DIPLOMARBEIT

Gateway zur Übertragung von Audiodaten von einem RTSP-Server in ein IEEE 1394-Netzwerk

Ausgeführt am
Institut für Computertechnik (E384)
der Technischen Universität Wien

unter der Anleitung von
o. Univ. Prof. Dipl.-Ing. Dr. Dietmar Dietrich
und
Dipl.-Ing. Manfred Weihs
als verantwortlich mitwirkendem Universitätsassistenten

durch
Mario Ruthmair
A-3142 Perschling, Thalheimerstraße 5
Matrikelnummer: 9826157
Studienkennzahl: E881

Wien, am 15. März 2006

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

< Ort, Datum > < Unterschrift >

Danksagung

An dieser Stelle möchte ich einigen Menschen danken, ohne deren Hilfe der Abschluss dieses Projekts und der vorliegenden Arbeit zwar möglich aber wesentlich schwieriger gewesen wäre. Zum einen wären da meine Eltern und Großeltern, die mich zu Beginn der Arbeit finanziell und auch in sonst jeder Hinsicht tatkräftig unterstützt haben. Weiters möchte ich Emmerich Zöchbauer, dem Chef der Firma Syncomp Data Systems, wo ich schon längere Zeit neben dem Studium beschäftigt bin, für die äußerst flexiblen Arbeitszeiten und die Bereitstellung von EDV-Equipment auch für private Zwecke beziehungsweise zur Erstellung dieser Diplomarbeit danken. Auch meinem Betreuer Dipl.Ing. Manfred Weihs bin ich wegen seiner Geduld, der schnellen Beantwortung dringender Fragen und der Versorgung mit entsprechender Literatur zu Dank verpflichtet.

Kurzfassung

Das Ziel dieses Projekts ist es, eine softwaretechnische Brücke zwischen zwei verschiedenen Kommunikationsstandards zu schaffen. Audiodaten werden dabei von einem beliebigen RTSP-Streaming-Server im Internet auf einen Gateway-Rechner übertragen, der zusätzlich über einen FireWire Host-Adapter mit einem IEEE 1394-Netzwerk verbunden ist. Dieses Gateway konvertiert die eintreffenden Audiodaten, die in verschiedensten Formaten kodiert sein können, in ein Datenformat, das vom Standard IEC 61883-6 definiert wird. Nach dieser Umwandlung werden die Audiosamples auf einem isochronen Kanal gemäß IEEE 1394 ausgesendet.

Der frei verfügbare Media-Player MPlayer wird aufgrund seiner breiten Unterstützung aktueller Datenformate und der regelmäßigen Updatezyklen für die Dekodierung der externen verschiedenartigen Audioquellen verwendet. Die Softwarekomponente, die die Verbindung mit dem IEEE 1394-Netzwerk herstellt, fungiert als eine Art Ausgabemodul für den MPlayer, sodass es auch möglich ist, es in zukünftige Versionen zu integrieren.

Das größte Problem in diesem Anwendungsfall ist die Tatsache, dass zwischen den beiden Standards große Unterschiede in den Echtzeiteigenschaften existieren, die zu Problemen bei der Synchronisation des Audiopuffers führen können. Wenn zum Beispiel die externe Verbindung zum RTSP-Server aus welchen Gründen auch immer langsamer wird, läuft der Datenpuffer aus, da im internen IEEE 1394-Netzwerk die Sendegeschwindigkeit weiterhin gleich bleibt. Deshalb muss die Synchronisationskomponente diese Abweichungen kompensieren, was in dieser Implementierung durch eine geringfügige Veränderung der internen nominellen Samplingfrequenz auf dem IEEE 1394-Bus erreicht wird.

Abstract

This project wants to build a software bridge between two different communication standards. Audio data is transferred from an RTSP streaming server anywhere in the Internet to a gateway machine, which is also connected via a FireWire host adapter to an IEEE 1394 network. The gateway converts the incoming audio data, which can be coded in various formats, to a data format compliant with the IEC 61883-6 standard. After conversion the audio samples will be sent out using an IEEE 1394 isochronous channel.

The open source media-player MPlayer is used for decoding the external heterogenous audio sources because of its wide support for current data formats and its frequent update cycles. The software part which connects to the IEEE 1394 network acts as a kind of output module of MPlayer, so it is possible to integrate it in future releases.

The biggest problem in this use case is the fact that there are great differences in the realtime characteristics of both standards which can lead to synchronization problems concerning the audio buffer. When the external link to the RTSP server for example gets slower, the data in the buffer is running out, because in the internal IEEE 1394 network the speed stays the same. So the sychronization engine has to compensate this divergency, in this implementation by smoothly varying the internal nominal sampling frequency on the IEEE 1394 bus.

Inhaltsverzeichnis

1 Einleitung

Die Motivation für das vorliegende Thema entstand ursprünglich aus dem schon lange existierenden Interesse an der Vernetzung von verschiedensten Komponenten zum Zwecke des Informationsaustauschs. Aufgrund der weiten Verbreitung und meinen beruflichen Tätigkeiten neben dem Studium hatte ich bis zum Beginn dieser Arbeit jedoch hauptsächlich mit dem Aufbau und der Wartung von Ethernet-Netzwerken zu tun. Auf der Suche nach einem geeigneten Diplomarbeitsthema wurde ich dann schließlich auf der Homepage des Instituts für Computertechnik an der technischen Universität Wien auf den Datenübertragungsstandard IEEE 1394 aufmerksam, der auch unter dem Namen FireWire[1] oder i.LINK[2] bekannt ist. Auch dieser Standard ermöglicht den Aufbau von kleinen Netzwerken mit zwar eingeschränkten Abmessungen aber doch interessanten Eigenschaften, wodurch auch schon eine gewisse Verbindung zum Grundprinzip des Ethernet-Standards hergestellt war. Um sich einerseits mit vertrauten als auch andererseits mit mir unbekannten Techniken beschäftigen zu können, wählte ich schließlich im April 2004 ein Thema, das es sich zum Ziel setzt, beide Netzwerkstandards direkt aneinander zu koppeln und einen Datenaustausch untereinander zu gewährleisten.

Präziser gesagt handelt diese Arbeit von der Übertragung von Audiodaten, die ein öffentlicher Server im Internet oder auch ein PC im lokalen Heimnetzwerk bereitstellt, in ein internes FireWire-Netzwerk, wo ein geeignetes Endgerät diese weiterverarbeiten und auch über eventuell angeschlossene Lautsprecher ausgeben kann. Um die beiden sehr unterschiedlichen Netzwerktechnologien miteinander verbinden zu können, wird ein Rechner eingesetzt, der mit beiden Schnittstellen ausgestattet und somit auch mit beiden Netzwerken verbunden ist. Dieser Gateway-Rechner nimmt entweder von sich aus oder bei bidirektionaler Kommunikation auf Anforderung eines FireWire-Geräts Kontakt mit dem Audioserver im externen Netzwerk auf und transferiert die Audiodaten basierend auf dem Standard RFC 791 (Internet Protocol) [5] in seinen Speicher. Anschließend werden die Daten, die oftmals mit effizienten und platzsparenden Algorithmen kodiert wurden, in ein einfacheres Format konvertiert, damit das Endgerät diese ohne größeren Aufwand auch weiterverarbeiten kann, und schlussendlich an die IEEE 1394-Schnittstelle und damit ins FireWire-Netzwerk gesendet.

Als Beispiel für ein geeignetes FireWire-Gerät sei ein Audiokomponentensystem von Sony namens LISSA genannt, das die über die integrierte IEEE 1394-Schnittstelle empfangenen, bestimmten Normen entsprechenden Audiodaten über den angeschlossenen Verstärker ausgeben kann. Somit ist es Musikanlagen, die zwar über eine FireWire-Schnittstelle, nicht aber über die Fähigkeit verfügen, eine IP-basierende Kommunikation aufzubauen, möglich, auf Audioquellen zuzugreifen, die sich normalerweise außerhalb der Reichweite des Standards IEEE 1394 befinden. Der entscheidende Vorteil des IP-Standards ist die Tatsache, dass alle ans Internet angeschlossene Rechner über dieses Protokoll miteinander kommunizieren, weshalb das Angebot an potentiellen

[1] Apple Computer, Inc., siehe http://www.apple.com
[2] Sony Corporation, siehe http://www.sony.net

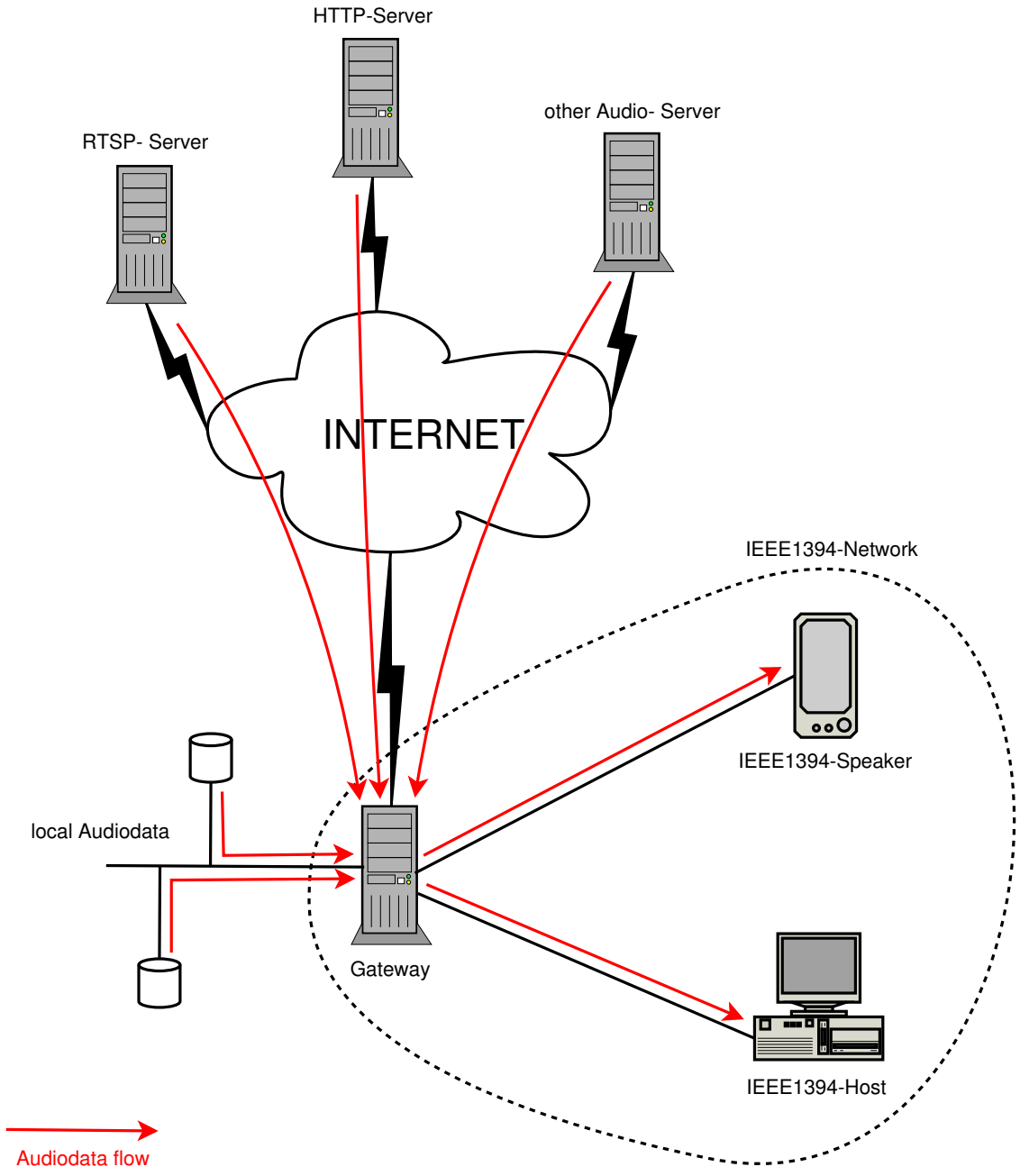

Abbildung 1: Funktionsweise des Gateways

Audioarchiven enorm groß ist. Das LISSA-System kann mit Hilfe dieses Gateways sowohl auf Internetradiosender als auch auf Audiodatenbestände zugreifen, die sich zum Beispiel im Laufe der Zeit auf Festplatten im lokalen Ethernet-Netzwerk angesammelt haben.

Ein aktuelles Audiogerät muss, um sich hoher Beliebtheit erfreuen zu können, eine ständig wachsende Vielfalt von Audioformaten unterstützen, damit es auch alle neu aufkommenden Audioquellen nutzen und wiedergeben kann. Nun ist es sehr schwierig respektive unmöglich, ein Gerät wie die Sony LISSA in dieser Hinsicht auf dem neuesten Stand zu halten. Der einzige Ausweg wären regelmäßige Firmware-Updates, deren Möglichkeiten aber durch den begrenzten Speicherplatz und die geringe Prozessorgeschwindigkeit stark eingeschränkt werden. Das vorliegende Projekt kann dieses Problem teilweise lösen, da hier der Gateway-Rechner die Konvertierung übernimmt und somit die ursprüngliche Kodierung der Audiodaten für die Endgeräte vollständig transparent macht. Dem Gateway selbst bleibt es aber trotzdem nicht erspart, sich die verschiedenen Übertragungsprotokolle und Algorithmen zur Dekodierung der Audioquellen anzueignen. In diesem Fall ist dies aber ein beliebiger PC mit dem Betriebssystem Linux, wodurch die Flexibilität und Erweiterbarkeit gegenüber reinen Embedded-Geräten deutlich gesteigert wird. Die Aufgabe der Formatumwandlung übernimmt hier der frei verfügbare Multimedia-Player MPlayer, der von einer großen Entwicklergemeinde laufend weiterentwickelt und perfektioniert wird und somit offen für neue Audioformate und Transferprotokolle ist.

Weiters soll gezeigt werden, dass der inzwischen wieder etwas aus dem Blickfeld geratene Standard IEEE 1394 schon seit seiner ersten Zertifizierung im Jahre 1995 Eigenschaften aufweist, die aufgrund aktueller Entwicklungen ansatzweise im IP-Standard nachgerüstet werden, um den Einsatz in bestimmten neuen Anwendungsfeldern zu ermöglichen. Die rasante Verbreitung der Voice-over-IP-Technik (VoIP) zum Beispiel bewirkte notwendigerweise, sich Gedanken über die garantierte und schnelle Übertragung der Audiodaten zu machen, sodass ein VoIP-Gespräch zwischen zwei oder mehreren Teilnehmern ohne lange Aussetzer oder sonstigen Verbindungsproblemen ablaufen kann. Die Voraussetzungen, um diese Probleme schon im Keim zu ersticken, liefert IEEE 1394 mit der isochronen Datenübertragung schon von Haus aus mit. Das soll nicht bedeuten, dass IEEE 1394 den IP-Standard vollständig ersetzen kann, da er unter teilweise zur IP-Basis sehr differierenden Bedingungen operiert und in manchen Aspekten wie der Reichweite stark eingeschränkt ist. Es soll nur in Erinnerung gerufen werden, dass auf bestimmte Anwendungssituationen bezogen einiges an Potential in diesem Standard steckt.

Abschließend wird noch kurz auf den grundlegenden Aufbau der vorliegenden Arbeit Bezug genommen. Im ersten größeren Teil, dem Kapitel 2, wird hauptsächlich auf die für dieses Projekt relevanten Grundlagen von Audioformaten und -kodierungsalgorithmen, des verwendeten internationalen Standards IEEE 1394 und des Echtzeitprotokolls RTP eingegangen. Das darauf folgende Kapitel 3 beschreibt das zentrale Problem und die Ausgangssituation dieser Arbeit, sowie die grundlegenden Entscheidungen, die getroffen wurden, um die gesetzten Ziele zu erreichen. Kapitel 4 zeigt die Lösungsansätze und deren detaillierte Ausführung. Im Kapitel 5 werden noch offene Probleme aufgezeigt und eventuell mögliche Lösungswege dafür kurz umrissen. Der letzte

Teil 6 fasst die Arbeit, gewonnene Erkenntnisse und erreichte Resultate zusammen und versucht, das Thema abzurunden.

2 Grundlagen

Dieses Kapitel beginnt ohne Anspruch auf Vollständigkeit mit den Definitionen einiger oft verwendeter Begriffe in dieser Arbeit. Ein gewisses Basiswissen in der Informatik wird für das Verständnis der folgenden Texte jedoch vorausgesetzt. Weiters werden in diesem Teil die verwendeten mathematischen, informationstechnischen und stilistischen Konventionen beschrieben und klargestellt. Da es in diesem Projekt um die Übertragung von Audiodaten geht, werden nach den Definitionen kurz einige Audioformate und deren besondere Eigenschaften beschrieben, um dem Leser eine der Hauptaufgaben des Gateway-Programms näherzubringen, und zwar die Konvertierung der Audiodaten vom ursprünglichen Format in das simplere Zielformat. Nach diesem Unterkapitel werden die für diese Arbeit grundlegenden Eigenschaften des Standards IEEE 1394 und des Streaming-Protokolls RTP beschrieben, um für das Kapitel 3 vorzubereiten, indem auf das Projektziel - die Entwicklung einer Brücke zwischen den beiden Standards - eingegangen wird. Nach den Grundlagen folgt noch eine Gegenüberstellung der beiden Welten, in der die gravierenden Unterschiede anhand von verschiedenen Bewertungskriterien aufgezeigt werden sollen.

2.1 Konventionen

Wenn in der vorliegenden Arbeit der Begriff „Gateway" allein verwendet wird, dann ist damit die Summe der Softwarekomponenten dieses Projekts gemeint, die für den Datenfluss zwischen den beiden Technologien IEEE 1394 und IP verantwortlich sind, unabhängig von der eingesetzten Hardware. Die allgemeine Definition dieses Begriffs kann in 2.2 nachgelesen werden.

Die Grafiken und Diagramme in dieser Arbeit wurden den jeweils erwähnten Quellen teilweise nicht direkt entnommen, sondern aus Gründen der Qualität detailgetreu nachgebildet. Dadurch wurden unschöne Treppeneffekte, die durch Konvertierungen und Veränderungen der Größe entstanden wären, vermieden. Alle Begriffe innerhalb der Grafiken sind in englischer Sprache gehalten, einerseits aufgrund der Vorgaben der Quellen und andererseits um die internationale Verständlichkeit der selbst entworfenen Diagramme zu erhöhen.

