen ähnlichen Ansatz beschreibt die Firma Apple 2013 in einem ihrer Patente [13]. Dabei soll ein Medium an zwei verschiedenen, voneinander unabhängigen Kopfhörern über Bluetooth synchron abgespielt werden. Das Patent deutet auf die Apple AirPods[19] hin und geht besonders auf den Kopplungs- und Übertragungsvorgang ein.

Der Server[20] ermittelt die Übertragungslaufzeiten zu den einzelnen Kopfhörern und berechnet daraus die Differenz beider Verbindungen [vgl. 13, S. 27 Z. 21]. Bevor der Stream startet, wird der schnelleren Verbindung dieser Wert mitgegeben. Anders als in dem beschriebenen Ansatz wartet nicht der Server, sondern der Client die ermittelte Differenz ab, bevor ein Audiopaket abgespielt werden kann [vgl. 13, S. 21 Z. 58]. So ist es möglich, dass der rechte und linke Kopfhörer ohne Zeitsynchronisationsprotokoll, allerdings mit ähnlichen Problemen wie im lokalen Netzwerk das Medium synchron abspielt.

[19] kabellose Bluetooth-Kopfhörer der Firma Apple
[20] kann in diesem Fall Smartphone, Tablet, PC sein

# 9 Synchronisation während des Streamings

Die vorher erklärten Synchronisationsverfahren beschreiben ausschließlich den ersten Synchronisationsvorgang, bevor die Wiedergabe an allen Clients startet. Nun ist fraglich, ob dies ausreichend ist oder ob während einer Wiedergabe die Streams kontinuierlich neu synchronisiert werden müssen. Grund dafür können die unterschiedlichen Wiedergabegeschwindigkeiten sein, die durch die Ungenauigkeit der Quarzoszillatoren hervorgerufen werden. Es folgt eine Rechnung.

Für ein Medium, welches mit 44.1kHz abgespielt werden soll, wird ein Quarz, welcher mit 22.5792MHz taktet [vgl. 14, S. 2], für die Wiedergabe verwendet. Es wird angenommen, dass dieser eine typische Genauigkeit von $\pm$30ppm besitzt. Nun kann mit

$$\frac{30ppm}{1000000} \cdot 22579200Hz = 677.376Hz \tag{13}$$

die mögliche Abweichung pro Sekunde und mit

$$3600s \cdot 677Hz = 2437200Hz \tag{14}$$

die Abweichung nach einer Stunde für ein Gerät berechnet werden. Die Differenz eines Streams beträgt somit im Extremfall

$$\frac{2437200Hz}{22579200Hz} = 107.94ms \tag{15}$$

In dem Fall, dass zwei Geräte bis zu 60ppm auseinander liegen, würde die Differenz nach einer Minute bei 3.6ms und nach einer Stunde 216ms betragen.

Auch wenn hier der Extremfall angenommen wird und in der Praxis durch relativ gleich bleibende Temperaturen die Genauigkeit der Quarze besser ist, wird deutlich, dass eine kontinuierliche Synchronisation benötigt wird, um die durch Quarzoszillatoren hervorgerufenen Ungenauigkeiten zu beseitigen.

Trotz unterschiedlicher Synchronisationsverfahren muss es die Möglichkeit ge-

ben, die unterschiedlichen Differenzen auszugleichen. Natürlich kann nach einer gewissen Zeit die Wiedergabe an allen Geräten gestoppt und neu synchronisiert werden. Wesentlich besser ist jedoch eine Lösung, die während der laufenden Wiedergabe die einzelnen Streams synchronisiert. Dazu gibt es folgende Lösungen zur Anpassung.

## 9.1 Pakete wiederholen/überspringen

Ist die Differenz eines Streams am Client größer als die Zeit, die benötigt wird, um den Inhalt eines Paketes abzuspielen, können entsprechend viele Pakete gelöscht oder wiederholt werden. Hierbei leidet vor allem die Qualität. Zum einen werden wichtige Informationen ausgelassen, zum anderen gibt es Sprünge zwischen den einzelnen Audiosamples.