Verweise auf andere Kapitel und genauere Informationen zu bestimmten Schlüsselwörtern werden in Fußnoten untergebracht, wenn sie den Lesefluss unterbrechen würden. Falls ein Verweis jedoch Bestandteil eines grammatikalisch korrekten Satzes ist, wird er, genau wie Zitate, sofort im laufenden Text benannt.

Bei Größenangaben von Datenmengen oder Übertragungsgeschwindigkeiten stehen die offiziellen Abkürzungen „Byte" für 1 Byte und „Bit" für 1 Bit. Die gebräuchlichsten Vorsilben für die kurze Beschreibung größerer Mengenangaben oder Frequenzen lauten „k" für den Faktor 10^3, „K" für 2^{10}, „M" für 10^6 und „Mi" für 2^{20}.

Wertangaben in anderen als dem dezimalen Zahlensystem werden von einem charakteristischen Kleinbuchstaben gefolgt. Die Darstellung 1010b bedeutet somit die dezimale Zahl 10 im binären System und 3Fh im Hexadezimalsystem entspricht dem dezimalen Wert 63.

Die Algorithmen beinhalten entweder Kommandos, die direkt auf der Kommandozeile unter Linux eingegeben werden können oder Bestandteil einer Konfigurationsdatei oder eines Makefiles sind. Nicht zur Ausführung dieser Befehle gehörige Kommentare werden im Algorithmus hinter dem Rautensymbol # angefügt.

2.2 Definitionen

API (Application Programming Interface): Ein API bezeichnet eine Programmierschnittstelle, mit der der Entwickler auf Teile eines anderen Softwaresystems oder des Betriebssystems zugreifen kann. Im API ist detailliert spezifiziert, wie und mit welchen Parametern dieser Zugriff durchgeführt wird.

Broadcasting: Wenn ein Host Broadcasting betreibt - sowohl auf IP-Netzwerke als auch auf andere Arten von Netzwerken bezogen -, erreichen diese Informationen im Gegensatz zum Multicasting alle Teilnehmer in diesem Verbund, ob sie sich für diese Daten interessieren oder nicht.

Codec: „Als Codecs (englisches Akronym aus coder und decoder) bezeichnet man Verfahren bzw. Programme, die Daten oder Signale digital codieren und decodieren. [...] Meistens werden beim Codiervorgang die analogen Signale nicht verlustfrei digitalisiert, sondern es werden eine Dynamikreduktion des analogen Signals sowie eine Datenkompression des digitalen Signals vorgenommen." [32] Häufig wird dieser Begriff in Verbindung mit Multimedia-Daten genannt, da es aufgrund der Größe der Rohdaten notwendig ist, den ursprünglich aufgenommenen Datenstrom so gut wie möglich zu komprimieren, um die Daten den aktuell verbreiteten Speichermedien anzupassen. Auch für die schnelle Übertragung dieser Daten über das Internet müssen aufgrund der derzeitig geringen Bandbreiten effiziente Verfahren zur Minimierung der Datenmenge angewandt werden.

DLL (Dynamic Link Library): Unter dem Betriebssystem Microsoft Windows werden oft verwendete Programmroutinen in DLL-Dateien gespeichert und damit allen installierten Programmen zur Verfügung gestellt. Damit wird vermieden, dass die selben Programmsequenzen mehrmals im System vorkommen und damit unnötigen Speicherplatz belegen.

Echtzeit (Realtime): Eine Berechnung geschieht in Echtzeit, wenn ihr Ergebnis innerhalb eines gewissen Zeitraumes (dieser entspricht einem Zeitintervall der realen Welt) garantiert vorliegt, das heißt bevor eine bestimmte Zeitschranke erreicht ist. Man unterscheidet dabei zwischen harter und weicher Echtzeit. Bei einem harten Echtzeitprozess ist die vorgegebene Zeitschranke verpflichtend, falls er diese nicht erreicht, können ernste Schäden im System auftreten. Die Einhaltung der Frist bei weicher Echtzeit wird zwar empfohlen, jedoch ist das Resultat nicht völlig wertlos, wenn es zu spät ankommt. Auch die Stabilität des Systems wird in diesem Fall nicht weiter negativ beeinflusst. [25, S.429-430]

Event: Besteht eine Aufnahme aus mehreren Kanälen (z. B. Stereo bei Audioaufnahmen), dann existieren zu einem Zeitpunkt mehrere Samples. In weiterer Folge wird in dieser Arbeit der Begriff

Event für die Gesamtheit aller Samples in einem Zeitpunkt verwendet.

FIFO: FIFO (First In - First Out) bezeichnet ein Prinzip der Zwischenspeicherung von Daten, bei dem diejenigen Elemente, die zeitlich zuerst in den Speicherbereich aufgenommen wurden, auch zuerst wieder entnommen werden. Man kann dieses Prinzip vereinfachend mit einer Warteschlange vergleichen: „Wer zuerst kommt, mahlt zuerst."

Gateway: Ein Gerät wird Gateway genannt, wenn es zwei oder mehr verschiedene Bereiche miteinander verbindet und eine Kommunikation zwischen ihnen ermöglicht. In diesem Projekt besteht die Aufgabe des Gateways darin, den Standard IEEE 1394 mit einem Ethernet-Netzwerk zu koppeln, wobei die Daten in der derzeitigen Implementierung nur in eine Richtung fließen, und zwar vom IP- ins FireWire-Netzwerk.

isochron: Im Gegensatz zur asynchronen Datenübertragung, bei der Daten nur nach Bedarf und eventuell mit Bestätigung des Empfängers gesendet werden, gibt es bei isochronen (Übersetzung: gleich lang dauernd) Transfers ein bestimmtes Zeitintervall, in dem mit einer präzisen Regelmäßigkeit Daten von der Quelle zur Senke geschickt werden.

MPlayer: MPlayer bezeichnet eine Wiedergabesoftware für Multimediadaten, die kostenlos unter Berücksichtigung der General Public License (GNU GPL [34]) auf [35] zum Download bereitsteht. Nähere Informationen zum Thema MPlayer werden im Kapitel 3.5 behandelt.

Multicasting: In IP-Netzwerken spricht man von Multicasting, wenn ein Sender Informationen an mehrere Empfänger schickt, die sich aber zuvor bei dieser Multicast-Gruppe registrieren müssen, um an die Daten zu gelangen. Die zuständigen Router müssen die Multicasting-Pakete entsprechend vervielfachen und zu den jeweiligen Empfängern weiterleiten.

NAT (Network Address Translation): Dieses Feature wird in Routern in IP-Netzwerken eingesetzt, um ein komplettes internes Netzwerk über eine offizielle IP-Adresse ans Internet anzubinden. Sobald ein Host im privaten Netzwerk ein Paket ins Internet schicken will, wird die Absenderadresse durch die öffentliche IP-Adresse ersetzt, denn sonst würde das Antwortpaket im Nirvana landen, da die private Adresse nicht von außen erreichbar ist. Um die Antwort auch wieder dem richtigen internen Host zuordnen zu können, wird in der Anfrage auch der Quellport geändert, den der Router zusammen mit der internen IP-Adresse in der sogenannten NAT-Tabelle verwaltet.

Node: Ein Node bezeichnet ein Gerät am IEEE 1394-Bus. Jeder Node bekommt durch den automatischen Prozess der Adressierung des IEEE 1394-Standards eine eindeutige Identifikationsnummer innerhalb dieses Netzwerks, wodurch eine klar definierte Kommunikation untereinander möglich wird.

Puffer: Wenn zwei Prozesse, die in ihrer Ausführung zueinander asynchron laufen, miteinander kommunizieren und Daten austauschen wollen, werden diese Daten oft in einem dafür reservierten Speicherbereich zwischengespeichert, um dann zu beliebiger Zeit weiterverarbeitet werden zu können. Damit müssen sich die Prozesse nicht um das Timing und die Zeiten der Sende- und Empfangsbereitschaft des jeweils anderen kümmern. Es existieren mehrere Implementierungen

dieser Puffer, in diesem Projekt wird ein sogenannter Ringpuffer verwendet, der in 4.4.3 näher beschrieben wird.

QoS (Quality of Service): Isochrone Transfers in IEEE 1394-Umgebungen bieten zum Beispiel die Garantie, dass die Daten rechtzeitig beim Empfänger ankommen. Gerade bei Echtzeitanwendungen, bei denen es von ihrer Natur her zwingend notwendig ist, dass die Informationen innerhalb eines bestimmten Zeitintervalls beim Kommunikationspartner eintreffen, ist es wichtig, dass das Übertragungsmedium und alle anderen Komponenten zwischen den beiden kommunizierenden Geräten diese Garantie bieten, und das nennt man Quality of Service.

Quadlet: Ein Quadlet ist ein Datenfeld, das vier Bytes umfasst und in vielen Bereichen des IEEE 1394-Standards die kleinste Einheit darstellt.

Reverse-Engineering: Unter diesem Begriff versteht man die Tätigkeit, die ein Programmierer durchführt, um nur mit Hilfe der ausführbaren Dateien eines Programms die internen Abläufe während der Exekution zu verstehen, ohne den Quellcode zu kennen. Bei kommerzieller Software, die nur in der Endfassung vorliegt, wird diese Strategie oft angewandt, um zu den vom Hersteller geschützten Algorithmen zu gelangen.

Router: Diese Komponente in einem IP-Netzwerk regelt aufgrund der Zieladresse, die im Header eines IP-Pakets angegeben ist, die Weiterleitung der Daten an andere Router oder an den Empfänger selbst.

Sample: Bei einer Aufnahme beziehungsweise Messung eines Signals werden dessen Amplituden mit einer bestimmten Frequenz abgetastet, der sogenannten Samplingfrequenz oder Samplingrate. Eine dieser gemessenen Amplituden wird Sample genannt und zum Beispiel in einer 16 Bit langen Zahl gespeichert. Beträgt die Auflösung der Abtastung mehr als diese 16 Bit, kann genauer zwischen den einzelnen Messwerten unterschieden werden, wodurch die Aufnahme detailreicher wird und präziser dem Original entspricht. Nachteilig wirkt sich die Erhöhung der Auflösung aber auf den rapide steigenden Speicherbedarf aus.

Samplingrate: Die Samplingrate oder Samplingfrequenz bezeichnet die Abtastfrequenz bei der Aufnahme einer Audioquelle. Typischerweise werden folgende Werte in der Consumer-Electronics- und der Computerbranche verwendet: 32 kHz, 44,1 kHz, 48 kHz, 96 kHz und 192 kHz.

(Live-)Streaming: Die kontinuierliche Übertragung von Daten - meistens Audio- oder Videodaten - bezeichnet man als Streaming, wie zum Beispiel die Wiedergabe von Radiosendungen aus dem Internet. Hier spricht man zusätzlich von Live-Streaming, da die einzige Zeitverzögerung zwischen der Aufnahme des Audiosignals im entfernten Radiostudio und der Wiedergabe auf lokalen Lautsprechern von der Konvertierungs- und Übertragungszeit vom Sender zum Empfänger definiert wird.

Switch: Der Switch stellt als zentraler Punkt die Verbindung zwischen Hosts im IP-Netzwerk her und kann in leistungsfähigeren intelligenteren Ausführungen den Datenverkehr auch filtern beziehungsweise die Weiterleitung regeln.

Zeiger: Eine Zeigervariable ist ein programmiertechnisches Konstrukt, bei dem nur eine Speicheradresse in der Variable selbst festgehalten wird. Diese Adresse verweist dann auf den tatsächlichen Wert des anfangs deklarierten Typs.

2.3 Audioformate

Die aufgenommenen Audiosamples[3] werden vor dem Abspeichern auf einem Medium oft in ein bestimmtes Datenformat, das heißt in eine bestimmte Dateistruktur, konvertiert. Für diese Aufgabe wird ein sogenannter Codec[4] herangezogen, der, je nachdem, wofür er programmiert wurde, grundsätzlich zwei Arten von Audioformaten erzeugen kann, die verlustfreien und die verlustbehafteten Formate. Rohe Audiodaten können, je nach Samplingrate, Kanalanzahl und Sampleformat, sehr schnell enorme Speichermengen verschlingen, deswegen werden Wege gesucht, die Daten effizient zu komprimieren und den Speicherverbrauch somit zu verringern.

Ein zweites sehr wichtiges Unterscheidungsmerkmal in der breiten Palette der Audioformate ist der Grad der Einsicht in die technischen Details und Vorgänge der Codecs. Viele proprietäre Formate werden von den Herstellern nicht offengelegt und dadurch für die Programmierer von Multimedia-Playern, die nicht unbedingt teure Abgaben an die Eigentümer leisten wollen, zu teilweise unüberwindbaren Hürden. Aus diesem Grund ist der Benutzer oft dazu gezwungen, für ein bestimmtes Format die Software des jeweiligen Herstellers auf seinem System zu installieren, die teilweise kostenpflichtig oder mit Werbeeinblendungen versehen ist.

Was alle Audioformate jedoch gemeinsam haben, ist die Tatsache, dass die Audiodaten für die akustische Ausgabe durch Lautsprecher wieder in die ursprüngliche Datenstruktur konvertiert werden müssen, das heißt, die Soundkarte oder der Verstärker einer HiFi-Anlage kann nur die rohen unkomprimierten Samples interpretieren. Auch für diesen Umkehrungsprozess, der in jeder Multimedia-Player-Software integriert sein muss, ist es natürlich unerlässlich, die Interna der verschiedenen Codecs zu kennen.

2.3.1 Verlustfreie Formate

Um zu den verlustfreien Codecs zählen zu dürfen, darf beim Konvertieren der Rohdaten in das jeweilige Format keinerlei Information der originalen Aufnahme verloren gehen. Die unkomprimierten Mitglieder dieser Gruppe sind am simpelsten gestrickt, da sie die Audiosamples einfach in eine bestimmte Dateistruktur verpacken und sie mit Zusatzinformationen, wie Samplingrate, Kanalanzahl, Sampleformat und Beschreibung, versehen. Dadurch werden die Audiodaten aber nur in eine gewisse Ordnung gebracht, sodass sie für jemanden, der kein Wissen über die ursprünglichen Aufnahmeparameter besitzt, ebenfalls interpretierbar sind. Der Speicherbedarf steigt

[3] siehe Kapitel 2.2 auf Seite 14: Sample
[4] siehe Kapitel 2.2 auf Seite 14: Codec

ob dieser zusätzlichen Metainformationen sogar noch an. Vertreter dieser Sparte sind zum Beispiel AIFF[5], WAV, das aber in seltenen Fällen auch als Containerformat für komprimierte Daten fungieren kann, oder das wahrscheinlich weltweit am weitesten verbreitete Format CompactDisc Digital Audio, das auf jeder handelsüblichen CD zu finden ist. Die Codecs, die die eben erwähnten Datenstrukturen erzeugen, und deren detaillierte Abläufe stellen ob ihrer Einfachheit für Programmierer von Multimedia-Software kein großes Geheimnis dar und finden deshalb vollständige Unterstützung in vielen Softwarepaketen. Dennoch sinkt die Relevanz dieser Formate mit hoher Beständigkeit, da mittlerweile weitaus bessere Algorithmen in diesem Bereich existieren.

Durch geeignete Kompressionsverfahren erreichen einige Formate eine höhere Kompaktheit der Daten und können gleichzeitig für den Erhalt der gesamten Information garantieren. Dabei sind diesen Verfahren, was die Effizienz der Speichergewinnung betrifft, natürlich Grenzen gesetzt, die von den verlustbehafteten Codecs durchbrochen werden können. Beispiele für die komprimierenden verlustfreien Formate sind Windows Media Audio Lossless (ein Unterformat der Gruppe Windows Media Audio mit Dateiendung .wma) oder Free Lossless Audio Codec (.fla, .flac).

2.3.2 Verlustbehaftete Formate

Für die meisten Anwendungsfälle und vor allem für Privatanwender besitzt die Einsparung von Speicherplatz weit höhere Priorität als der präzise Erhalt aller Informationen. Die verlustbehafteten Formate, deren Hauptaufgabe es ist, effiziente Kompressionsverfahren einzusetzen, sind deshalb die weitaus wichtigeren. Deren Zukunft kann durch die ständig größer werdende Verbreitung als gesichert betrachtet werden.

Bei aktuellen Methoden der Audiokompression werden zusätzlich zu den mathematischen Algorithmen viele Schwächen der menschlichen Hörleistung ausgenutzt, um die Informationsmenge zu verringern. Wenn zum Beispiel ein leiser Ton kurz vor oder nach einem lauten Ton wiedergegeben wird, ist er für den Großteil der Menschen nicht hörbar. Genauso können Frequenzen, die außerhalb des hörbaren Spektrums liegen, herausgefiltert werden. Ob Datenreduktionen dieser Art wahrgenommen werden können, ist nur subjektiv von jedem einzelnen entscheidbar, da die akustische Empfindsamkeit von Individuum zu Individuum sehr stark variieren kann. [33]

Naturgemäß benötigt ein effizienteres Format mit komplexeren Kompressionmethoden auch mehr Prozessorleistung bei der Verarbeitung dieser Audiodaten. Zwar ist das Kodieren der Originaldaten weit diffiziler und rechnerisch aufwendiger, aber auch beim Dekodieren der komprimierten Daten zur Weitergabe an die Sound-Hardware müssen genügend Leistungsressourcen zur Verfügung stehen, um den Vorgang in Echtzeit[6] durchführen zu können.

Bei den verlustbehafteten Formaten kommt auch der Unterschied zwischen vom Hersteller gehüteten und freien Codecs deutlicher zum Tragen, da aufgrund der hohen Komplexität es schwer

[5] Audio Interchange File Format
[6] siehe Kapitel 2.2 auf Seite 14: Echtzeit

möglich ist, den Codec mittels Reverse-Engineering[7] nachzubauen. Beispiele für geschützte proprietäre Formate sind RealAudio (.ra) und Windows Media Audio (.wma), für freie standardisierte Formate MPEG 1 Layer 3 (.mp3), Advanced Audio Coding (.aac, .mp4. m4a) und OGG Vorbis (.ogg).

2.4 IEEE 1394

Der Standard IEEE 1394 [2] entstand aus der Technologie FireWire, die von Apple Computer in den 80er Jahren entwickelt wurde. Aufgrund des steigenden Interesses wurde diese Datenübertragungstechnik dann schließlich 1995 vom IEEE in Form eines internationalen Standards publiziert. Wegen unterschiedlicher Interpretationen dieses Standards durch die Hardwarehersteller entstanden Kompatibilitätsprobleme zwischen den IEEE 1394-Geräten, weshalb im Jahr 2000 die Spezifikationen durch den Zusatz IEEE 1394a [3] korrigiert und teilweise auch mit neuen Features erweitert wurden [24]. 2002 folgten im Standard IEEE 1394b [4] weitere Neuerungen, die jedoch weitaus umfangreicher waren als diejenigen im ersten Zusatz und vor allem die Erhöhung der Reichweiten und der Datendurchsatzraten betraf. Geräte, die IEEE 1394b unterstützen, befinden sich zwar schon vereinzelt auf dem Markt, aber sie reizen den Standard vor allem in Hinblick auf die Übertragungsrate noch bei weitem nicht aus.

Die hauptsächlichen Anwendungsgebiete des Standards IEEE 1394 findet man bei der Verbindung von Multimediageräten, wie zum Beispiel Videokameras, professionelle Audiohardware, MP3-Player, DVB-Hardware, externe Speichermedien (Festplatten) und Computersysteme im allgemeinen. Ähnlich dem USB-Standard [21] spielt IEEE 1394 seine Stärken im Heimbereich aus, das heißt bei der Kommunikation von Geräten über kurze Distanzen hinweg, auf denen sehr hohe Datenraten erzielt werden können. Durch die isochrone Datenübertragung wird Quality of Service[8] ermöglicht, womit dieser Standard prädestiniert für Echtzeitanwendungen wie das Streaming von Multimediadaten ist. Der Konfigurationsaufwand für die einzelnen Knoten (Nodes) wird minimal gehalten, da sich die Adressverteilung und hierarchische Ordnung innerhalb des IEEE 1394-Busses selbst organisiert und diese Fähigkeiten bereits in die Hardware verankert wird.

In folgendem Teil sollen dem Leser die Grundlagen des IEEE 1394-Standards nähergebracht werden, ohne dabei zu tief ins Detail zu gehen, da dies in anderen Arbeiten beziehungsweise der Spezifikation des Standards [2] selbst schon zur Genüge getan wurde und nicht unbedingt notwendig für das Verständnis der Vorgänge im vorliegenden Projekt ist. Wenn in weiteren Abschnitten dieser Arbeit der Begriff IEEE 1394 verwendet wird, dann soll dies als Überbegriff für den ursprünglichen Standard und die beiden Zusätze verstanden werden.

[7] siehe Kapitel 2.2 auf Seite 14: Reverse-Engineering
[8] siehe Kapitel 2.2 auf Seite 14: Quality of Service (QoS)

2.4.1 Topologie und grundlegende Eigenschaften

Die physikalische Topologie für einen Verbund aus mehreren IEEE 1394-Geräten kann man graphentheoretisch als Baum bezeichnen. „Ein ungerichteter Graph, der keine Kreise positiver Länge enthält, heißt ein *Wald*. Ein zusammenhängender Wald heißt ein *Baum*." [23] Das bedeutet, ein serieller Bus auf Basis von IEEE 1394 muss zyklenfrei und jeder Knoten (Node) muss von jedem aus erreichbar sein, wie man in Abbildung 2 anhand eines Beispiels klar sieht. Ungerichtet heißt in diesem Fall, dass entlang dieser Verbindungen eine bidirektionale Kommunikation möglich ist. Die direkte Verbindung von zwei benachbarten Nodes mit Twisted-Pair-Kabeln sollte laut Empfehlung im Standard IEEE 1394a [3] höchstens 4,5 m lang sein und der längste Weg zwischen zwei Nodes sollte aufgrund von begrenzten Signallaufzeiten maximal 16 Kabelstücke umfassen, sodass sich eine maximale Entfernung von 72 m ergibt. Insgesamt existieren für den 1394a-Standard und diese Art der Verkabelung drei mögliche Datenübertragungsraten: der Modus S100 weist schon mit seinem Namen auf die Datenrate hin, denn er läuft mit 100 MBit/s, und so gibt es auch die Modi S200 und S400 mit doppelter und vierfacher Geschwindigkeit.

Im ursprünglichen Standard von 1995 wurde das Medium, das die Nodes miteinander verbindet, durch drei Drahtpaare definiert, wobei zwei davon für die Übertragung der Datensignale und das dritte für die Energielieferung verantwortlich ist. Die zugehörigen 6-poligen Stecker für die Anschlüsse an den Geräten sind in ihrer Form einzigartig und sehr robust konstruiert. In der Erweiterung IEEE 1394a kam dann schließlich die 4-polige Variante mit einem eigenen kleineren Stecker und passendem Port hinzu, die ohne der integrierten Stromversorgung arbeiten.

Im Zusatz IEEE 1394b werden andere Medien und andere Arten der Signalverarbeitung eingesetzt, um einerseits weit längere Entfernungen zu überbrücken und andererseits die Übertragungsraten zu steigern. So ist es möglich, mit einem Kabel aus Plastic Optical Fiber (POF) ohne aktiven Verstärker bis zu 50 m lange Verbindungen aufzubauen, jedoch nur im S200-Modus. Mit den bei gewöhnlichen Netzwerken verwendeten CAT-5-Kabeln können 100 m Links errichtet werden, wiederum ist die Geschwindigkeit aber sogar auf 100 MBit/s eingeschränkt. Nur das Medium Glass Optical Fiber (GOF) erreicht die 100 m sogar mit den neuen Modi S800 und S1600. Mit den gewöhnlichen wohl am weitesten verbreiteten Twisted-Pair-Kabeln mit einer maximalen Länge von 4,5 m lassen sich ebenfalls diese hohen Datenraten des 1394b-Standards erzielen, natürlich nur, wenn die beiden miteinander kommunizierenden Devices auch diesen neuen Zusatz unterstützen. In der Spezifikation wurden auch schon Vorbereitungen für einen S3200 Modus mit 3,2 GBit/s getroffen, jedoch haben die bekannten Hersteller ausschließlich Produkte in ihrem Sortiment, die im besten Fall eine Unterstützung für S800 mitbringen. Mit den höheren Datenraten wurde auch ein neuer Anschluss konzipiert, der sogenannte Beta-Konnektor, an dem ein Betrieb der älteren Geräte bis zum 1394a-Standard nicht möglich ist. Allerdings gibt es einen bilingualen Konnektor, der beide „Sprachen" spricht und deshalb auch mit den langsameren Produkten im S400 Modus kommunizieren kann.

Aufgrund der eingesetzten Baumstruktur besteht die Möglichkeit, dass ein Node die einzige Ver-

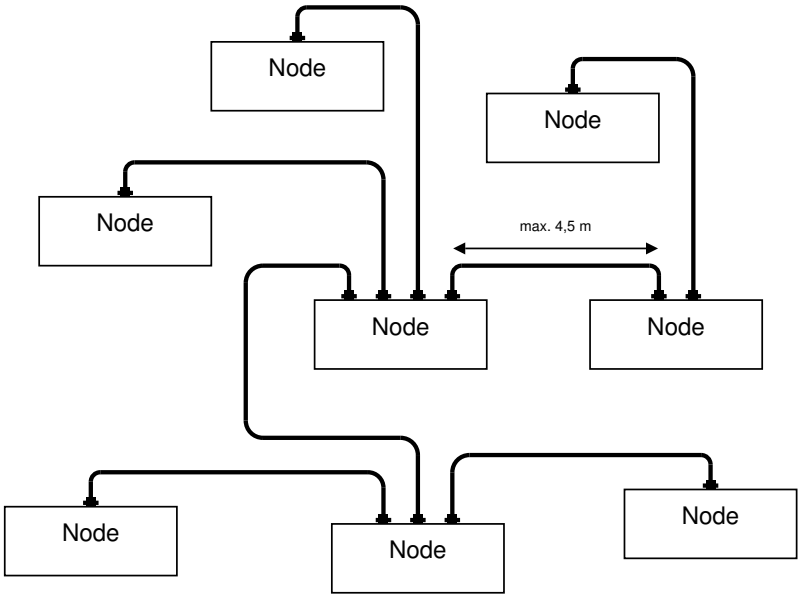

Abbildung 2: IEEE 1394 Topologie

bindung zwischen zwei anderen Nodes darstellt und deshalb müssen alle die Funktion eines Repeaters übernehmen, das heißt, sie senden alle eingehenden und selbst produzierten Signale an alle anderen direkt angeschlossenen Nodes weiter und verbreiten somit die Information. Wenn nun ein Node deaktiviert oder entfernt wird, kann sich in speziellen Topologieausprägungen der Bus in zwei getrennte Bäume aufteilen, die keine Verbindung mehr zueinander haben. Zumindest für den Fall, dass die lokale Stromversorgung eines Nodes ausgeschaltet wird, gibt es eine Abhilfe: das dritte Adernpaar im Medium liefert ausreichend Energie, um den Physical Layer eines nicht aktiven Nodes in Betrieb zu halten und damit die Weiterleitung der Signale zu gewährleisten. Der Hauptvorteil des integrierten stromführenden Paares stellt jedoch die vollständige Energieversorgung nicht zu stromhungriger Devices dar, die sich mit 8 - 40 Volt Gleichstrom und bis zu 1,5 Ampere begnügen. Wieviel Energie zum Beispiel ein IEEE 1394-Adapter in einem Computer tatsächlich an andere Nodes liefern kann, ist jedoch systemabhängig und abgesehen von den genannten Grenzen nicht im Standard spezifiziert.

Für die Adressierung eines Nodes ist ein Feld mit einer Länge von 6 Bit reserviert, wodurch sich für einen Bus maximal 63 Teilnehmer plus eine Broadcastadresse ergeben. Allerdings können durch spezielle Bus Bridges bis zu 1023 Busse miteinander gekoppelt werden, sodass sich in der größten Ausbauform 64449 Nodes in einem Netzwerk befinden.

2.4.2 Protokollarchitektur

Der Protokoll-Stack dieses seriellen Busses besteht aus drei Schichten, der unterste und der Hardware am nächsten ist der Physical Layer, darauf baut der Link Layer und schließlich der Tran-

saction Layer auf. Jeder Node besitzt Kontroll- und Statusregister (CSRs = Control and Status Registers), die seinen momentanen Zustand beschreiben und durch bestimmte Transaktionen modifiziert werden können.

1. **Physical Layer:**

   Der Physical Layer verbindet die Hardware mit der Software, das bedeutet, er setzt die elektrischen Signale, die über das Medium vom Bus kommen, in logische Bitfolgen um, die der Link Layer weiterverarbeitet, und umgekehrt. Er regelt mit Hilfe eines Entscheidungsverfahrens auch, dass nicht zwei oder mehr Nodes gleichzeitig Daten senden. Weiters findet auf dieser Schicht die Initialisierung des Busses statt, während dieses Prozesses unter anderem jeder vorhandene Node eine Adresse erhält und damit eindeutig identifizierbar wird und gerichtet angesprochen werden kann. Genaueres zur automatischen Bus-Konfiguration wird im Kapitel 2.4.4 diskutiert.

2. **Link Layer:**

   Im Link Layer wird die Adressierung und die Überprüfung der Daten durchgeführt. Hier werden auch die Datenpakete für das Senden zusammengestellt und die Daten aus den empfangenen Paketen extrahiert. Diese Schicht erzeugt auch das Cycle-Signal, das in gleichen Abständen wiederholt produziert wird und essentiell für die isochronen Transfers ist, da es für das Timing und die Synchronisation der Datenströme verwendet wird.

3. **Transaction Layer:**

   Der Transaction Layer implementiert ein Request-Response-Protokoll für die asynchronen Transfers, wobei nur drei unterschiedliche Arten von Transaktionen existieren: Read, Write und Lock. Alle isochronen Daten, abgesehen von den Paketen, die für das Management der isochronen Datenübertragung relevant sind, laufen an dieser Schicht unbeachtet vorbei.

2.4.3 Datenübertragungsmodi

Der Standard IEEE 1394 sieht zwei Arten der Datenübertragung vor: asynchrone und isochrone Transfers, die sich grundlegend voneinander entscheiden und aufgrund ihrer genauen Gegensätzlichkeit zusammen alle möglichen Anwendungsfälle abdecken. Die gesamte zur Verfügung stehende Bandbreite, die zum Beispiel im langsamsten S100 Modus 100 MBit/s beträgt, wird auf beide Übertragungsarten aufgeteilt, wobei die isochronen Transfers eine höhere Priorität genießen und damit auch - wenn nötig - mehr Bandbreite erhalten als die asynchronen Transfers.

Die Aufgabe des Cycle Masters, die von genau einem Node am Bus übernommen wird, besteht darin, die Zeitlinie in Teile zu je 125 μs aufzuteilen und am Beginn eines dieser sogenannten Cycles ein Paket auszusenden, um den Start zu signalisieren. Damit bestimmt die Uhr des Cycle Masters den Ablauf auf dem gesamten Bus. In der Abbildung 3 aus [24, S.40] ist einer dieser

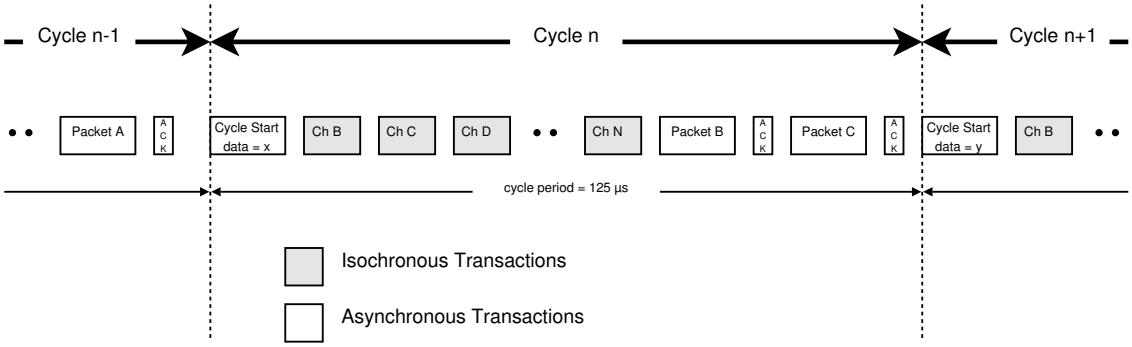

Abbildung 3: IEEE 1394 Cycles [24]

Cycles im Detail gezeigt, wobei man auch hier die Bevorzugung der isochronen Transfers sehen kann, denn bis zu achtzig Prozent der Zeitspanne eines Cycles kann für diese Daten reserviert werden. Die restlichen zwanzig Prozent, also 25 μs, werden fair unter allen Nodes, die asynchrone Daten senden wollen, aufgeteilt.

Jeder Node am Bus besitzt das sogenannte Cycle Time Register, das man mit einer Art internen Uhr gleichsetzen kann. Das 32 Bit breite Feld wird in drei Unterbereiche unterteilt: Die untersten 12 Bit werden durch einen 24,576 MHz Signalgenerator bis zu einem Wert von 3071 inkrementiert, bevor wieder bei 0 begonnen wird. Bei jedem Überlauf werden die nächsthöheren 13 Bit erhöht, was in Intervallen von 125 μs passiert und somit den Beginn eines neuen Cycles signalisiert. Nach 8000 Cycles, also nach genau einer Sekunde, werden die restlichen höchsten 7 Bit erhöht und der mittlere Bereich - der Cycle-Zähler - wird wieder zurückgesetzt. Zu Beginn jedes Cycles sendet der Cycle Master im Startpaket den aktuellen Inhalt seines Cycle Time Registers mit und synchronisiert damit alle anderen Nodes. Wie man später in dieser Arbeit noch sehen wird, ist dieser interner Zähler essentiell für die isochrone Übertragung und die rechtzeitige Präsentation von multimedialen Daten beim Empfänger.

2.4.3.1 Asynchrone Datenübertragung Wenn es darum geht, dass bestimmte Daten mit hundertprozentiger Sicherheit beim Empfänger ankommen und dabei auch noch ohne Fehler übertragen werden sollen, werden asynchrone Transfers eingesetzt, denn nur durch regelmäßige Bestätigungen des Erhalts und der Richtigkeit von Seiten des Kommunikationspartners kann sichergestellt werden, dass keine Störungen und Zwischenfälle die Konsistenz der Daten beeinträchtigt haben. Diese Prozedur ist unumgänglich bei Daten, die einen gewissen Langzeitwert besitzen und auf deren Informationen zukünftige kommunikative Abläufe aufbauen.

Pro Cycle stehen für asynchrone Pakete zum Beispiel im S100 Modus maximal 512 Byte zur Verfügung, bei den schnelleren Modi sind dementsprechend dann Vielfache dieser Datenmenge erlaubt. Die Informationen sind immer an einen bestimmten Node auf dem Bus gerichtet, der mit Hilfe einer 64-Bit-Adresse eindeutig identifiziert wird und nach Erhalt mit einer Antwort die kor-

rekte Übertragung bestätigt. Sie haben aufgrund ihrer Natur eine niedrigere Priorität als isochrone Pakete, da hier der zeitliche Faktor eine eher geringere Rolle spielt.

Oft wird diese Art der Datentransfers für die anfängliche Aushandlung der Kommunikationsparameter verwendet, um eine Garantie zu besitzen, dass alle beteiligten Nodes sich über die Zuteilung der Ressourcen im Klaren sind und ihr weiteres Verhalten auf diese Entscheidungen aufbauen können.

2.4.3.2 Isochrone Datenübertragung Im Gegensatz dazu steht die isochrone Übertragung, bei der in regelmäßigen vorgegebenen zeitlichen Abständen ohne Ausnahme ein Paket vom Sender Richtung Empfänger weggeschickt wird, ohne Kenntnis darüber, ob die Daten auch tatsächlich ankommen. Genau aufgrund der Tatsache, dass hier nur eine Einwegkommunikation stattfindet, müssen die Informationen einen speziellen Charakter haben, sie müssen sehr kurzlebig sein, das heißt, aktuelle Daten sollten so wenig wie möglich auf den Inhalt vergangener isochroner Pakete aufbauen. Dadurch entstehen auch keine ernsten Probleme, wenn ab und zu Informationen verloren gehen, da keine oder nur geringfügige Abhängigkeiten zwischen den Daten besteht. Es würde auch keinen Sinn machen, die nicht beim Empfänger angekommenen Daten in zukünftigen Paketen zu wiederholen, denn dann sind die Informationen bereits nutzlos, abgesehen davon, dass der Sender über den Verlust überhaupt nichts weiß. Man kann hier eine enge Verbundenheit zwischen der zeitlichen Komponente und den Informationen selbst erkennen, die in dieser Form bei asynchronen Transfers nicht besteht.

Der Standard IEEE 1394 spezifiziert insgesamt 64 isochrone Kanäle, identifizierbar durch eine 6 Bit lange Adresse, in denen in Intervallen von 125 μs je ein isochrones Paket abgeschickt wird. Diese Kanalnummern stellen auch die einzige Möglichkeit dar, eine Adressierung vorzunehmen, das heißt, es wird kein definitiver Node angesprochen, sondern ein gewisser Sendebereich für bestimmte Informationen reserviert. Jeder Node auf dem Bus, der Interesse an den Daten eines Kanals hat, kann seine Empfangskomponenten auf diesen Bereich ausrichten und schließlich den Datenstrom empfangen.