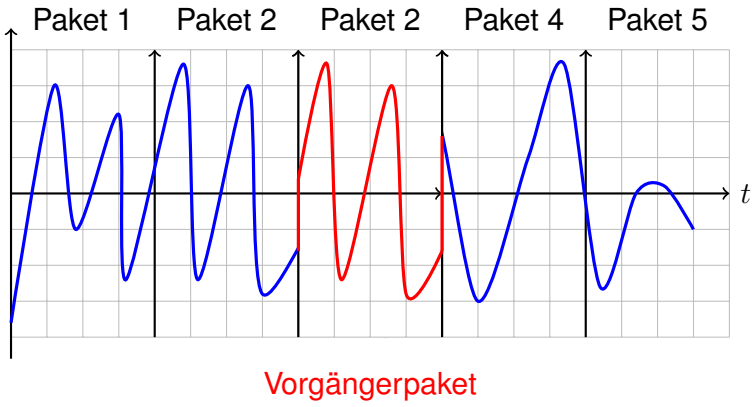



Abbildung 18: Signalverlauf mit wiederholtem Paket

Diese können durch Filter geglättet werden, was hier aber nicht näher betrachtet wird. Allein das Löschen und Wiederholen von Audiopaketen wird für eine korrekte Synchronisation nicht ausreichen. Mit größer werdenden Paketen steigt auch die Ungenauigkeit, da nur um die Wiedergabezeit des Paketes gesprungen werden kann.

## 9.2 Stream verzögern/beschleunigen

Zudem kann die Wiedergabegeschwindigkeit der einzelnen Pakete angepasst werden. Dazu wird die Abtastrate (Samplerate) so verändert, dass die einzelnen Audiopakete schneller oder langsamer abgespielt werden. Abhängig davon ob die Differenz des Streams zum tatsächlichen Abspielzeitpunkt positiv oder negativ ist, wird die Abtastrate gesenkt oder erhöht. So können kleinste Differenzen genau ausgeglichen werden. Die benötigte Abtastrate lässt sich durch die umgestellte Formel der Periodendauer berechnen.

$$\text{Abtastrate} = \text{Abschnittsgröße} \cdot \text{Periodendauer} \tag{16}$$

Durch das Anpassen der Abtastrate leidet die Audioqualität, weshalb diese korrekt konvertiert werden sollte.

# 10 Paketverlust

Wie bereits erwähnt stellt das Versenden von UDP Paketen ein Problem dar. Es gibt keinerlei Garantie, ob die Pakete auch tatsächlich am Client ankommen. Die Paketverlustrate kann durch Forward Error Correction erheblich gesenkt werden. Aber selbst mit diesem Fehlerschutzmechanismus ist nicht sicher, dass alle Pakete tatsächlich eintreffen.
Würden die verschiedenen Geräte die Audiopakete nacheinander ohne Rücksicht auf Verluste abspielen, ist die Gefahr, dass die Streams durch das Überspringen wichtiger Informationen auseinander laufen, sehr hoch. Somit muss für die Abspielzeit eines verlorenen Paketes etwas unternommen werden, damit die Streams synchron bleiben.

Die einfachste Variante ist, verlorene Pakete durch Nullpakete zu ersetzen. Dabei müssen die Nullpakete die gleiche Abschnittsgröße wie alle anderen Pakete haben. Je größer die verlorenen Audiopakete sind, desto größer ist auch die entstehende und wahrnehmbare Stille. Zudem entstehen an den Abschnittsübergängen unnatürliche Amplitudensprünge.

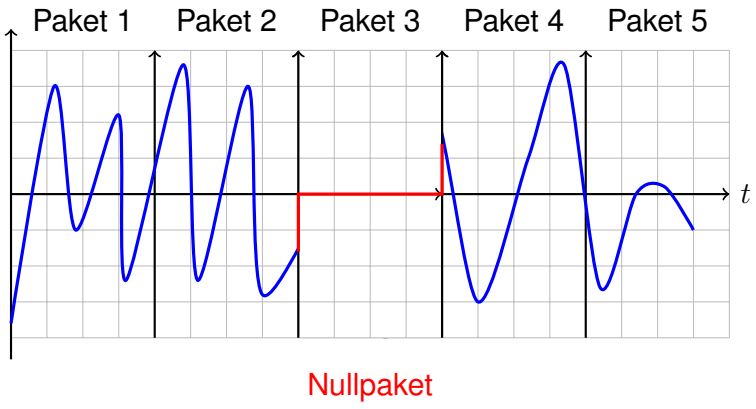



Abbildung 19: Signalverlauf mit Nullpaket

Diese Variante lässt sich verbessern, indem das letzte empfangene Paket noch einmal abgespielt wird. Durch die identischen Periodendauern ist dies grundsätzlich kein Problem. Obwohl keine Stille entsteht, ist dieser Ansatz nur bedingt bes-

ser. Der Verlust an Informationen wird ebenso auch durch Amplitudensprünge wahrnehmbar sein.

Die dritte Variante führt über die vorher empfangenen Audiopakete eine sinusförmige Analyse[21] durch. Das Verfahren wird *Packet Loss Concealmen* genannt. Ziel ist es, aus den bereits empfangenen Informationen ein neues Audiopaket zu erstellen, welches statt dem verlorenen Paket eingesetzt werden kann und den Verlust eines Paketes im Signalverlauf so gut wie möglich verschleiert. Längere Analysezeiten machen die Messungen genauer, während kürzere Analysen besser mit Signalschwankungen umgehen können. So ist ein gutes Mittelmaß für das Interpolieren der zeitabhängigen Sinusfunktion eines Ersatzpaketes zu wählen [4, S. 3].

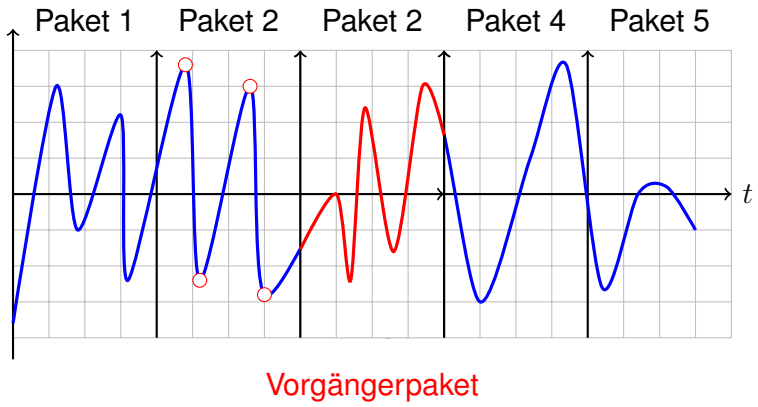



Abbildung 20: Signalverlauf mit sinusförmiger Analyse

Mit *Packet Loss Concealment* können verlorene Pakete bis zu einer Länge von 30 ms verschleiert werden. Ebenso kann damit auf Paketverluste von bis zu 30% reagiert werden. Das Verfahren wird unter anderem bei IP-Telefonie verwendet [1, vgl.].

[21] basierend auf der Fourier-Transformation

# 11 Praktische Tests

Im Folgenden soll das Audiosynchronisationsverfahren, welches die Audiopakete verzögert sendet, genauer betrachtet und untersucht werden. Es stellt sich die Frage, wie gut das Protokoll trotz genannter Probleme in der Praxis funktioniert. Die gewonnenen Ergebnisse sollen Aufschluss darüber geben, ob das Verfahren für den praktischen Gebrauch verwendet werden kann oder ob die Differenz der einzelnen Ausgangssignale zu hoch ist.

## 11.1 Umsetzung

Das Audiosynchronisationsverfahren wurde mit der Programmiersprache Python 2.7 umgesetzt.

Der Server besitzt zwei elementare Aufgaben. Zum einen muss sich dieser mit den einzelnen Clients verbinden und eine erste jeweilige Abspielverzögerung bestimmen. Zum anderen ist er für das verzögerte Senden der einzelnen Audiopakete an die verschiedenen Clients zuständig. Für jede Verbindung zu einem Client gibt es einen Thread, welcher die Pakete je nach Übertragungslaufzeit verzögert sendet.

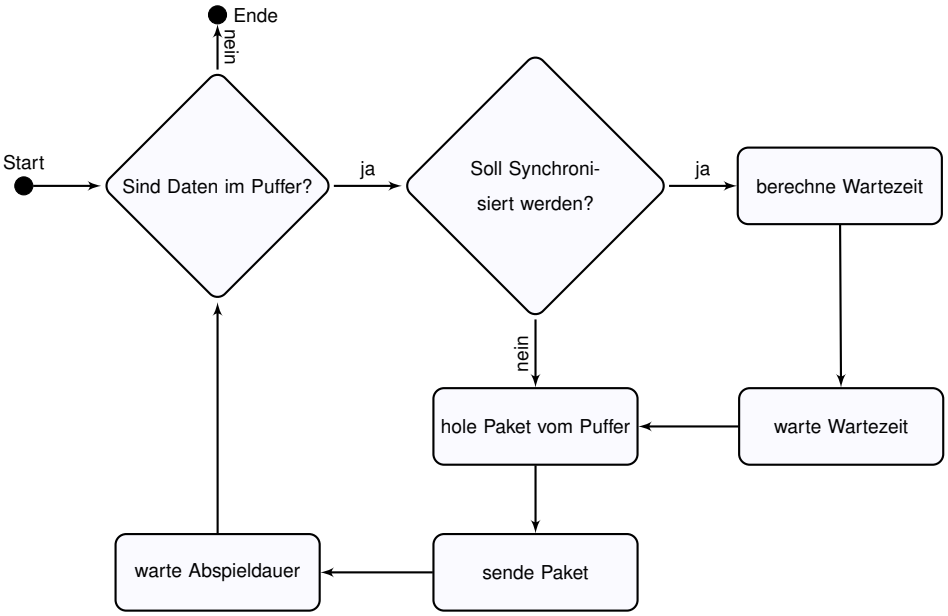



Abbildung 21: Programmablauf eines Threads für eine Verbindung

Die Aufgabe des Clients ist es, die empfangenen Pakete so schnell wie möglich wiederzugeben. Falls eine Anfrage für das Ermitteln der Abspielverzögerung empfangen wird, muss dieser seine doppelte Verarbeitungszeit abwarten, bevor eine Antwort zurückgesendet werden kann.