Natürlich muss der Sender zuvor eine Genehmigung beantragen, die ihm die nötige Bandbreite zusichert. Diese zentrale Bandbreitenverwaltung der isochronen Daten übernimmt der Isochronous Resource Manager (IRM). Er führt Buch über die noch verfügbaren Kanäle und die im aktuellen Modus noch frei Bandbreite und teilt den anfragenden Nodes einen Teil davon zu. Pro Cycle dürfen die maximal 64 isochronen Pakete eine Datenmenge von 1024 Bytes im S100 Modus nicht überschreiten, wobei das Limit in den anderen Modi proportional zur Geschwindigkeit steigt, sodass im schnellsten S3200 Modus die Grenze schon bei höchstens 32768 Bytes liegt.

In der Abbildung 4 wird der allgemeine Aufbau eines isochronen Datenpakets gemäß [2, S.157] dargestellt, dessen nicht selbsterklärende Felder hier noch kurz beschrieben werden [18, S.17]:

- **tag:** Der Wert 01b deutet darauf hin, dass sich im Datenblock noch ein zweiter Header, der

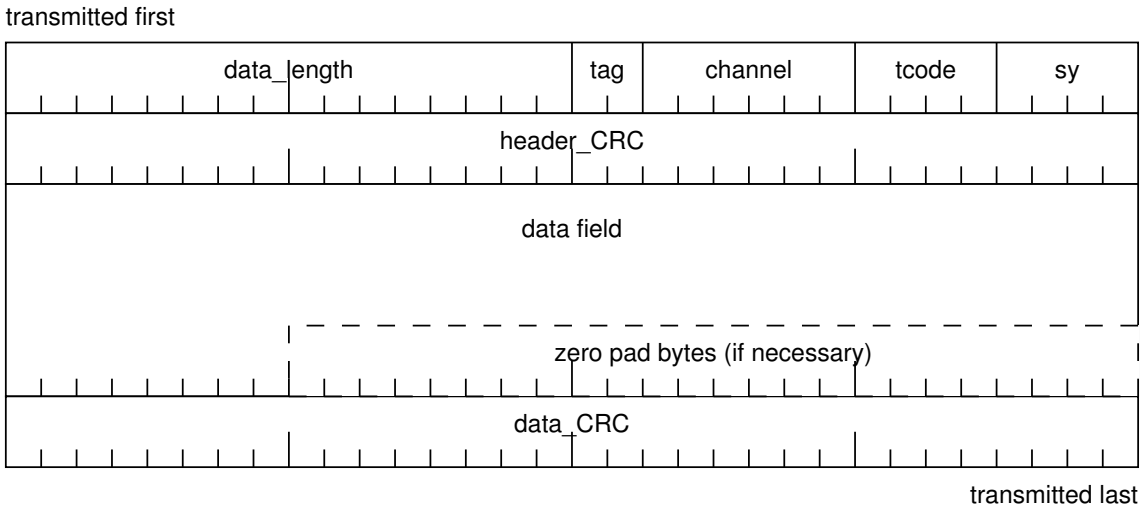

Abbildung 4: Isochrones Paketformat [2]

sogenannte CIP-Header (Common Isochronous Packet Header)[9], befindet.

- **channel:** Diese 6 Bit stehen für den isochronen Kanal, auf dem gesendet werden soll.
- **tcode:** Der Transaktionscode beschreibt den Typ des IEEE 1394-Pakets. Ein Wert von 1010b definiert ein isochrones Paket.
- **sy:** Dieses reservierte Feld steht im Großteil der Fälle auf 0000b, außer es wird von einer speziellen Anwendung benötigt und dann dementsprechend geändert.
- **header_CRC** und **data_CRC:** Diese Felder beinhalten Prüfsummen für den Header und den Datenteil zur Gewährleistung einer korrekten Datenübertragung ohne Störungen.
- **zero pad bytes:** Ist der Wert in data_length nicht durch vier teilbar, das heißt, dass die Datenmenge nicht in komplette Quadlets[10] aufgeteilt werden kann, dann wird das letzte begonnene Quadlet mit Nullen aufgefüllt.

2.4.4 Buskonfiguration und Aufgabenverteilung

Jedes Mal, wenn ein Node zum Netzwerk hinzugefügt oder vom Bus entfernt wird, das heißt, sobald sich an der Topologie Änderungen ergeben, wird ein Bus Reset ausgelöst. Dadurch wird das gesamte automatische Konfigurationsverfahren zur Bestimmung der Busstruktur und der IDs der Nodes erneut durchgeführt. Dieser Reset kann auch trotz unveränderter Topologie auf Wunsch von einem beliebigen Node absichtlich initiiert werden.

[9] siehe Paragraph 2.4.5.1 auf Seite 28
[10] siehe Kapitel 2.2 auf Seite 14: Quadlet

Insgesamt besteht diese automatische Buskonfiguration aus drei Phasen, der Bus Initialization, der Tree Identification und der Self Identification.

1. Während der Bus Initialization werden alle Informationen zur Busstruktur und die IDs in allen Nodes gelöscht. Dazu wird ein Reset-Signal von einem Node startend ausgesendet und an alle anderen durchgereicht, wonach anschließend alle laufenden Transaktionen gestoppt werden und jeder sich in den Grundzustand begibt. Am Ende dieser Phase ist jedem Node bewusst, ob er ein Endstück darstellt, das heißt, ob er nur mit einem Port mit dem Bus verbunden ist, oder ob er die Funktion eines Verbindungsglieds für mehrere Teilbereiche übernimmt. Dieses Wissen ist entscheidend für den Aufbau der restlichen Topologie-Informationen in den folgenden zwei Phasen.

2. In der Tree Identification Phase wird eine hierarchische Struktur aufgebaut, die an der Spitze den sogenannten Root-Node stehen hat. Wie man in der Abbildung 5 deutlich sehen kann, kann jeder Node über seine Ports mit Eltern und/oder Kinder verbunden sein, die in der Grafik dementsprechend mit „p" für „parent" und „c" für „child" gekennzeichnet sind. Aufgrund der internen Logik dieser zyklenfreien Struktur muss es aber zwangsweise einen Node geben, der keine Eltern hat, der dann der schon erwähnte Root-Node wird. Zusätzlich nummeriert jeder Node seine aktiven Anschlüsse durch, um sich für die dritte Phase der Self Identification vorzubereiten.

3. In dieser letzten Phase wird jedem Node eine eindeutige ID zugeordnet, die er dann bis zum nächsten Bus Reset behält. Der Vorgang der Identifikation beruht auf einem Depth-First-Search-Algorithmus, der bewirkt, dass die Nummerierung bei den Blättern am unteren Ende des verkehrt stehenden Baumes beginnt und der Root-Node am Schluss die höchste ID erhält. Zusätzlich sendet jeder Node Informationen über seinen aktuellen Status, die das Bus Management auswertet, um die gesamte Topographie endgültig zu fixieren.

Das Bus-Management spezifiziert drei Aufgabenbereiche, wobei jeder nur von maximal einem Node ausgeführt werden darf. Meistens übernimmt sogar ein Node alle drei Funktionen, was jedoch nicht zwingend vorgeschrieben ist. Die ersten beiden Bereiche wurden schon in vergangenen Kapiteln beschrieben.

1. Cycle Master

2. Isochronous Resource Manager (IRM)

3. Bus Manager: Er verwaltet detaillierte Informationen zur Topologie, wie die maximalen Geschwindigkeiten einzelner Segmente des Busses, und versucht auch, die Kommunikation zwischen den Nodes durch Optimierung der Wartezeiten zu verbessern. Weiters verwaltet er die Stromversorgungen der einzelnen Nodes und die Bandbreiten für asynchrone Übertragungen. Da diese komplexeren Funktionen hauptsächlich in Software ausgeführt werden,

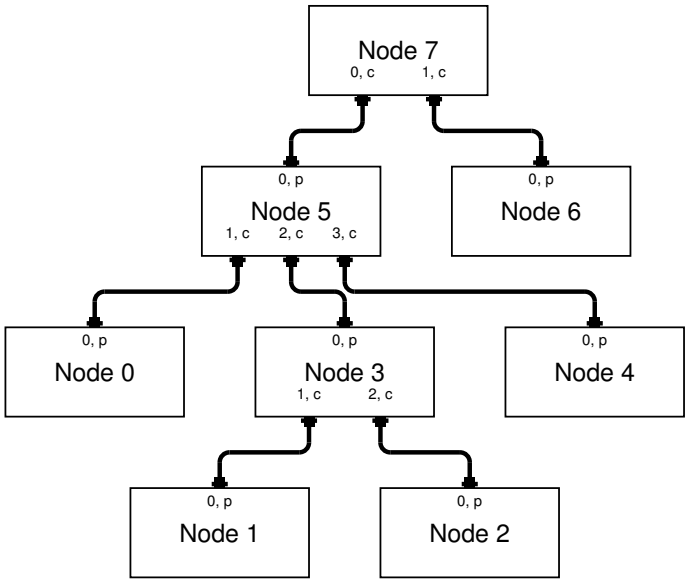

Abbildung 5: Topologie nach einem Bus Reset [24]

kann es vorkommen, dass kein Node auf dem Bus geeignete Fähigkeiten für deren korrekte Umsetzung besitzt, deshalb ist die Existenz des Bus Managers optional und entbehrlich, aber empfohlen. Im Notfall kann der IRM notwendige Teile des Power Managements übernehmen.

2.4.5 IEC 61883

Der Standard IEC 61883 beruht auf IEEE 1394 und versucht, ein allgemeines Interface und Richtlinien für die Übertragung von audiovisuellen Daten über den IEEE 1394-Bus zu definieren. Er richtet sich dabei vor allem an Hersteller von Consumer-Electronics-Hardware, die im Bereich Audio/Video-Equipment angesiedelt sind und eine IEEE 1394-Schnittstelle in ihre Geräte integrieren. Festgelegt wird einerseits das genaue Verfahren zum Aufbau von Verbindungen zwischen Sender und Empfänger und zum Aushandeln der zugehörigen Parameter und andererseits die Struktur der Datenpakete selbst.

Insgesamt gibt es sieben Bestandteile, aus denen dieser Standard aufgebaut ist und von denen der erste und vor allem der sechste essentiell für dieses Projekt sind und deswegen detaillierter behandelt werden:

1. General: Der Teil 1 des IEC 61883 spezifiziert das generelle Paketformat, das Datenfluss- und Verbindungsmanagement für Audio- und Videodaten, sowie die allgemeinen Übertragungsrichtlinien für Kontrollbefehle. [15]

2. SD-DVCR Data Transmission: Mit diesem Teil und den Teilen 3 und 5 wird festgelegt, wie Daten eines digitalen Videorekorders auf einem isochronen Kanal basierend auf IEEE 1394 übertragen werden.
3. HD-DVCR Data Transmission
4. MPEG2-TS Data Transmission: Hier wird die isochrone Übertragung eines MPEG2-Transportstroms definiert.
5. SDL-DVCR Data Transmission
6. Audio and Music Data Transmission Protocol: Die Details dieses Abschnitts werden nach der Aufzählung noch genauer behandelt.
7. Transmission of ITU-R BO.1294 System B: Dieser letzte Teil definiert die Übertragung von Transportströmen des Standards ITU-R BO.1294 System B (DirecTV System/DSS).

Der Standard IEC 61883-6, auch bekannt als „Audio and Music Data Transmission Protocol" (AMDTP), beschreibt ein Protokoll für die Übertragung von Audiodaten über IEEE 1394-1995 und die späteren Versionen. Dies beinhaltet den Transport des IEC 60958 Digitalformats, von rohen Audiosamples und MIDI-Daten. [18, S.8] In den folgenden Unterkapiteln werden einige wichtige Aspekte dieses Protokolls, wie der interne Aufbau und die Übertragungsarten, diskutiert.

2.4.5.1 CIP-Header In der Abbildung 6, die ihren Ursprung in [15, S.51] hat, aber für diese Arbeit noch etwas modifiziert wurde, zeigt einen CIP-Header, dessen Werte schon teilweise an einen Header für ein Paket im AMDTP-Format angepasst wurden. Die restlichen noch offenen Felder haben laut [18, 15] folgende Bedeutung:

- **SID (Source ID):** In dieses Feld wird die Node-ID des Senders eingetragen.
- **DBS (Data Block Size):** Diese Größe definiert, aus wievielen Quadlets ein Event oder Datenblock im Paket besteht. Im Fall von AMDTP kann man diesen Wert mit der Anzahl der Audiokanäle gleichsetzen, da ein Sample in einem Quadlet gespeichert wird, und ein Event aus der Summe aller Audiosamples zu einem Zeitpunkt zusammengesetzt wird.
- **DBC (Data Block Count):** Da in einem isochronen Paket mehrere Datenblöcke respektive Events gesendet werden können, wird im DBC-Feld von Beginn der Transmission an ein Zähler geführt, deren Wert immer auf den ersten Block nach dem CIP-Header ausgerichtet ist. Aufgrund des sehr kleinen maximalen Wertes in diesem 8 Bit breiten Feld läuft der Zähler schnell über, doch hauptsächlich dient der DBC zur Fehlererkennung: Falls ein Übertragungsfehler auftritt und dadurch nicht alle Datenblöcke beim Empfänger angekommen sind, kann dieser das feststellen und dementsprechend die fehlenden Daten in das Timing einfließen lassen. Ein weiterer wichtiger Anwendungszweck wird beim SYT-Feld beschrieben.

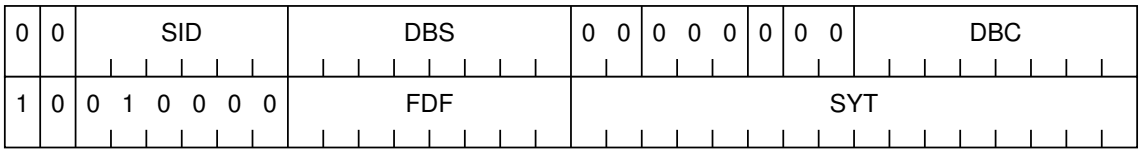

Abbildung 6: CIP Header mit SYT-Feld [15]

- **FDF (Format Dependent Field):** Dieses Feld beschreibt das verwendete Audioformat und kodiert auch zusätzlich die nominelle Samplingrate der nachfolgenden Audiodaten. Im AMDTP-Standard sind nur folgende Samplingfrequenzen zulässig: 32000 Hz, 44100 Hz, 48000 Hz, 88200 Hz, 96000 Hz, 176400 Hz und schließlich 192000 Hz.[11]

- **SYT (Synchronization Timestamp):** Mit diesem Feld kann man den Präsentationszeitpunkt der gesendeten Multimediadaten beim Empfänger regulieren beziehungsweise festlegen, natürlich nur, wenn dieser die Interpretation des hier gespeicherten Timestamps auch durchführt und sein weiteres Vorgehen danach richtet. Da in den meisten Fällen mehrere Events in einem isochronen Paket verschickt werden, muss festgelegt werden, für welches Event der Timestamp gilt. Dazu wurde das sogenannte SYT-Intervall eingeführt, das vor Beginn der Übertragung abhängig von der Samplingrate gewählt wird. Für eine Frequenz von 44,1 kHz wird standardmäßig ein SYT-Intervall von 8 definiert, wodurch auch gleichzeitig die maximale Anzahl an Events pro Paket fixiert wird. Der Zeitstempel im SYT-Feld bezieht sich immer auf das mitgelieferte Event, das genau auf die Grenze des SYT-Intervalls fällt und somit ein Vielfaches davon darstellt. Um diese Position berechnen zu können, muss ein Zähler von Beginn an geführt werden, wofür sich der DBC hervorragend eignet.

  Somit wird auch klar, wieso sich nicht mehr Events, als das SYT-Intervall umfasst, in einem Paket befinden dürfen, da sonst nicht eindeutig wäre, für welches der SYT-Wert gilt. Andererseits kann es aber sehr wohl vorkommen, dass ein Paket kein Event enthält, das ein Vielfaches des SYT-Intervalls darstellt. In solchen Fällen hat es auch keinen Sinn, einen Timestamp mitzuliefern, weshalb das Feld einfach auf FFFFh gesetzt wird.

  In den Synchronisationsalgorithmen des vorliegenden Projekts wird dieses Feld dazu verwendet, die Wiedergabegeschwindigkeit geringfügig anzupassen und damit unter anderem Pufferleerläufe zu vermeiden.

2.4.5.2 Übertragungsmodi Eine der wichtigsten Aufgaben des Senders der Audiodaten ist die Berechnung, wieviele Events nun tatsächlich im aktuellen isochronen Paket mitgeschickt werden sollen, um den Datenfluss zur Senke nicht ins Stocken geraten zu lassen und den Empfänger auch nicht mit zu vielen Informationen, die er zu dem Zeitpunkt noch nicht benötigt, zu überschütten.

[11] genauere Informationen zum FDF findet man im Kapitel 2.4.5.3 auf der nächsten Seite und in [18, S.40]

Wenn diese Aufgabe gewissenhaft erfüllt wird, ist beim Empfänger nur ein kleiner Zwischenpuffer notwendig, bevor die multimedialen Daten dann tatsächlich präsentiert und danach nutzlos werden. Die naheliegendste Lösung wäre eine Versendung der einzelnen Events genau in der Geschwindigkeit, die die nominelle Samplingrate vorgibt, doch wie schon erwähnt, plant der Cycle Master genau 8000 isochrone Pakete pro Sekunde, womit man an eine Rate von 8000 Hz gebunden wäre. Da die Audioquellen aber ihre Daten in verschiedenen und meist höheren Samplingfrequenzen anbieten und man als Gateway flexibel bleiben muss und diese Parameter nicht modifizieren sollte, muss notgedrungen die Menge an Events pro IEEE 1394-Paket variiert beziehungsweise erhöht werden. Dafür existieren zwei grundlegende Ansätze respektive Modi:

1. **Blocking Mode:** In diesem Modus wird zu Beginn der Übertragung ein bestimmter Wert, der abhängig von der Samplingrate ist, fixiert. Dieser Wert ist mit dem SYT-Intervall gleichzusetzen. In ein isochrones Paket wird nun genau diese festgelegte Anzahl an Events integriert. Der Wert wird dabei so gewählt, dass er, multipliziert mit der Anzahl der Pakete pro Sekunde - also 8000 -, größer ist als die nominelle Samplingrate, sodass es notwendig wird, zwischen den vollen auch ab und zu leere Pakete zu verschicken.