Bei den folgenden Versuchen wird ausschließlich die Differenz nach dem ersten Synchronisieren gemessen.

## 11.2 Messmöglichkeiten

Der Server sendet ein Audiosignal in Form eines Taktes oder ansteigenden Tones an die Clients. Im Aufnahmeprogramm sind die wiedergegebenen Signale durch jeweilige Kanäle in einem Amplitude-Zeit-Diagramm visuell dargestellt. Wenn die die Differenz anhand eines wiedergegebenen Taktes gemessen werden soll, muss der Abstand der aufgenommenen Schläge zwischen den Kanälen bestimmt werden.

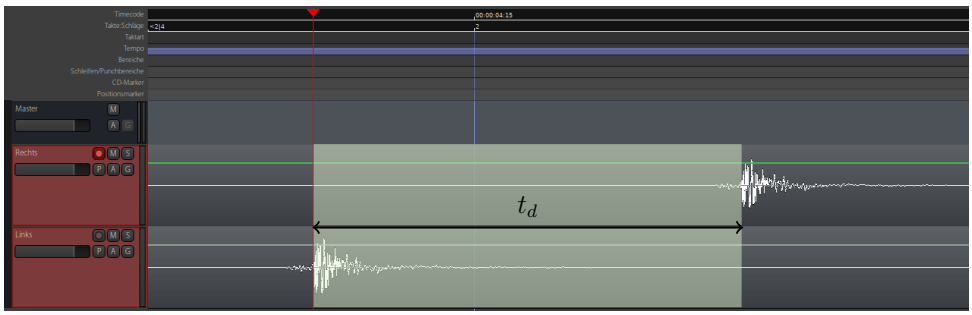



Abbildung 22: Versuchsaufbau - Takt

Wurde hingegen ein ansteigender Ton aufgenommen, muss der Wert ermittelt werden, der benötigt wird, um das Signal eines Kanals soweit zu verschieben bis beide Signale der Clients korrelieren. Dabei spielt nicht die Amplitude, sondern die aktuelle Periodenlänge der Frequenz eine Rolle.

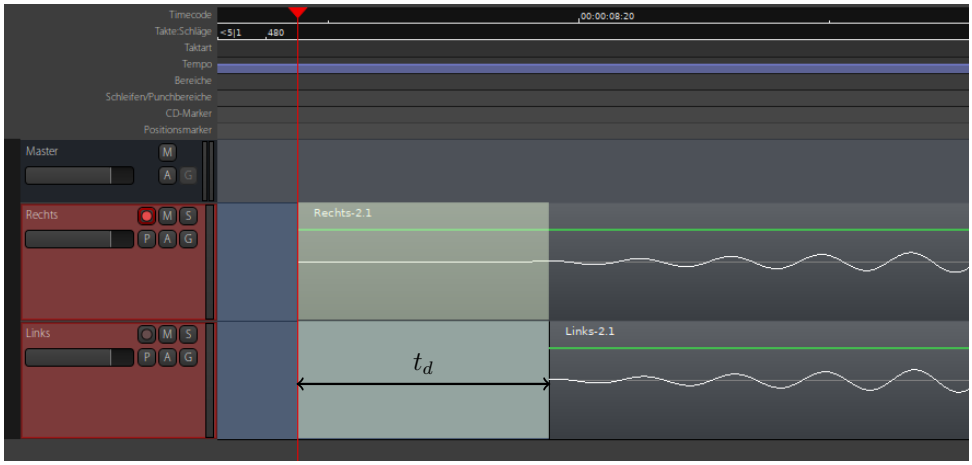



Abbildung 23: Versuchsaufbau - ansteigender Ton

Alle folgenden Messungen werden anhand eines abspielenden Taktes durchgeführt. Diese Variante wurde gewählt, da zum einen die Differenzen mit dem Aufnahme-Programm Ardour [22] leicht zu bestimmen sind, und zum anderen bei Aufnahmen mit dem Mikrofon die Wiedergabecharakteristiken der Lautsprecher durch unterschiedliche Amplituden im Aufnahmeprogramm irreführend sein könnten. Ardour löst die aufgenommenen Audiospuren mit bis zu 2,6 µs auf.

[22] professionelles freies Audioaufzeichnungsprogramm, basiert auf JACK

## 11.3 Ideale Bedingungen

Um das Protokoll unter idealen Bedingungen zu testen, müssen kurze, fehlerfreie Übertragungswege vorausgesetzt werden. Dazu werden beide Clients und der Server auf demselben Gerät ausgeführt. So sind die Übertragungszeiten gering und nahezu identisch. Dies gilt auch für die Verarbeitungszeiten der Clients.

Den Clients werden unterschiedliche Ausgabegeräte zugeordnet. Der Ausgang der Ausgabegeräte wird direkt wieder aufgenommen.

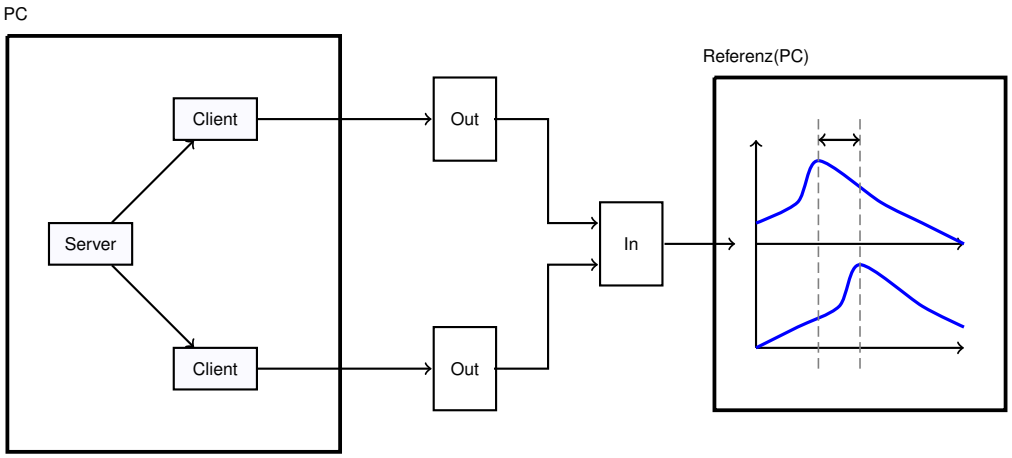



Abbildung 24: Alles in einem Rechner

## 11.4 WLAN

Es soll untersucht werden, wie genau das Protokoll mit zwei typischen Empfängern im WLAN funktioniert. Dazu werden für die folgenden Versuche als Empfänger zwei Raspberry Pi Zeros mit dem installierten Betriebssystem Raspbian Stretch Lite[23] verwendet. Neben einen 1GHz Single-Core Prozessor und 512 MB Arbeitsspeicher besitzen diese WLAN nach dem Standard IEEE 802.11[24]. Die Empfänger sind mit dem Funknetzwerk verbunden, während der Server ein Kabel verwendet.

[23] Release 2018-06-27, Kernel 4.14
[24] IEEE 802.11 ermöglicht Übertragungsraten von bis zu 54 Mbit/s

Der Server sendet ein Audiosignal an die Clients, welche die empfangenen Audiopakete abspielen. Beide Empfänger werden separat mit einem Mikrofon im Abstand von 5 cm abgenommen. Im Aufnahmeprogramm sind die Mikrofone als einzelne Kanäle sichtbar.