2. **Non-Blocking Mode:** Im Non-Blocking Mode wird die Gesamtanzahl an Events pro Sekunde so gut wie möglich auf die 8000 Pakete aufgeteilt, was zum Beispiel bei einer Samplingfrequenz von 44,1 kHz präzise gerechnet in $5\frac{41}{80}$ Events pro Paket resultieren würde. Da natürlich nur ganzzahlige Mengen von Samples verwendet werden können, werden nun alternierend fünf und sechs Events verschickt, sodass im Endeffekt die gewünschte Samplingrate eventuell mit einer geringen Abweichung erreicht wird. Später in dieser Arbeit wird von dieser Methodik Gebrauch gemacht, um die Samplingfrequenz für Synchronisierungszwecke zu modifizieren.

2.4.5.3 AMDTP Sampleformate Direkt anschließend an den CIP-Header werden die einzelnen Audiosamples aneinander gereiht, wobei jedes Sample genau ein Quadlet belegt. Da im AMDTP-Standard aber nur Samples mit einer Größe von 16 Bit, 20 Bit oder 24 Bit zulässig sind, bleibt noch Platz für einige Zusatzinformationen, die in den ersten 8 Bit des Quadlets kodiert werden. In den meisten Fällen wird das AM824-Datenformat verwendet, das die Aufteilung des Quadlets bereits im Namen impliziert, wie man auch in der Abbildung 7 sehen kann. Ist das Audiosample selbst kleiner als 24 Bit, werden noch Nullen angefügt, um das Quadlet zu vervollständigen.

Die Informationen im ersten Byte des Quadlets definieren den Typ des Sampleformats, wobei hier nur zwei der neun in [18] beschriebenen Formate basierend auf der Wichtigkeit für dieses Projekt näher diskutiert werden:

- Multi-bit Linear Audio mit RAW-Daten: Samples in diesem Format können ohne vorherige Bearbeitung direkt an einen Digital-Analog-Konverter weitergegeben werden, was die ein-

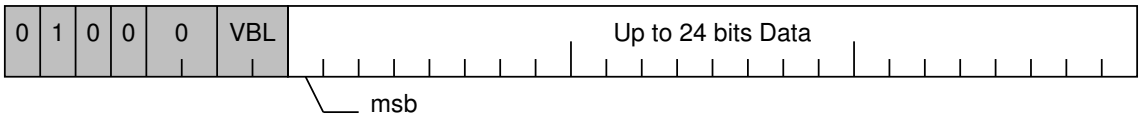

Abbildung 7: Quadlet mit Multi-bit Linear Audio Daten [18]

fache Handhabung dieser Daten erhöht. In der Grafik 7, entnommen aus [18, S.31], sieht man beispielhaft den Aufbau eines Quadlets, wobei des Feld VBL die Größe des Samples definiert.

- IEC 60958 Format: Dieses Format bietet erweiterte Möglichkeiten, Zusatzinformationen mitzuliefern, da es Strukturen definiert, die über ein Quadlet hinausgehen. So besteht ein sogenannter Block aus maximal 192 Frames, wobei ein Frame zwei Subframes enthalten kann, die mit einem Quadlet gleichzusetzen sind. Die ersten 4 Bit des Quadlets werden für die Kennzeichnung dieser Metastrukturen verwendet, in den restlichen 4 Bit werden komplexere Informationen kodiert, die über den gesamten Block reichen können.

2.4.6 Zusammenfassung

Dieses Unterkapitel soll noch einen kurzen zusammenfassenden Überblick über die Charakteristika und damit verbundenen Vor- und Nachteile des IEEE 1394-Standards geben.

- **Vorteile:**
  - Isochroner Transfermodus: Isochrone unidirektionale Übertragungskanäle eignen sich für echtzeitkritische Kommunikation zwischen verschiedenen Geräten.
  - Hohe Geschwindigkeiten: Aktuell sind Datenraten von maximal 800 MBit/s im IEEE 1394b-Standard, gebräuchlich und zum Beispiel in nahezu allen durchschnittlich ausgestatteten PC-Mainboards und in externen Festplatten integriert sind Adapter mit 400 MBit/s (IEEE 1394a-Standard).
  - Hohe Benutzerfreundlichkeit: Die Einfachheit der in alle IEEE 1394-Geräte integrierten und automatisch durchgeführten Bus-Konfiguration erleichtert die Handhabung.
  - Die grundlegenen IEEE 1394-Funktionen sind vom verwendeten Host-System unabhängig.
  - Weitere Nodes können durch Anstecken an einen beliebigen freien Port eines vorhandenen Nodes einfach hinzugefügt werden.
- **Nachteile:**

- Durch die begrenzte Medienlänge ist der IEEE 1394-Standard nahezu ausschließlich für die Multimediavernetzung innerhalb eines Raumes geeignet.
- Im Vergleich zu anderen Technologien, wie zum Beispiel dem Universal Serial Bus (USB), leidet der IEEE 1394-Standard unter einem geringeren Bekanntheitsgrad und deswegen unter einer weniger weiten Verbreitung. Aus diesem Grund investieren die Hersteller auch weniger in eine Unterstützung dieses Standards in ihren Produkten, und somit ergibt sich eine beschränkte Auswahl an IEEE 1394-Geräten.

2.4.7 Marktsituation und Ausblick

Bei Betrachtung des aktuellen Marktes und der Verbreitung von IEEE 1394-Produkten agiert der Standard eher im Hintergrund. Einige Hersteller, darunter Sony und Apple, die diese Technologie mitbegründet haben, setzen zwar noch auf die Integration des Standards in ihre Geräte, doch viele andere ignorieren ihn. Hauptsächlich kommt er noch in externen Festplatten und Videokameras zum Einsatz, wobei hier vor allem der FireWire800-Modus noch große Geschwindigkeitsvorteile gegenüber dem USB2.0-Standard mit seinen maximalen 480 MBit/s bietet.

Trotz dieser stagnierenden Entwicklung ist nach Meinung des Autors noch ein weiterer Schritt offen, den zum Beispiel der IP-Standard schon gegangen ist, und zwar der Übergang zum Datenübertragungsmedium Luft. Das Problem der hohen Störungsanfälligkeit und der daraus resultierenden geringen Datenraten wird zumindest teilweise durch eine neue Funktechnologie - die Ultrabreitbandtechnik - gelöst, womit eine drahtlose Verbindung von Geräten wesentlich attraktiver für Standards wie IEEE 1394 wird. Im Nahbereich von ungefähr zehn Metern ist mit theoretischen Geschwindigkeiten von 1 GBit/s zu rechnen. Falls man diese Technik als Basis für IEEE 1394 in Betracht ziehen sollte, wäre für drahtlosen Datenverkehr auch keine Konvertierung auf andere Protokolle und Standards, wie zum Beispiel IEEE 802.11, mehr notwendig, sodass die klaren Vorteile, wie die isochrone Datenübertragung, beibehalten werden können. [28]

2.5 RTP: Streaming auf IP-Basis

Das weltweit in der IT-Branche am intensivsten genutzte Kommunikationsprotokoll ist das Internet Protocol (IP) [5] und wird es mit sehr hoher Wahrscheinlichkeit auch bleiben. Es steht im Bereich der weltweiten Vernetzung von Rechnern an oberster Stelle und ist im Prinzip an keine physikalischen Grenzen gebunden. Die Abstände zwischen zwei Kommunikationspartnern können durch aktive Geräte wie Router oder Switches stets erweitert werden, wobei nur die Signallaufzeit unter den größeren Entfernungen leidet. Die einzige Einschränkung, die allmählich immer aktueller wird, ist der begrenzte Addressraum der öffentlich erreichbaren IP-Geräte, der in der momentan verwendeten Version IPv4 32 Bit umfasst und dessen noch freie Bereiche langsam zur Neige gehen. Doch durch den schon seit 1995 existierenden aber noch nicht umfassend in die

Praxis umgesetzten Standard IPv6 [8] wird auch dieses Problem aus der Welt geschafft sein, da hier der Adressraum auf 128 Bit erweitert wurde.

Gerade wegen der vielen Teilnehmer und der großen Entfernungen wurde der IP-Standard auf asynchrone Datenübertragungen optimiert, da hier weit wichtiger ist, dass jemand Informationen korrekt anstatt zu einem bestimmten Zeitpunkt erhält. Aus diesem Grund ist es schwierig, auf dieser Basis zuverlässige echtzeitkritische Verbindungen, wie zum Beispiel Audio- und/oder Videokonferenzen, aufzubauen. Trotzdem drängt sich der Wunsch auf, diese internationale Vernetzung auszunutzen und unter anderem auf Audio- und Videodatenbestände mit möglichst geringem Zeitverlust zuzugreifen.

2.5.1 RTP Grundlagen

1996 wurde das RTP (Realtime Transport Protocol) [9] von der IETF (Internet Engineering Task Force) in dem RFC (Request For Comments)-Dokument 1889 definiert und seitdem wird es als Protokoll zur Datenübertragung von zeitkritischen meist multimedialen Inhalten eingesetzt. Im Jahr 2003 wurde ein überarbeiteter RFC mit der Nummer 3550 publiziert [14], der das alte Dokument vollständig ersetzt. Er beinhaltet auch noch die Diskussion des Kontrollprotokolls RTCP (RTP Control Protocol), dessen Aufgabe das Monitoring der Datenströme darstellt. 1998 wurde ebenfalls von der IETF das Streamingprotokoll RTSP (Real-Time Streaming Protocol) in [11] festgelegt, das sich um die Aushandlung der Verbindungsparameter zu Beginn einer Übertragung kümmert.

2.5.1.1 Quality of Service Mit RTP besteht nun ein Ansatz, das Echtzeitproblem auf IP-Basis zumindest teilweise in den Griff zu bekommen. Trotzdem muss erwähnt werden, dass dieses Protokoll auf einer nicht echtzeitfähigen Infrastruktur aufbaut und nur einen Aufsatz auf den IP-Standard darstellt, der einige Charakteristika von Echtzeittransfers hinzufügt. RTP selbst implementiert keine Mechanismen zur Reservierung von Ressourcen für die Übertragung und stellt somit kein Quality of Service zur Verfügung, darum müssen sich die niedrigeren Layer des Netzwerk-Schichtenmodells kümmern. Darin liegt auch das größte Problem, und zwar, dass keine im IP-Standard verankerten priorisierten Kanäle für isochronen Datenverkehr existieren, so wie es zum Beispiel IEEE 1394 definiert. Deswegen ist es auch möglich, dass der gewöhnliche asynchrone Verkehr die Stabilität der Echtzeitverbindungen stören und somit die Anzahl der nicht rechtzeitig beim Empfänger ankommenden Datenpakete erhöhen können. Performance- und Bandbreitenengpässe in den vielen Routern und Switches auf dem Weg von der Quelle zur Senke können dafür die Ursache sein. Moderne Netzwerkkomponenten erlauben zwar die Priorisierung bestimmter Dienste, aber solange nicht in allen Zwischenstationen eines Datenstroms dieses Quality of Service verfügbar und genau für die benötigten Protokolle aktiviert und genügend Bandbreite reserviert ist, wird immer die Möglichkeit einer Störung bestehen, auch wenn

es nur eine einzige Komponente sein sollte, die die Voraussetzungen nicht erfüllt und somit den Single-Point-of-Failure darstellt.

Einige Internetzugangstechnologien, wie zum Beispiel MPLS (Multiprotocol Label Switching), bieten echtes QoS für bestimmte Dienste schon auf unterster Protokollschicht an, hauptsächlich für den Einsatz geschäftlich lebenswichtiger Dienste wie Voice over IP. Mit MPLS angebundene Hosts haben zwar auch Zugriff auf alle übrigen Rechner im Internet, aber natürlich bleiben die speziellen Fähigkeiten nur innerhalb des eigenen Netzes erhalten.

2.5.1.2 OSI Layer RTP und das unterstützende Protokoll RTCP positionieren sich im Transport Layer, wenn man sich auf das von der Standardisierungsorganisation ISO (International Organization for Standardization) definierte OSI (Open System Interconnection)-Schichtenmodell [22, S.59] bezieht. In der Abbildung 8 ist das für die Internetkommunikation vereinfachte OSI-Modell dargestellt, das die ursprünglichen sieben Layer zu vier Schichten zusammenfasst, wobei der Transport Layer aus Verständnisgründen zusätzlich unterteilt dargestellt wird. In diesem positionieren sich nämlich sowohl UDP als auch RTP, obwohl RTP auf UDP aufsetzt und somit logisch eine Schicht höher wäre.

Theoretisch kann RTP auf unterschiedlichsten darunter liegenden Protokollen aufbauen und agiert deshalb völlig unabhängig von ihnen. Meistens wird RTP jedoch in Zusammenhang mit IP und UDP verwendet, wodurch das Echtzeitprotokoll seine Stärken am besten ausspielen kann. Da es unter Umständen bei Verwendung von UDP zu Problemen in NAT[12]-Umgebungen kommen kann, kann auch das weniger geeignete Transportprotokoll TCP als Untersatz zum Einsatz kommen. Die gravierenden Unterschiede werden kurz diskutiert:

- **TCP (Transmission Control Protocol):** Dieses verbindungsorientierte Transportprotokoll läuft ähnlich ab wie beim asynchronen Transfer im IEEE 1394-Standard. Zu Beginn einer Datenübertragung wird mit einem Handshake-Verfahren sichergestellt, dass der Empfänger auch wirklich zuhört und empfangsbereit ist. Während des Sendens der Nutzdaten werden der Quelle ständig Pakete zurückgesendet, mit denen der korrekte Empfang der Informationen bestätigt wird. Hat der Sender nichts mehr zu sagen, wird die Verbindung wieder ordnungsgemäß abgebaut. Anders gesagt, obwohl nur einer der Partner dem anderen etwas mitteilen will, findet eine bidirektionale Kommunikation statt, damit eine erfolgreiche Übertragung garantiert werden kann. TCP wird in aktuellen Anwendungen im Internet und im lokalen Netzwerk sehr oft verwendet, da der Informationsgehalt der meisten Daten nicht an bestimmte Zeitpunkte bei der Übermittlung gebunden ist, sondern mehr die Aufrechterhaltung der Authentizität im Vordergrund steht und ein Verlust oder ungewollte Modifikationen nicht akzeptiert werden können.

[12] siehe Kapitel 2.2 auf Seite 14: NAT (Network Address Translation)

<table>
<tr><td>7. Application Layer</td><td rowspan="3">RTSP, HTTP, ...</td></tr>
<tr><td>6. Presentation Layer</td></tr>
<tr><td>5. Session Layer</td></tr>
<tr><td rowspan="2">4. Transport Layer</td><td>RTP, RTCP, Multicast Backbone</td></tr>
<tr><td>TCP, UDP</td></tr>
<tr><td>3. Network Layer</td><td>IP</td></tr>
<tr><td>2. Data Link Layer</td><td rowspan="2">Ethernet, TokenRing, MPLS, ...</td></tr>
<tr><td>1. Physical Layer</td></tr>
</table>

Abbildung 8: OSI Layer

- **UDP (User Datagram Protocol):** Im Gegensatz dazu ist UDP ein verbindungsloses Transportprotokoll, das man teilweise mit der isochronen Datenübertragung nach IEEE 1394 gleichsetzen kann. Bei diesem rein unidirektionalen Informationsaustausch werden die Nutzdaten ohne Vorwarnung und ohne Rückmeldung Richtung Empfänger geschickt. Dieser kann zwar mit Hilfe einer mitgelieferten Prüfsumme feststellen, ob die Daten, falls er überhaupt das Paket erhalten hat, durch irgendwelche Störungen geändert wurden, jedoch hat er keine Möglichkeit, die korrekten Informationen durch fehlerkorrigierende Mechanismen wiederherzustellen. Der große Vorteil von UDP gegenüber TCP liegt im viel geringeren Overhead dieses Protokolls, sowohl zeitlich gesehen als auch die Datenmenge betreffend. Da die komplette Übertragungszeit der Informationsmenge in einem Datenpaket nur durch das Zeitintervall vom Absenden dieses Pakets bis zum Empfang definiert ist, kann diese ungefähr abgeschätzt werden respektive wird sie sich beim nächsten Mal bei gleicher Datenmenge durchschnittlich betrachtet nur geringfügig ändern. Bei TCP-Verbindungen sind aufgrund der Mehrfachstrecken meist größere Abweichungen üblich, wodurch die Ankunftszeit mit geringerer Wahrscheinlichkeit vorhergesagt werden kann. Aus diesem Grund wird UDP hauptsächlich bei zeitkritischen Anwendungen eingesetzt, bei denen Informationen nur für eine gewisse Zeitspanne gültig sind und dann irrelevant werden.

Aufgrund dieser Fakten scheint es wenig vernünftig, TCP als Basis für RTP zu verwenden, da dann der Echtzeitgedanke teilweise ad absurdum geführt wird. Da manche Firewalls aufgrund der schwierigeren Zuordnung der verbindungslosen Pakete den UDP-Verkehr blockieren können, wird trotzdem auf TCP zurückgegriffen, doch dann könnte man genauso RTP zum Beispiel durch das verbindungsorientierte Anwendungprotokoll HTTP[13] ersetzen und man hätte einen ähnlich geringen Nutzen.

2.5.1.3 Sequence Numbering Ein weitere Herausforderung für RTP stellt die Tatsache dar, dass die Pakete, die vom Sender nacheinander abgeschickt werden, nicht in derselben Reihenfolge

[13] siehe Kapitel 2.5.2 auf Seite 38

beim Empfänger ankommen müssen. Der Grund dafür liegt in der Komplexität und Größe des die Welt umspannenden Internets, denn um die hohe Anzahl der sich chaotisch verhaltenden Kommunikationsflüsse zeitlich zu optimieren, existieren immer mehrere Wege zum Ziel. Wenn also einer der auf dem Pfad liegenden Router entscheidet, dass die Hauptroute für ein bestimmtes Paket zum Beispiel aus Gründen der Auslastung nicht verwendet werden kann, wird eine alternative Route gewählt. Genauer gesagt entscheiden die einzelnen Router nicht über den gesamten Weg eines Pakets, sondern nur aufgrund der bekannten Zieladresse über die nächste Zwischenstation. Dadurch können sich Strecken mit großen Längenunterschieden und somit auch stark differierende Ankunftszeiten für verschiedene Pakete des selben Datenstroms ergeben. Der Empfänger muss deshalb damit rechnen, dass einige Daten entweder zu spät ankommen und somit verworfen werden oder er die Reihenfolge wieder rekonstruieren muss, was mit Hilfe von Sequence Numbering möglich wird. Dabei werden die Pakete einer Session vom Sender durchnummeriert und somit wird eine eindeutig zu reproduzierende Abfolge erzeugt.