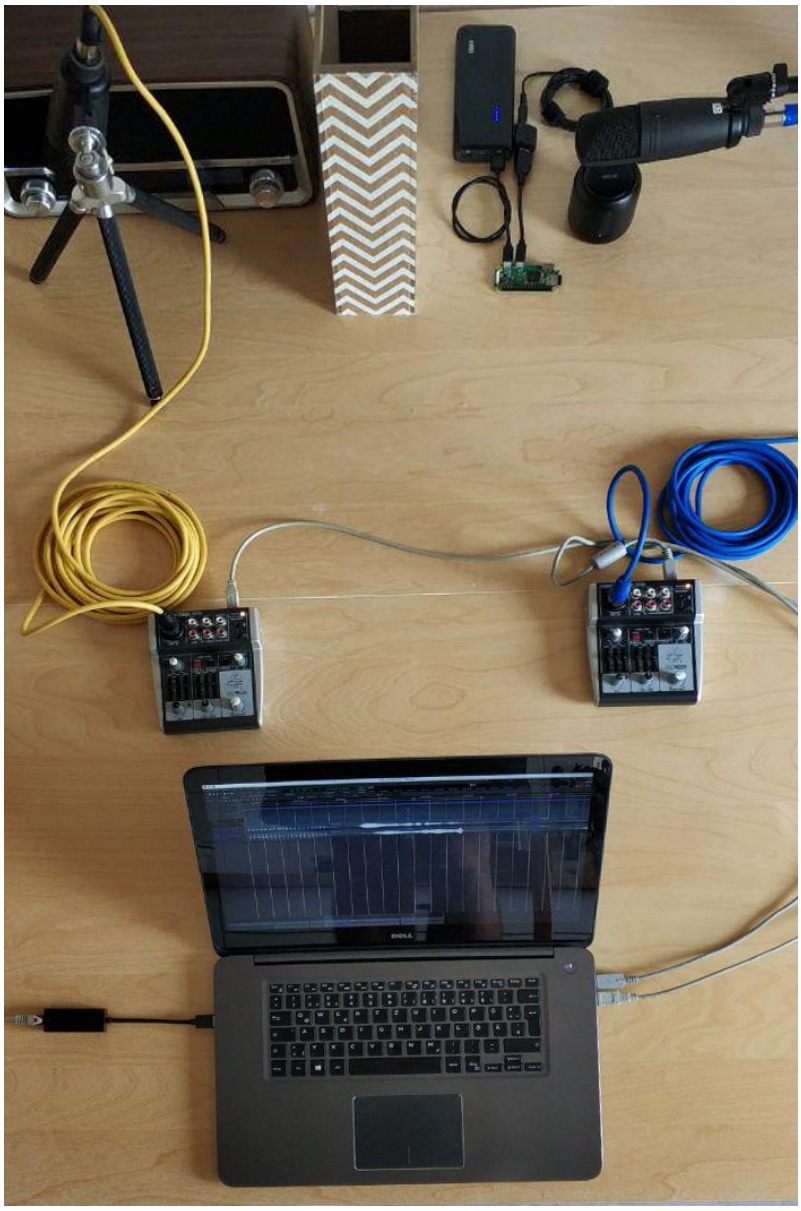


Abbildung 25: Versuchsaufbau

Anhand des sichtbar aufgenommenen Frequenzbandes lässt sich die zeitliche Verschiebung der beiden eingehenden Audiosignale erkennen und bestimmen.

## 11.5 Ergebnisse

Es wurden 20 Synchronisationen unter den oben genannten Bedingungen durchgeführt. Bei idealen Voraussetzungen wurde eine durchschnittliche Differenz von 424,2 µs erreicht. Die größte Abweichung betrug 748 µs. Wie zu erwarten war, sinkt die Genauigkeit unter Verwendung von WLAN. Bei 20 Versuchen wurde eine durchschnittliche Differenz der beiden Audiosignale von 744,2 µs erreicht. Die größte Abweichung betrug 1 ms.

Die erreichten Differenzen sind überraschend gering. Da die Werte kleiner als 1.5 ms sind, können mehrere Empfänger in einem Raum verwendet werden, ohne dass ein Qualitätsverlust entsteht. Das beantwortet auch die Frage, ob das getestete Audiosynchronisationsverfahren für eine praktische Anwendung geeignet ist.

## 11.6 Fehlerquellen / Probleme

Die gemessenen Abweichungen geben Aufschluss darüber, wie sehr die Übertragungslaufzeiten schwanken und wie ungenau die ermittelten Verarbeitungszeiten der Initialisierungsfunktion sind. Da die Differenzen gering sind, kann davon ausgegangen werden, dass die Übertragungslaufzeiten kaum schwanken und die ermittelten Verarbeitungszeiten nur kleine Abweichungen besitzen.

Die Initialisierungsfunktion, welche im Client die Verarbeitungszeit ermittelt, kann einen gewissen Fehler verursachen. Durch die unterschiedliche CPU-Auslastung und die damit verbundene Priorisierung kann sich die Verarbeitungszeit ständig ändern. Werden diese im Voraus genauer ermittelt und die CPU-Auslastung bleibt konstant, kann der Fehler für die Verarbeitungszeit minimiert werden.

# 12 Ergebnisse und Bewertung

In dieser Arbeit wurden verschiedene Audiosynchronisationslösungen vorgestellt. Dabei wurde vor allem auf die theoretische Funktionsweise Wert gelegt. Neben den vier vorgestellten Verfahren existieren noch weitere Ansätze, welche jedoch alle auf ähnliche Weise funktionieren. Für den Austausch der Daten und die Synchronisation der Geräte sollte das lokale Netzwerk verwendet werden.

Ausgehend von dem schnellstmöglichen Senden und dem Abspielen der Daten an den verschiedenen Clients ohne Synchronisationsprotokoll konnten Fehlerquellen und Probleme aufgezeigt werden, sowie Möglichkeiten, um diese zu beseitigen.
Die folgende Tabelle fasst alle thematisierten Synchronisationslösungen noch einmal zusammen.

| | **Transport** | **Verbindung** | **Zeitsynchronisation** | **Genauigkeit** |
|---|---|---|---|---|
| mit Offset | UDP, RTP | Multicast | NTP / PTP | < 50 µs [vgl. 10, S. 13] |
| Client als Synchronisationsquelle | UDP, RTP | Multicast | PTP | < 60.6 µs [17, S. 788] |
| Snapcast | TCP | Unicast | unbekannt | < 1 ms [21] |
| verzögertes Senden | UDP, RTP | Unicast | / | < 1 ms |

Tabelle 1: Vergleich der Audiosynchronisationsverfahren

Dabei ist interessant, dass alle Systeme, die eine Uhrensynchronisation voraussetzten, sehr präzise sind. Die Erkenntnis liegt darin, dass sich der Fehler eines Zeitsynchronisationsprotokolls wieder in der Differenz mehrerer Ausgabegeräte widerspiegelt. Je höher also die Anforderung an die Genauigkeit einer Audiosynchronisationslösung ist, desto wichtiger ist es, die Uhren möglichst genau zu synchronisieren. Das grundsätzliche Prinzip der Verfahren Snapcast, Client als Synchronisationsquelle und Verwendung eines Offsets ist gleich. Jeder Client besitzt einen Puffer, welcher mindestens so groß wie die längste Abspielverzögerung, bestehend aus Übertragungs- und Verarbeitungszeit, ist. Nur die Berechnung und

die Herangehensweise, um das empfangene Medium synchron abzuspielen, unterscheiden sich.

Es ist ebenso überraschend, dass der Ansatz ohne Zeitsynchronisation und mit verzögertem Senden der Audiopakete funktioniert. Der Fehler dieses Systems wird nicht durch ein Zeitsynchronisationsprotokoll, sondern durch schwankende Übertragungslaufzeiten während des Synchronisationsvorganges bestimmt. Diese sind jedoch schwierig zu beseitigen.
Im Vergleich zu den anderen Varianten ist dieses Verfahren etwas ungenauer. Dennoch überzeugt das Ergebnis. Erste Zweifel und Bedenken, dass diese Lösung in der Praxis nicht funktionieren könnte, wurden beseitigt.

Während der Untersuchung von verschiedenen Synchronisationslösungen stellte sich die Frage, ob kontinuierlich neu synchronisiert werden muss, oder ob eine Synchronisation zu Beginn ausreicht. Die Berechnungen ergaben, dass durch die Ungenauigkeit der verwendeten Oszillatoren die Streams auseinander laufen. Durch diskutierte Ansätze, welche die Geschwindigkeit der Streams anpassen, ließen sich die bereits gewonnenen Erkenntnisse erweitern. So gehört zu einer Audiosynchronisationslösung nicht nur der synchrone Start, sondern auch die kontinuierliche Kontrolle und Anpassung bei entstehenden Differenzen.

Alle Audiosynchronisationsverfahren erzielen niedrige Differenzen, sodass trotz unterschiedlicher Anforderungen und Möglichkeiten synchrones Audiostreaming funktioniert. Je kleiner diese sein sollen, desto größer werden Kosten und Aufwand, um ein Verfahren umzusetzen.

# 13 Schlussbemerkungen und Ausblick

Diese Arbeit stellte verschiedene Synchronisationslösungen vor. Je nach den Gegebenheiten und Anforderungen an die Genauigkeit kann das eine oder andere Verfahren seine Stärken ausspielen. Immer häufiger werden diese implementiert und finden nicht zuletzt auch in kommerziellen Produkten Anwendung. Besonders im Blick auf die Zukunft, wo das vernetzte Zuhause eine immer größere und wichtigere Rolle spielen wird, gelten diese Verfahren als richtungsweisend.

Momentan besitzen die meisten Produkte eigene Audiosynchronisationslösungen. So ist der Käufer gezwungen weitere Geräte des Herstellers zu kaufen, um Musik synchron abzuspielen. In der Zukunft ist aber auch ein Standard denkbar, der das Synchronisieren verschiedener Geräte und unterschiedlicher Hersteller ermöglicht.

Auch wenn die Arbeit zeigt, dass ein Medium ebenso ohne Zeitsynchronisationsprotokoll synchron abgespielt werden kann, ist es vorstellbar, dass in zukünftigen Ansätzen immer wieder auf synchronisierte Uhren aufgebaut wird. Dafür spricht die hohe Genauigkeit bei einfacher Umsetzung.