2.5.1.4 Multicasting Multicasting [6] ist aufgrund der Optimierung für Punkt-zu-Punkt-Verbindungen nicht auf so grundlegende Art und Weise in IP-Netzwerken implementiert wie etwa im IEEE 1394-Standard. Es existieren zwar IP-Broadcasts, doch diese wirken nur im selben Subnetz, in dem sich der Sender befindet, und können die Grenzen, die durch Router repräsentiert werden, nicht überschreiten. Als Ausweg für diese Misere wurden Multicast-Groups eingeführt, wobei jede dieser Gruppen durch eine IP-Adresse im Bereich von 224.0.0.0 bis 239.255.255.255 definiert ist. Um nun als IP-Host die Datenpakete einer Multicast-Group zu empfangen, ist eine Registrierung bei dieser notwendig. Die Aufgabe eines Multicast-Routers ist es nun, die Pakete, die an eine bestimmte Gruppe gerichtet sind, zu replizieren und an die registrierten Teilnehmer weiterzuleiten, wobei an dieser Stelle anzumerken ist, dass nicht jeder Router im Internet multicastfähig ist, sogar nur ein geringer Prozentsatz ist dazu in der Lage. Deswegen bilden sich einzelne Multicasting-Inseln im Internet, die intern diese Technik erfolgreich verwenden aber untereinander keine Kommunikation aufbauen können. Als Lösung für dieses Problem wurde folgender Ansatz entwickelt: Die Multicastpakete werden von geeigneten Routern in gewöhnliche Unicastpakete verpackt und in dieser Form durch das Internet zu anderen Empfängern gesendet, wo sich natürlich auch ein Router befinden muss, der dieses Feature beherrscht, das ursprüngliche Paket wiederherstellen und ins eigene multicastfähige Netzwerk weiterleiten kann.

2.5.1.5 Timestamping Auch das Timestamping, also das Mitliefern von Zeitangaben im Datenpaket, gehört zu den unverzichtbaren Charakteristika einer Echtzeitkommunikation, denn damit wird der Zusammenhang zwischen Zeit und Information hergestellt, ohne den es keine Echtzeitdaten wären. Dieser Zeitstempel im Header des Pakets gibt den Zeitraum an, in dem die im Datenblock gesendeten Informationen gültig sind. Befinden sich mehrere Datenblöcke im Paket, so muss vorher, das heißt in den Spezifikationen des Datenformats, geregelt werden, für welchen Teil der Zeitstempel gilt. Erhält der Empfänger das Datenpaket erst nach diesem Zeitpunkt, werden die

Daten überflüssig und können verworfen werden, weshalb es es von höchster Wichtigkeit ist, dass die Informationen so schnell wie möglich ihr Ziel erreichen. Dadurch können die Daten eventuell noch geeignet verarbeitet werden, bevor der angegebene Zeitpunkt eintritt und sie präsentiert werden müssen.

Meistens kommt es bei Echtzeitanwendungen weniger darauf an, dass ein einziges Datenpaket den Empfänger in einer möglichst kurzen Zeitspanne erreicht, sondern auf die Aufrechterhaltung eines konstanten Datenstroms, bei dem die Zeitdifferenzen zwischen den Paketen möglichst gleich beziehungsweise innerhalb gewisser Toleranzgrenzen bleiben sollten. In diesen Fällen ist die Erhaltung der Konstanz dieser Zeitintervalle von höherer Priorität als die Minimierung der Transferzeit.

Bei RTP werden die Timestamps meistens im NTP (Network Time Protocol)-Format dargestellt, woraus ein absoluter realer Zeitpunkt berechnet werden kann, da der angegebene Wert gleichzusetzen ist mit der Anzahl der Sekunden seit dem 1. Januar 1900 nach der UTC (Coordinated Universal Time). Laut [7] verwendet NTP ein Datenfeld mit 64 Bit Breite für die Zeitangabe, wobei genau nach den ersten 32 Bit der ganzteilige Anteil der Sekunden endet und der gebrochene Teil beginnt. In vielen Bereichen von RTP werden insgesamt nur 32 Bit für den Zeitstempel reserviert, sodass nicht der komplette NTP-Wert, sondern nur die mittleren 32 Bit verwendet werden.

Es besteht aber kein Zwang, die Zeitstempel in diesem absoluten Zeitformat darzustellen. Genausogut kann der Wert zu Beginn einer Übertragung auf Null gesetzt und dann fortlaufend basierend auf einer beliebigen sinnvoll gewählten kleinsten Einheit linear inkrementiert werden, denn wie schon zuvor erwähnt, geht es bei RTP primär um die Differenzen zwischen den einzelnen Timestamps.

2.5.1.6 Medien Die Medien, auf denen RTP übertragen wird, sind natürlich dieselben, auf denen auch IP übertragen werden kann, da das erstere ja auf dem letzteren und somit auch auf dessen Infrastruktur aufbaut. So gibt es im Nahbereich das am weitesten verbreitete CAT5-Kabel auf Kupferbasis und die drahtlose Übertragung in Form des IEEE 802.11-Standards. In Backbone-Anwendungsfeldern werden aufgrund ihrer erhöhten Reichweite Lichtwellenleiter und speziell für lange Richtfunkstrecken entwickelte Funktechnologien eingesetzt.

2.5.1.7 Anwendungen Wie schon beim Timestamping erwähnt, liegen die hauptsächlichen Anwendungsgebiete von RTP nicht im Senden von kurzen zeitlich unregelmäßigen Datenpaketen, sondern in der Verteilung von kontinuierlichen Datenströmen, in denen die Informationen meist sehr redundant ausgelegt sind, sodass Verluste oder Modifikationen geringer Datenanteile durch Störungen während der Übertragung vernachlässigbar sind und keine wiederholte Sendung dieser Daten erforderlich machen. Es werden nun einige der Anwendungssituationen exemplarisch genannt:

- **Konferenzen mit Audio- und/oder Videoübertragung:** Dieses klassische Beispiel zeigt einige der zuvor genannten Charakteristika sehr deutlich. So ist es nicht von Bedeutung,

wenn einige Sprach- oder Bildfragmente während dem Transfer verloren gehen, da einerseits die menschliche Sprache eine hohe Redundanz aufweist und andererseits bei einer Bildübertragung mit meistens gleichbleibenden und mehr oder weniger unbeweglichen Motiven nahezu alle visuellen Informationen mehrmals wiederholt werden.

- **Unidirektionales Streaming von Audio- oder Videoquellen:** Im Gegensatz zu den Konferenzen wird hier nur eine einseitige Form der Kommunikation gewählt. Diese oft live übertragenen Multimedia-Streams werden zum Beispiel von Radio- oder Fernsehsendern teilweise kostenlos angeboten, wobei die Verbindungen entweder über Multicast-Gruppen oder direkt zu den Empfängern aufgebaut werden. Im letzteren Fall ist natürlich weit mehr Bandbreite unmittelbar beim Streaming-Server notwendig.

- **IP-Telefonie (Voice over IP):** Diese immer interessanter werdende Alternative zu den herkömmlichen Telefonnetzen ist eigentlich ein Unterpunkt der Konferenzsituation. Die Übertragung der Sprachdaten basiert unter anderem auf der ursprünglich aus der Telekommunikationsbranche stammenden Empfehlung ITU (International Telecommunication Union)-H.323 [16], die auf einer Adaption von RTP aufbaut. Auch das in diesem Umfeld berühmte Protokoll SIP (Session Initiation Protocol) [13] greift bei der tatsächlichen Datenübertragung auf RTP zurück, da es selbst nur zum grundsätzlichen Aufbau der Kommunikation verwendet wird und danach die Kontrolle an SDP (Session Description Protocol)[14] und RTP übergibt.

- **Übertragung von Messwerten:** Die Messwerte von weit entfernt sitzenden ins Netzwerk eingebundene Sensoren können via RTP zu einer zentralen Datensammelstelle übertragen werden, die die Daten anschließend aufbereitet. Auch hier wirkt sich die Schlichtheit des Echtzeitprotokolls vorteilhaft aus, denn dadurch verringert sich die Komplexität in den netzwerkfähigen Sensoren. Die nicht vorhandene Garantie für eine erfolgreiche Übertragung stellt in den meisten Anwendungsfeldern dieser Art kein großes Problem dar, da die Messwerte in kurzen regelmäßigen Abständen versendet werden und die absolute Differenz zwischen zwei Werten meistens eher gering ausfällt.

2.5.2 Übersicht über Streamingprotokolle

Grundsätzlich kann zwischen drei Typen von Protokollen unterschieden werden, die beim Streaming eine Rolle spielen und an verschiedenen Punkten im gesamten Ablauf der Übertragung ansetzen:

1. Zu Beginn einer Datenübertragung müssen standardisierte Abläufe dafür sorgen, dass die Verbindungsparameter, wie zum Beispiel die verwendeten Codecs und Kommunikationsports, korrekt ausgehandelt werden, sodass die beide Partner den nachfolgenden Daten-

[14] siehe nächstes Kapitel

transfer verstehen und nicht aneinander vorbeireden. Dafür existiert eine eigene Protokollgattung, die über die Fähigkeiten des jeweiligen Clients oder Servers Bescheid weiß und diese dementsprechend dem Kommunikationspartner mitteilt. Nach diesem Aushandlungsgespräch stehen die Verbindungsparameter fest, unter der Annahme, dass die Durchschnittsmenge der Fähigkeiten beider Beteiligter nicht leer ist. Beispiele für die Implementierung dieser Funktionen sind die folgenden beiden Protokolle:

- **RTSP (Real-Time Streaming Protocol)**[15] **[11]**
- **SDP (Session Description Protocol) [12]**

2. Die zweite Protokollart regelt die anschließende Datenübertragung selbst, wobei hier neben RTP noch weitere teilweise weniger sinnvolle Varianten existieren:

   - **RDT (Real Data Transport):** Dieses proprietäre Echtzeitprotokoll, das vom Softwarehersteller RealNetworks entwickelt wurde, wird nur in Streaming-Servern dieses Herstellers angewandt, trotzdem findet es aufgrund der weiten Verbreitung hier eine Erwähnung.
   - **HTTP (HyperText Transfer Protocol):** Dieses auf TCP aufbauende Protokoll wird hauptsächlich zur Übertragung von Dokumenten rund um HTML verwendet, was wohl auf nahezu jedem PC unter allen Betriebssystemen möglich ist. Genau deswegen wurde es auch für Streamingzwecke herangezogen, obwohl es wegen seiner Natur und seinem protokolltechnischen Untersatz aus schon genannten Gründen überhaupt nicht dafür geeignet ist.

3. Mit Hilfe des dritten Typus, der Kontrollprotokolle, wird eine gewisse Überwachung des Datenstroms gewährleistet. In Zusammenarbeit mit RTP versucht das Protokoll RTCP (Real-Time Control Protocol) in regelmäßigen Intervallen während des Datentransfers Informationen zu sammeln. Sowohl die Quelle als auch der Empfänger versenden Zusammenfassungen, in denen festgehalten wird, wieviele Pakete in dem vergangenen Teil verloren gegangen beziehungsweise insgesamt verschickt oder angekommen sind.

   Eine weitere wichtige Aufgabe das Timestamping betreffend übernimmt RTCP in folgenden Situationen: Für jeden RTP-Datenstrom wird ein zufälliger Anfangswert für die interne Zeitmessung gewählt und die Inkrementierung und Auflösung der Timestamps dem Datenformat entsprechend angepasst. Müssen nun mehrere Streams wieder miteinander synchronisiert werden, wie es bei getrennten Audio- und Videoströmen der Fall ist, können die Timestamps der Streams aufgrund dieser Eigenheiten nicht direkt miteinander verglichen werden. Um dieses Problem zu lösen, wird nun in den regelmäßigen Sender Reports des RTCP mit Hilfe eines Timestamp-Paares eine Verbindung zwischen der internen Zeitmessung und einer offiziellen Zeit, zu der alle beteiligten Datenströme gleichermaßen Zugang haben,

[15] siehe Kapitel **2.5.4 auf Seite 41**

hergestellt. Mit diesem Anhaltspunkt ist nun eine einwandfreie Synchronisation möglich, ohne in jedem einzelnen RTP-Paket einen zusätzlichen Zeitstempel mitschicken zu müssen.

2.5.3 RTP-Header

In der Abbildung 9, entnommen aus [14, S.13], ist der allgemeine Aufbau des Headers eines RTP-Pakets mit seinen Elementen zu sehen, die nun näher erläutert werden:

- **V (Version):** Seit 1996 existiert die RTP Version 2 und deshalb ist dies der Standardwert für dieses Feld.
- **P (Padding):** Dieses Bit zeigt an, ob im Datenfeld abschließend an die Nutzdaten noch Nullbytes zur Vervollständigung angefangener Blöcke hinzugefügt werden.
- **X (Extension):** Ist dieses Bit gesetzt, werden zusätzliche Headerinformationen anschließend an den Standardheader erwartet.
- **CC (CSRC Count):** Dieses Feld beinhaltet die Anzahl an Contributing Source Identifiers, die dem fixen Header angehängt werden.
- **M (Marker):** Der anwendungsspezifische Marker wird für verschiedenste Zwecke eingesetzt oder auch manchmal mit dem Payload Type zusammengefasst.
- **PT (Payload Type):** Hier wird der Typ der mitgelieferten Daten definiert, so steht zum Beispiel der dezimale Wert 14 für das Format MPEG-I/II Audio.
- **sequence number:** Wie schon bei den grundlegenden Charakteristika erwähnt, dient diese fortlaufende Nummerierung der besseren Sortierung und Fehlererkennung beim Empfänger. Der Anfangswert sollte zufällig gewählt werden, um bei verschlüsselten RTP-Streams gegen spezielle Attacken von Dritten besser geschützt zu sein.
- **timestamp:** Dieses essentielle Feld beinhaltet den schon beschriebenen Zeitstempel, den der Empfänger zur zeitgerechten Präsentation benötigt.
- **synchronization source (SSRC) identifier:** Jeder Datenstrom bekommt einen eigenen eindeutigen 32 Bit langen Wert zugewiesen, mit deren Hilfe der Empfänger die Zugehörigkeit der eintreffenden Pakete feststellen kann und in der Folge keine Gefahr besteht, mehrere Streams zu vermischen. Bietet ein Sender Ströme von mehreren Audio- oder Videoquellen an, müssen sich diese in der SSRC unterscheiden.
- **contributing source (CSRC) identifiers:** In bestimmten Anwendungsgebieten, wie zum Beispiel bei Konferenzen, werden spezielle Softwarekomponenten, die sogenannten Mixer, verwendet. Diese nehmen mehrere RTP-Quellen auf und generieren daraus einen einzigen

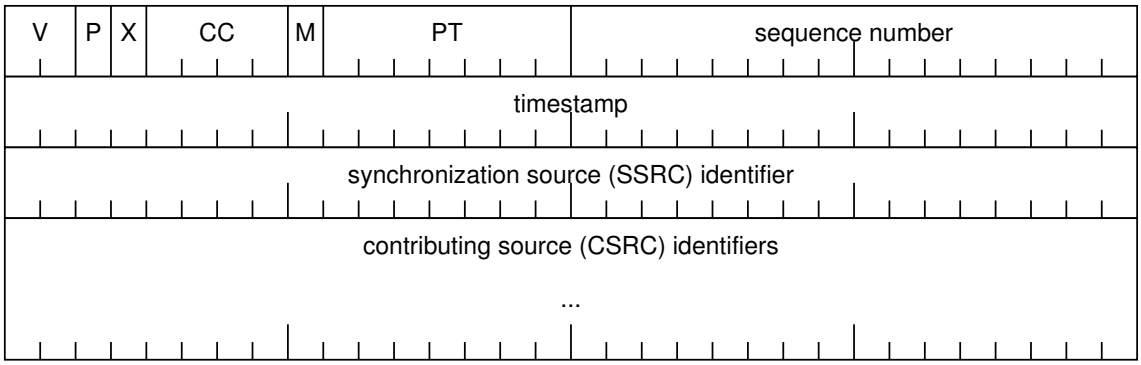

Abbildung 9: RTP Header [14]

Datenstrom, der mit einer neuen SSRC betitelt wird. Die ursprünglichen SSRCs der einzelnen Quellen werden nun zu CSRCs und in eine Liste im RTP-Header eingetragen. So kann ein Empfänger feststellen, welche Teilnehmer in dem vereinigten RTP-Stream mitgewirkt haben, was speziell bei Audiokonferenzen wichtig sein kann.

2.5.4 Streamingprotokoll RTSP

„... RTSP acts as a 'network remote control' for multimedia servers." [11, S.5]

RTSP steht für „Real-Time Streaming Protocol" und bezeichnet eine spezielle Handshake-Phase zu Beginn einer Kommunikation zwischen einem Server, der bestimmte Daten anbietet, und dem Client, der diese Daten abrufen will. Nachdem die anfängliche Aushandlungsphase abgeschlossen ist und sich die Kommunikationspartner auf beiden Seiten über die Verbindungsparameter erfolgreich geeinigt haben, wird der Datenverkehr wenn möglich auf Basis eines Echtzeitprotokolls, wie zum Beispiel RTP, begonnen. RTSP überträgt somit nur Metainformationen über die folgenden Datenstrukturen und -protokolle, und keine Nutzdaten selbst. Eine Ausnahme zu dieser Regel existiert jedoch: In einigen Fällen kann aufgrund schon erwähnter Firewall-Probleme mit UDP-Verkehr der Datenstrom in RTSP-Pakete eingekapselt werden. Natürlich muss dann der RTSP-Verkehr auf TCP-Basis laufen, um nicht wieder von bestimmten Firewalls blockiert zu werden. Mit dieser Technik können auch komplette RTP-Pakete auf UDP-Basis über RTSP getunnelt werden, wobei jedoch aufgrund der Verwendung von TCP als Transportprotokoll wichtige Echtzeiteigenschaften verloren gehen.