# Literaturverzeichnis


[1] VoIP Troubleshooter LLC a. Problem: Packet loss concealment, 27 Juli 2018. http://www.voiptroubleshooter.com.

[2] Dirk Mohl Andreas Dreher. Präzise uhrzeitsynchronisation – ieee 1588. 20 Seiten, Rev 1.2, Hirschmann Automation and Control GmbH Neckartenzlingen, Germany.

[3] Lars Immisch und Casper Wilstrup arsimmisch.github.io. Pcm terminology and concepts, 2017.

[4] Stefan Bruhn. Audio frame loss concealment, 5. April 2018. United States Patent US20180096691A1.

[5] J. Martin Ed. J. Burbank W. Kasch D. Mills, U. Delaware. *RFC 5905: Network Time Protocol Version 4: Protocol and Algorithms Specification*, 2010.

[6] Patrick Schnabel Elektronik-Kompendium.de. Soundkarte, 11. Mai 2018.

[7] Patrick Schnabel Elektronik-Kompendium.de. Qos - quality of service, 22. Juni 2018.

[8] Patrick Schnabel Elektronik-Kompendium.de. Ieee 802.3 / ethernet-grundlagen, 4. Juni 2018.

[9] R. Frederick V. Jacobson H. Schulzrinne, S. Casner. *RFC 3550: Request for Comments: 3550*, 2003.

[10] R. Frederick V. Jacobson H. Schulzrinne, S. Casner. An internet protocol (ip) sound system, 31 Oktober 2004.

[11] Tam Hanna. Standards: Kurz erklärt: Das precision time protocol - wie genau, Juni 2016. heise online.

[12] https://www.fairaudio.de. Samplingrate / samplingfrequenz / abtastrate, 11. Mai 2018.

[13] Renaud Lienhart Apple Inc Joakim Linde, Baek S. Chang. Synchronization of multi-channel audio communicated over bluetooth low energy, 27. Januar 2014. United States Patent US9712266B2.

[14] Eric Juaneda. 44.1 to 192khz audio clock, Dezember 2011. Rev 1.0, www.junilabs.com.

[15] Yhean-Sen Lai Kannan Rajamani and C. W. Farrow. An efficient algorithm for sample rate conversion from cd to dat, 10. Oktober 2000. IEEE SIGNAL PROCESSING LETTERS, VOL. 7.

[16] Michael Branicky Kendall Correll, Nick Barendt. Design considerations for software only implementations of the ieee 1588 precision time protocol.

[17] Timo Ojala Matti Hosio Aki Mäkivirta Mika Rautiainen, Hannu Aska and Niko Haatainen. Swarm synchronization for multi-recipient multimedia streaming, 2009. Department of Electrical and Information Engineering University of Oulu, Finland.

[18] D. Mills. *RFC 4330: Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI*, Juni 2006.

[19] Rüdiger Mosig. Synchronous play-out of media data packets, 9 Mai 2010. United States Patent US007675943B2.

[20] Johannes Pohl. Can i use rtp protocol to replace the tcp one?, 10 November 2017. GitHub - Forum.

[21] Johannes Pohl. Snapcast, 6. August 2018. GitHub.

[22] Eberhard Sengpiel. Haas-effekt und präzedenz-effekt (gesetz der ersten wellenfront), Dezember 2003. Tutorium.

[23] Eberhard Sengpiel. Praktische daten zur lokalisation von phantomschallquellen bei 'intensitäts'- und laufzeit-stereofonie, Januar 2009. F + A.

[24] Steven W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. 1997. Englisch, 626 Seiten, ISBN: 0966017633.

[25] Logica Banica Teodor Neagoe, Valentin Cristea. *2006 IEEE International Symposium on Industrial Electronics*. 9-12 Juli 2006. Englisch, ISBN: 1424404967.

[26] Gerhard Fettweis Tim Hentschel. Sample rate conversion for software radio, August 2000. IEEE Communications Magazine.

[27] Precise Time and Inc. Frequency. Ntp (network time protocol) and ptp (precision time protocol), 10 Oktober 2005.

[28] Sonos Inc. Timothy W. Sheen. Audio synchronization among playback devices using offset information, 21. Mai 2013. United States Patent US9313591B2.

[29] Deadlyhappen Wikipedia, OSI-Modell. Iso-osi-7-schichten-modell, 22. Juni 2018.

# Selbstständigkeitserklärung

Ich versichere hiermit, dass ich die Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

| | |
|---|---|
| 10. August 2018 | |
| **Datum** | **Ben Schönherr** |