Der Aufbau und Ablauf von RTSP ähnelt dem von HTTP [10] mit einigen Unterschieden. So sind bei RTSP Anfragen von beiden Seiten möglich, das heißt sowohl vom Server als auch vom Client aus. HTTP muss auf keine Informationen von vergangenen Verbindungen aufbauen, RTSP-Server hingegen müssen sich in den meisten Fällen den Status der Verbindung merken. Bis auf die schon genannte Ausnahme, wird die Übertragung der Nutzdaten bei RTSP-Verbindungen von

anderen Protokollen geregelt, bei HTTP läuft die gesamte Kommunikation über dieses rein auf TCP aufbauende Protokoll ab. Gemeinsam ist ihnen aber das Request-Response-Modell, das RTSP mit einigen neuen Methoden erweitert, von denen hier die wichtigsten kurz beschrieben werden [11, S.30]:

- **OPTIONS:** Der Client kann jederzeit, auch während der laufenden Datenübertragung, einen OPTIONS-Request senden, wenn keine der Standardmethoden für seine Anfrage passend erscheint. Der Serverstatus wird dadurch nicht beeinflusst.
- **DESCRIBE:** Diese vom Client initiierte Methode fordert vom Server eine genaue Beschreibung der angebotenen Multimediadaten, und zwar in den vom Client unterstützten Formaten. Die Antwort vom Server könnte zum Beispiel in Form des schon erwähnten Protokolls SDP zurückkommen.
- **SETUP:** Hier wird geregelt, welche Verbindungsparameter für die Datenübertragung verwendet werden. Dazu sendet der Client zuerst eine Liste seiner Möglichkeiten, von denen der Server eine auswählt, die auch seinen Fähigkeiten entspricht, und teilt dem Client diese Entscheidung mit. Falls der Server keine der angebotenen Übertragungsarten akzeptiert, kommt keine Verbindung zu Stande. Festgelegt werden unter anderem das Echtzeitprotokoll (RTP, RDT, ...), das Transportprotokoll (TCP oder UDP), die Ports auf Transportebene für Client und Server und die Art der Verteilung - unicast oder multicast. Ein Beispiel für die Verwendung der SETUP-Methode wird im Algorithmus 1, entnommen aus [11, S.33], demonstriert. Von den beiden in einem Intervall angegebenen Ports wird der erste zur Übertragung des Daten-Streams und der zweite für den Kontroll-Stream, der vom RTCP genutzt wird, benötigt.
- **PLAY:** Diese Methode gibt dem Server die Anweisung, die Datenübertragung zu starten, wobei auch die Möglichkeit für den Client besteht, anzugeben, welchen zeitlichen Ausschnitt des Multimedia-Streams er empfangen möchte. Auch der absolute Zeitpunkt, wann die Übertragung begonnen werden soll, kann mit dem UTC-Zeitformat festgelegt werden.
- **PAUSE:** Wie der Name schon andeutet, wird damit der Stream solange angehalten, bis entweder ein erneutes PLAY gesendet und er damit fortgesetzt wird oder ein Timeout beim Server ein TEARDOWN verursacht.
- **TEARDOWN:** Damit wird die Übertragung beendet und die verwendeten Ressourcen am Server wieder freigegeben. Um den Stream erneut zu starten, muss wieder ein SETUP gefolgt von einem PLAY gesendet werden.

Algorithm 1 RTSP-Methode SETUP [11]

Client -> Server (Request):
SETUP rtsp://example.com/foo/bar/baz.rm RTSP/1.0
CSeq: 302
Transport: RTP/AVP;unicast;client_port=4588-4589

Server -> Client (Response):
RTSP/1.0 200 OK
CSeq: 302
Date: 23 Jan 1997 15:35:06 GMT
Session: 47112344
Transport: RTP/AVP;unicast;client_port=4588-4589;server_port=6256-6257

2.5.5 Zusammenfassung

Dieses Unterkapitel soll noch einen kurzen zusammenfassenden Überblick über die Charakteristika und damit verbundenen Vor- und Nachteile des RTP-Protolls geben.

- **Vorteile:**
  - hohe Flexibilität aufgrund der Unabhängigkeit von den unter RTP liegenden Protokollschichten
  - große Auswahl an möglichen Medien
  - nahezu unbegrenzte Reichweiten
  - hohe Geschwindigkeiten bei gut ausgebauter Infrastruktur
  - nahezu unbegrenzte Anzahl von Kommunikationspartnern (nur durch den möglichen Adressraum von IPv4 und IPv6 begrenzt)
  - billige Hardware
- **Nachteile:**
  - keine garantierte Übertragung bei Verwendung von UDP
  - kein echtes Quality of Service (QoS)
  - mögliche Router- und Firewallinkompatibilitäten

2.6 Gegenüberstellung von AMDTP und RTP

Die folgende Gegenüberstellung soll nicht den falschen Eindruck erwecken, dass die beiden Standards in direktem Konkurrenzkampf zueinander stehen und dass man sich für einen der beiden entscheiden müsse. Vielmehr soll diese Tabelle die teilweise sehr drastischen Unterschiede aufzeigen, die sehr differierende Einsatzgebiete ermöglichen, sodass sich die Entscheidungsfrage gar

nicht stellt. Eher sollte versucht werden, jede der beiden angepassten Standards in seiner Umgebung zu belassen und bei Bedarf eine Brücke dazwischen zu errichten, so wie es das Ziel dieses Projekts darstellt.

Die in der Tabelle 2.6 vorkommenden den IEEE 1394-Standard betreffenden Zahlen und Fakten wurden den Quellen [2, 3, 4, 24] entnommen.

Eigenschaften	AMDTP (basierend auf IEEE 1394)	RTP (basierend auf UDP/IP)
Echtzeitverhalten	isochrone Datenübertragung perfekt für Echtzeitanwendungen	kein „echtes" QoS, Ansätze dafür bei RTP auf UDP
Datendurchsatz, Bandbreite	bis zu 3,2 GBit/s (IEEE 1394b)	bis zu 10 GBit/s (nur bei Verwendung von Lichtwellenleitern)
Reichweite (ohne aktive Verstärker)	IEEE 1394a: max. 4,5 m bei Twisted-Pair-Verkabelung IEEE 1394b: max. 100 m mit speziellen Medien (optische Glasfaser oder CAT-5)	max. 100 m bei Twisted-Pair-Verkabelung bis mehrere hundert Kilometer bei Verwendung von Lichtwellenleitern
Übertragungsmedien	Twisted-Pair, optische Glasfaser, optische Plastikfaser, CAT-5	Twisted-Pair, Lichtwellenleiter, Funk
Netzwerkgröße (max. Anzahl von Knoten innerhalb eines zusammenhängenden Netzes)	1023 Subnetze zu je 63 Nodes => ca. 2^{16} Nodes	IPv4: max. 2^{32} Hosts IPv6: max. 2^{128} Hosts
Netzwerkinfrastruktur	baumförmiger Bus	mehrere Strukturen mit unterschiedlichen Techniken möglich: Bus, sternförmig (Ethernet), ringförmig (TokenRing)
Verbreitung	Heimnetzwerk, eher gering wegen Hauptkonkurrent USB 2.0	sehr hoch (Heimnetzwerke, Firmennetze, Internet)
Preis der zugehörigen Hardware	mittel bis hoch	gering
Sicherheit	standardmäßig nicht abhörsicher, aber Schutz durch DTCP (Digital Transmission Content Protection) [17] möglich	hohe Verschlüsselungen durch Zusatztechnologien wie IPSec möglich
Konfigurationsaufwand	keiner aufgrund der Selbstorganisation	manuell konfigurierbare IP-Adressen oder automatische Zuweisung durch DHCP (trotzdem ist ein DHCP-Server im Subnetz notwendig)

Tabelle 1: Vergleich von AMDTP und RTP

3 Projektbeschreibung

Dieses Kapitel soll das in der Einleitung bereits kurz charakterisierte Projekt näher erläutern und die wichtigen Punkte dabei herauskehren. Es werden die verwendeten Konzepte und Komponenten beschrieben, wobei jedoch nicht zu sehr auf technische Details eingegangen wird, da dafür das Kapitel 4 verantwortlich ist. Auch wird versucht, zu rechtfertigen, wieso bestimmte Lösungsvarianten verwendet wurden und andere nicht.

3.1 Problemdefinition

Die Motivationsgrundlage des dieser Arbeit zugrunde liegenden Problems stellt das Paper [26] dar, in dem auch eine theoretisch gehaltene Lösung präsentiert wird. Es geht dabei um die Vereinigung des Standards IEEE 1394[16] mit dem Datenübertragungsprotokoll IP[17], sodass es auch den Multimediageräten, die nur in der Welt des FireWire beheimatet sind, möglich wird, auf die Unmengen an Datenbeständen im Internet zuzugreifen. Aufgrund der großen Unterschiede zwischen den beiden Welten, die im Kapitel 2 ausreichend diskutiert wurden, muss einem Aspekt bei der Datenkonvertierung besondere Aufmerksamkeit geschenkt werden. Die zeitliche Synchronisierung des beim Gateway eingehenden von externen Quellen stammenden Datenstroms mit den Multimediadaten, die auf einem isochronen Kanal auf den IEEE 1394-Bus weitergeleitet werden, stellt aufgrund der unterschiedlichen Echtzeiteigenschaften der beiden Standards ein großes Problem dar. Die fehlende Übertragungsgarantie von Seiten der unter Umständen weit entfernten Multimedia-Server muss auf effiziente Art und Weise kompensiert werden, sodass das IEEE 1394-Endgerät möglichst wenig respektive gar nichts von den Problemen außerhalb seiner Reichweite und Fähigkeiten mitbekommt. Dazu muss das Gateway eine geeignete Strategie entwickeln, um mit Hilfe eines Zwischenspeichers diese Unregelmäßigkeiten, die einerseits zu Datenmangel und andererseits zu Datenüberfluss führen können, auszugleichen.

Der Begriff Multimediadaten umfasst sowohl Audio- als auch Videodaten, trotzdem liegt der Schwerpunkt der vorliegenden Implementierung auf reinen Audio-Streams. Beispiele dafür sind live übertragene Internetradiosender oder große Musikarchive, die zentral auf einem Server verwaltet und an alle Anfragenden entweder unentgeltlich oder gegen Gebühr verteilt werden. Die in dieser Arbeit demonstrierten Konzepte können aber genauso auf Videodaten angewandt werden, nur muss zusätzlicher Aufwand bei der synchronen Zusammenführung von Audio- und Videostrom betrieben werden.

3.2 Lösungsansätze

Dieses Projekt zeigt eine mögliche Implementierung, wobei jedoch die bidirektionale Kommunikation im IEEE 1394-Netzwerk außer Acht gelassen wurde. Das heißt, das Endgerät hat in dieser

[16] siehe Kapitel 2.4 auf Seite 19
[17] siehe Kapitel 2.5 auf Seite 32

softwaretechnischen Lösung noch keine Möglichkeit, auf das Gateway Einfluss auszuüben und eventuelle Rückmeldungen über empfangene Transfers an den Sender zu liefern. Im Kapitel 4.6.2 wird dieses Problem und eine mögliche Lösung dazu geschildert.

3.2.1 IP over IEEE 1394

Ein besonders interessanter Aspekt von IEEE 1394-Implementierungen ist die Verwendung von IEEE 1394-Hostadaptern als gewöhnliche Netzwerkkarten für IP-Datenverkehr. Für alle gängigen Betriebssysteme existiert bereits eine Implementierung, die den Protokollstack für IP-Daten direkt auf die asynchrone Datenübertragung auf IEEE 1394-Hardware legt. Diese Art der Netzwerkverbindung ist sogar weit schneller als der momentan am weitesten verbreitete 100-MBit/s-Ethernet-Standard, und zwar 400 MBit/s beim 1394a-Standard und 800 MBit/s beim 1394b-Standard, womit diese Vernetzungstechnik bereits mit Gigabit-Ethernet konkurrieren kann.

Natürlich wäre mit diesem IP-Aufsatz auf die IEEE 1394-Verbindung das Problem der Brücke zwischen den beiden Standards schon teilweise auf einfache Art und Weise gelöst. Die Gateway-Software leitet die eingehenden IP-Pakete entweder unverändert oder nach Dekodierung in ein einfacheres Audioformat sofort an die IP-Schnittstelle der IEEE 1394-Hardware weiter. Keine Konvertierung in das AMDTP-Protokoll wäre notwendig und um das Auspacken der IP-Pakete sowie um die zeitgerechte Präsentation der Audiodaten müsste sich das Endgerät kümmern. Im Prinzip werden durch diese schnelle Lösung die beiden Standards physikalisch miteinander verknüpft, was einer der Punkte der Problemstellung darstellt, und die Rechenleistung des Gateway-Rechners könnte minimiert werden. Gründe, wieso auf dieser Basis das Projekt seinem Ziel inklusive einer geeigneten Synchronisation nicht näherrückt, werden nun in den Nachteilen dieser Lösung aufgezeigt:

1. Es werden damit viele Vorteile des IEEE 1394-Standards außer Acht gelassen, da der IP-Aufsatz nicht auf der isochronen, sondern der asynchronen Datenübertragung aufbaut. Das wäre auch nicht möglich, wenn man die Eigenschaften und die Verfahren des Internet Protokolls bedenkt.

2. Es existieren abgesehen von vollständigen Computersystemen mit gängigen Betriebssystemen wenige bis gar keine IEEE 1394-Hardwareprodukte, die mit dem IP-Aufsatz und den damit verbundenen Komplikationen wie der Synchronisation der Audiodaten umgehen können.

3. Der Konfigurationsaufwand für die IEEE 1394-Geräte würde steigen, da sie zuvor in das IP-Netzwerk integriert werden müssen.

4. Eine Gateway-Software sollte den Endgeräten soviel Arbeit wie möglich abnehmen, damit deren Rechenleistung minimiert werden kann und nicht die des Gateway-Rechners. Es

ist bei gleichbleibendem Resultat weit effizienter, die Performance eines Rechners zu steigern als die aller restlichen Teilnehmer des Netzwerks. In einigen Situationen, in denen die Lastverteilung und die Entlastung einzelner Komponenten hohe Priorität hat, möge diese Aussage nicht zutreffen, aber in vorliegendem Szenario würde jedes der Endgeräte genau die gleiche Arbeit durchführen müssen wie alle anderen. Hier wäre es weit effizienter, wenn diese Arbeit vor dem Broadcasten von einem einzigen Gerät abgeschlossen werden könnte.

3.2.2 Konvertierung zu AMDTP

Die einzige Möglichkeit, die Vorteile der isochronen Übertragung von Audiodaten vollständig ausnutzen zu können und gleichzeitig die Daten so einfach wie möglich für die Endgeräte aufzubereiten, besteht in der Verwendung eines genau für diese Zwecke definierten Standards. AMDTP erfüllt diese Spezifikationen und wird deshalb in der endgültigen Lösungsvariante als Protokoll für den IEEE 1394-Bus verwendet. Um die Daten in dieses vordefinierte Format zu bringen, muss das Gateway aber weit mehr Leistung aufbringen als beim Weiterleiten der IP-Pakete im ersten Lösungsansatz. Da die Audiodaten im AMDTP-Format sehr einfach gestrickt und weder komprimiert noch speziell strukturiert sind, müssen zuerst die von extern eintreffenden Daten dekodiert werden, sodass nur mehr die rohen Audiosamples zur Verfügung stehen. Diese werden in einem Zwischenspeicher gelagert, bis der vorherberechnete Zeitpunkt für die Integration in ein isochrones Paket erreicht ist. Die dabei angewandten Synchronisierungsstrategien werden noch detaillierter in 3.7 dargestellt.

3.3 Projektziele

Das Ziel dieses Projekts ist es, einerseits eine allgemeine Lösung für die Verbindung der beiden Standards bieten zu können, die sich leicht adaptiert auf jegliche Art von Echtzeitdatenströmen anwenden lässt, und andererseits auch eine praktische Demonstration zu liefern, die die Theorie auch mit handfesten Resultaten unterstützt. Diese erste Softwareversion stellt eine funktionierende Variante der Übertragung von Audiodaten aus externen auf IP basierenden Quellen zu AMDTP unterstützenden IEEE 1394-Geräten dar. Die Daten können dabei in einer Vielzahl von Formaten[18] eintreffen, wobei die Komponente, die für das Dekodieren verantwortlich ist, auch unabhängig vom Fortschritt dieses speziellen Projekts erweitert werden kann, was im nächsten Kapitel noch genauer beschrieben wird. Das für die Übertragung von der Quelle zum Gateway verantwortliche Protokoll muss nicht unbedingt RTP sein, es werden auch einige Alternativen dazu unterstützt, die ähnlich den Audioformaten in regelmäßigen Abständen hinsichtlich der Stabilität und Kompatibilität aktualisiert werden.

Diese erste Implementierung legt mehr Wert auf die Funktion als auf die Optik der Benutzeroberfläche, vor allem weil es sich um eine Server-Applikation handelt und deswegen hauptsächlich

[18] siehe Kapitel 3.5.1 auf Seite 50

im Hintergrund arbeitet. Trotzdem gäbe es am Benutzerkomfort noch einiges zu verbessern. Auch existieren nach dem Start der Software noch wenige Interaktionsmöglichkeiten, die das Verhalten der Gateway-Software während der Laufzeit beeinflussen könnten. In der aktuellen Version müssen alle notwendigen Parameter noch vor dem Start in Konfigurationsdateien und Startskripten festgelegt werden, sodass die Ausführung im Prinzip unbeaufsichtigt ablaufen kann. Trotzdem wird dem Anwender mit dem Log-Viewer[19] eine Möglichkeit zur passiven Überwachung zur Verfügung gestellt, wo aufgetretene Fehler und wichtige Meldungen angezeigt werden.

Der vorliegende Lösungsansatz zu dem in [26] beschriebenen Problem und die daraus resultierende praktische Implementierung gehen nur einen der möglichen Wege, und auch dieser ließe sich noch erweitern, da einige Features nicht integriert und nicht jeder Anwendungsfall mit dem vorliegenden Gateway funktioniert[20], aber die wichtigen Punkte und vor allem die Synchronisation wurden korrekt berücksichtigt.

3.4 Grundlegende Entscheidungen

Als Programmiersprache, in der die Implementierung durchgeführt wurde, wurde die höhere Sprache „C" verwendet, da sie aufgrund ihrer Einfachheit und Effizienz eine weite Verbreitung gefunden hat. Die meisten systemnahen Komponenten von Betriebssystemen werden in „C" geschrieben, da sie ursprünglich auch für den direkten Kontakt mit der Hardware entwickelt wurde. Der hohe Bekanntheitsgrad förderte auch die Entwicklung vieler Programmbibliotheken, die jedem Programmierer wichtige Codesequenzen zur Verfügung stellen, so wie es auch in diesem Projekt beim Zugriff auf die IEEE 1394-Hardware der Fall war.

Ein weiterer und der wohl wichtigste Grund für die Verwendung dieser Sprache ist die Tatsache, dass die externe Softwarekomponente namens MPlayer, die die Datenübertragung im IP-Standard und die Dekodierung der Audiodaten übernimmt, in „C" entwickelt wurde und mit dem Teil, der den IEEE 1394-Standard behandelt, sehr eng gekoppelt ist. Durch die Verwendung derselben Programmiersprache fällt natürlich die Verbindung der beiden Systeme wesentlich leichter. Trotzdem wurde versucht, das Band dazwischen so lose wie möglich zu halten, um beide Komponenten getrennt voneinander weiterentwickeln zu können und nur darauf achten zu müssen, die gemeinsame Schnittstellenspezifikation aufrecht zu erhalten.

Die Wahl der Entwicklungsumgebung fiel ohne Zögern auf das freie Betriebssystem Linux, da aufgrund der Offenlegung des gesamten Systemquellcodes ausreichend Dokumentation und Unterstützung von Gleichgesinnten bereitsteht, um das nötige Verständnis zur Audio- und IEEE 1394-Programmierung zu erlangen. In kommerziellen Systemen kann die fehlende Einsicht in grundlegende Vorgänge bei der Kommunikation mit der Hardware die Durchführung eines Projekts der vorliegenden Art erschweren.

[19] siehe Kapitel 4.6.3 auf Seite 90
[20] siehe Kapitel 5.1 auf Seite 92

3.5 MPlayer

Der gemäß der GNU General Public License [34] frei verfügbare Mediaplayer MPlayer [35] wurde von unabhängigen freiwilligen Programmierern entwickelt und hat sich aufgrund seiner breiten Formatunterstützung inzwischen zu einem der beliebtesten Softwarepakete für diesen Zweck hochgearbeitet. Seine Königsdisziplin ist die Wiedergabe von Videos in vielen aktuellen Formaten, doch auch die Audiounterstützung wirkt sehr ausgereift. Durch die große Entwicklergemeinde wurden Versionen für alternative Betriebssysteme adaptiert, neben Microsoft Windows und Linux zum Beispiel auch für MacOS X. Die Verfügbarkeit des Quellcodes der gesamten Software ermöglicht natürlich auch die Verwendung auf gänzlich anderen Systemen, das Vorhandensein der notwendigen Werkzeuge zum Kompilieren und eventuell das Durchführen einiger nötiger Modifikationen vorausgesetzt. Auch wurden zur Laufzeitoptimierung und Erhöhung des Datendurchsatzes bei der Wiedergabe einige rechenintensive Routinen speziell an die vorhandene Hardware angepasst.

Für die Ausgabe der Video- beziehungsweise Audiodaten existieren mehrere systemabhängige Möglichkeiten, die als eine Art Modul an den Hauptteil des MPlayers gebunden sind. Gäbe es nun auch eine Ausgabemodul für die isochronen Kanäle des IEEE 1394-Standards, dann wäre dieses Projekt nicht notwendig gewesen, abgesehen von einer eventuellen Optimierung der Synchronisation. Doch genau auf diese Lücke und auf den modularen Aufbau setzt die hier beschriebene Lösungsvariante auf, was noch tiefergehender in 4.2 geschildert wird.

3.5.1 Formatunterstützung

Viele der aktuell in meist herstellerspezifischen Mediaplayern verwendeten Codecs sind nur in binärer Form erhältlich, das heißt es existiert keine Möglichkeit, abgesehen vom in manchen Fällen angebotenen kostenintensiven Erwerb, in die Interna des jeweiligen Formats Einblick zu erhalten. Zusätzlich dazu sind die binären Versionen oft nur für bestimmte kommerzielle Betriebssysteme kompiliert, sodass die Nutzung im freien Linux seinen treuen Anhängern dadurch verwehrt bleibt. Nun nutzt der MPlayer einen speziellen Trick, um dieses Problem zu umgehen: Er simuliert die Zugriffe auf die DLLs[21] genau auf die Art, wie sie unter Microsoft Windows geschehen, sodass es für den Codec selbst aussieht, als ob er sich in seinem für ihn vorgesehenen vertrauten Betriebssystemumfeld befindet. Damit können nun einfach die nicht offen gelegten Formatbibliotheken in andere Plattformen integriert werden. Folgend werden noch die unterstützten Codecs aufgezählt, wobei nur die Übergruppen der einzelnen Formate Erwähnung finden.

- **Video-Codecs:**
  - MPEG-1 (VCD)

[21] siehe Kapitel 2.2 auf Seite 14: DLL

  - MPEG-2 (SVCD, DVB), auch verschlüsselte Video-DVDs werden unterstützt
  - MPEG-4 in allen Varianten inklusive DivX ;-), OpenDivX (DivX4), DivX 5 (Pro), XviD
  - Windows Media Video (mit Hilfe der Windows-DLLs)
  - RealVideo
  - QuickTime
  - Sonstige eher seltenere Codecs: Sorenson, Cinepak, 3ivx, Intel Indeo, VIVO, H.263, MJPEG, AVID, VCR2, ASV2, FLI, FLC, HuffYUV, NSV (Nullsoft Streaming Video), NuppelVideo, Matroska

- **Audio-Codecs:**

  - MPEG Layer 1, 2 und 3 (MP3)
  - AC3, A52 (Dolby Digital)
  - AAC (MPEG-4 Audio)
  - WMA (DivX Audio), WMA 9
  - RealAudio: COOK, SIPRO, ATRAC3
  - QuickTime: Qclp, Q-Design QDMC/QDM2, MACE 3/6, ALAC
  - Ogg Vorbis
  - Sonstige Codecs: Voxware, VIVO (g723, Vivo Siren), alaw/ulaw, (ms)gsm, PCM

3.5.2 Hardware- und Protokollunterstützung

Die multimedialen Daten in den gerade genannten Formaten können über verschiedenste Wege zum Dekoder des MPlayers gelangen, entweder über Speichermedien, die mit lokalen Schnittstellen verbunden werden, oder über externe Kommunikationskanäle, wie zum Beispiel das Netzwerk oder unidirektionale Fernsehempfangsanlagen:

- Jegliche Zugriffe auf Daten, die über Funktionen des Betriebssystems laufen (lokale Festplatten, Freigaben im Netzwerk, CD- und DVD-Medien, sonstige mobile Speicher)
- IP-basierte Zugriffe auf Streaming-Server via HTTP, FTP, RTP/RTSP, MMS/MMST, MPST, SDP
- Unterstützung von lokalen DVB-Karten oder analoger TV-Hardware

Speziell im Bereich des IP-Streaming taucht ein störendes Kompatibilitätsproblem auf: Aufgrund des proprietären Protokolls RDT, das auf den Streaming-Servern von RealNetworks die Rolle des

RTP übernimmt, muss das Streaming von RealAudio-Dateien von RealNetworks-Servern gesondert behandelt werden. In diesem Fall greift der MPlayer auf eine spezielle Implementierung zurück, die ursprünglich vom Mediaplayer Xine[22] portiert wurde, und verwendet nicht die LIVE555-Bibliotheken[23] wie bei allen anderen RTP-Verbindungen. Zusätzlich zu diesem Umstand ist es auch nicht möglich, diese speziellen Streamingsitzungen auf UDP zu transportieren, sondern nur über TCP, womit wiederum eine signifikante Eigenschaft der Echtzeitdatenübertragung verloren geht.[24]

3.6 Beschreibung der Abläufe

Im dynamischen Sequenzdiagramm in der Abbildung 10, das dem UML-Standard [27] entspricht, sieht man einen Ausschnitt der programmtechnischen Vorgänge während der Laufzeit der Gateway-Software von der Initialisierung beginnend. Die auftretenden Namen und Begriffe entsprechen nicht zwingend den zugehörigen Funktionsnamen im Quellcode, auch sind bei weitem nicht alle in der Software vorkommenden Funktionen im Diagramm dargestellt, da es dann visuell zu überladen wirken würde. Mit dieser Grafik soll dem Leser nur die grobe Arbeitsweise der Gateway-Software und vor allem die Zusammenarbeit zwischen MPlayer und den selbst erstellten Komponenten nähergebracht werden. Genau zwischen den Objekten „MPlayer - AudioOut" und „IEEE 1394 Interface" findet der Übergang zwischen dem MPlayer und den hinzugefügten IEEE 1394-Teilen statt. Der einzige gravierende Eingriff in den Quellcode des MPlayers war das Einfügen der IEEE 1394-Funktionsaufrufe im Modul „MPlayer - AudioOut", weshalb das MPlayer-Hauptprogramm nichts von der Weitergabe der Audiodaten an die IEEE 1394-Schnittstelle weiß.

Mit Hilfe der vertikalen Zeitachse, deren Richtung von oben nach unten verläuft, ist das zeitliche Zusammenspiel der Komponenten etwas besser ersichtlich, obwohl durch das verwendete Multithreading eigentlich keine präzise Ablaufreihenfolge zwischen den Threads existiert. Die Aktivität eines Moduls wird durch ein senkrechtes Rechteck dargestellt, wodurch man auch sehen kann, dass abgesehen von einem kurzen Zeitintervall für die Initialisierung nach dem Start der Software die gesamte restliche Laufzeit drei parallel arbeitende Komponenten aktiv sind: das Hauptprogramm des MPlayers inklusive der Weiterleitung der Samples an den zentralen Audiopuffer, der Thread, der für die Übertragung der Audiodaten an die IEEE 1394-Schnittstelle verantwortlich ist, und schließlich der Thread für die Ausgabe des aktuellen Status und der Fehlermeldungen[25].

Ein Stern vor einer Nachricht bezeichnet ein sich wiederholendes Ereignis, das heißt der Prozess des Holens neuer Daten von der Audioquelle und des Weiterleitens dieser Daten an den AudioOut-Teil läuft in einer Schleife im Hauptteil des MPlayers. Die eckigen Klammern beim Anfordern neuer Daten beschreiben eine Bedingung, die erfüllt sein muss, damit die Ausführung gestartet

[22] xine - A Free Video Player, siehe http://www.xinehq.de
[23] siehe Algorithmus 2 auf Seite 64
[24] siehe Kapitel 5.1.2 auf Seite 92
[25] siehe Kapitel 4.6.3 auf Seite 90

wird. In dem vorliegenden Fall muss der Cache des MPlayers bis zu einer bestimmten Grenze beziehungsweise komplett geleert sein, damit dieser wieder befüllt wird.

Das Prinzip der wiederholten Ausführung findet man auch beim „IEEE 1394 Transmit Thread", der ständig Daten vom Ringpuffer einliest und sie dem IEEE 1394-Adapter übergibt. Die zeitliche Abfolge dieser Ereignisse muss somit nicht in der dargestellten Reihenfolge passieren, das Diagramm zeigt nur einen der möglichen Abläufe. Damit es zu keinen unvorhersehbaren Resultaten kommt, erfolgt jeder Zugriff auf den Ringpuffer atomar, das heißt, bevor ein Lese- oder Schreibvorgang nicht abgeschlossen ist, kann kein weiterer beginnen.[26] Nach jeder Modifikation des Puffers wird eine Nachricht an den „Status Output Thread" geschickt, um die im Log-Viewer dargestellten Werte auf dem neuesten Stand zu halten. Zusätzlich wird nach dem Hinzufügen neuer Daten zum Ringpuffer dem „IEEE 1394 Transmit Thread" dies signalisiert, damit dieser, falls er sich aufgrund mangelnder Audiodaten in einer Art Schlafzustand befindet, die Arbeit wieder aufnimmt.

3.7 Synchronisation der Audiodaten

Falls die Situation eintritt, dass der Audio-Puffer Tendenzen zum Über- oder Unterlauf zeigt, muss die Wiedergabegeschwindigkeit der Samples, also im Prinzip die Samplingrate, geringfügig variiert werden, um die Situation wieder zu normalisieren. Die Ursachen für Über- oder Unterläufe des Puffers können sehr vielfältig sein:

- Die internen in Hardware ausgeführten Uhren der verschiedenen bei der Kommunikation teilnehmenden Geräte können unterschiedlich schnell laufen, weshalb sich im Laufe der Zeit aufsummierte minimale Differenzen zu großen Unterschieden entwickeln können. Dadurch könnte sich die Situation ergeben, dass der Sender die Pakete zu langsam wegschickt und der Empfänger die Daten zu schnell wiedergibt.

- Falls die Audioquelle sich im Internet befindet und die gesendeten Samples bis zum Ziel sehr viele Router und Switches passieren müssen, können diese durch Überlastungen oder kompletten Ausfall der Komponenten entlang des Weges entweder unterschiedlich stark verzögert werden oder im schlimmsten Fall sogar verloren gehen.

- Konkurrierende und im Hintergrund laufende Prozesse auf dem Gateway-Rechner können zu Überlastungen führen, die die Verarbeitungsgeschwindigkeit der Audiosamples vermindern können. Diese Problemsituation kann aber von der Software selbst nicht gelöst werden, da sie auch nur einen der möglicherweise blockierten Prozesse darstellt und keinen Überblick über das gesamte System besitzt.

[26] siehe Kapitel 4.4.4 auf Seite 76

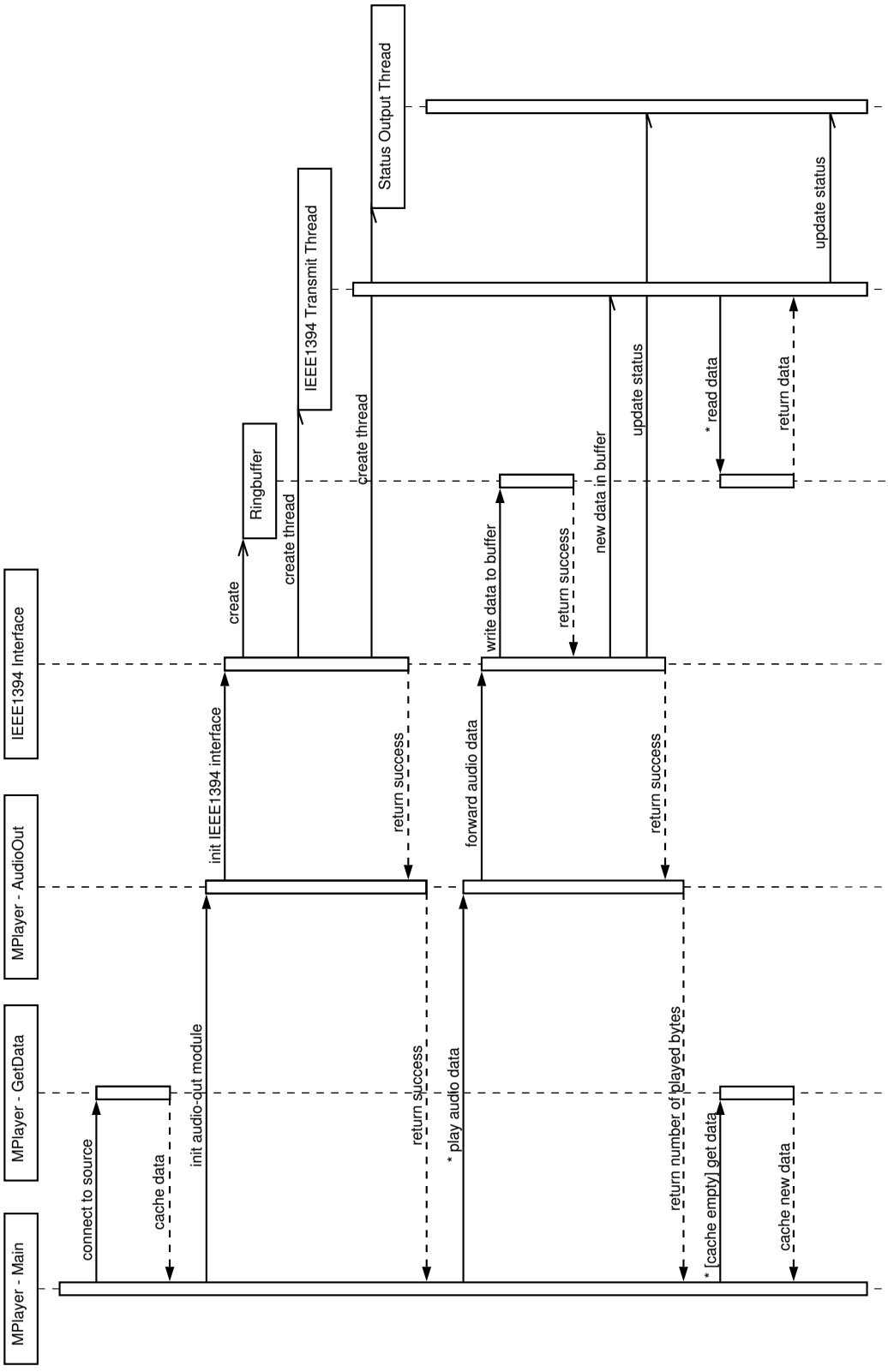

Abbildung 10: Sequenzdiagramm der Gateway-Software

Es existieren zwei Möglichkeiten, um die Unregelmäßigkeiten der Zeitpunkte der ankommenden Datensamples zu kompensieren und die Sendegeschwindigkeit anzupassen. Die Methoden setzen dabei an völlig verschiedenen Punkten an.

3.7.1 Variation der Samplingrate

Durch die Änderung der Anzahl der Samples pro Paket kann erreicht werden, dass im selben Zeitintervall mehr respektive weniger Daten an die IEEE 1394-Schnittstelle weitergeleitet werden, da die isochronen Pakete in konstanten Intervallen versendet werden. Natürlich muss dabei auch der Zeitstempel im SYT-Feld im CIP-Header angepasst werden, da sich die Zeitspanne zwischen zwei Samples nun geringfügig ändert.

Diese Maßnahme kann aber zu Problemen beim Endgerät führen, je nachdem, wie es programmiert wurde, mit solch speziellen Situationen - in diesem Fall die Abweichung von der anfangs festgelegten Samplingfrequenz - umzugehen. Sobald zum Beispiel mehr Samples als vorgesehen in ein isochrones Paket integriert werden, wird im eigentlichen Sinne die aktuelle Samplingfrequenz erhöht und das wird vom Endgerät sehr bald registriert. Die in diesem Projekt zum Testen verwendete Software amdtpplay[27] verwirft dann die zusätzlich gesendeten Samples, um die Nominalfrequenz aufrecht zu erhalten. Um die vorgesehene Wirkung der Synchronisation zu erzielen, müsste sich das Endgerät nur nach den Timestamps im SYT-Feld richten und dementsprechend auch die weiterverarbeitende Komponente, zum Beispiel die Soundkarte, ansteuern. Leider reagierte die professionelle HiFi-Anlage LISSA von Sony ähnlich wie amdtpplay auf die Variation der Samplingrate, jedoch schienen die kurzen Aussetzer etwas unauffälliger und seltener.

3.7.2 Modifikation der originalen Audiodaten

Um ein unvorhersehbares Verhalten des Endgeräts zu vermeiden, müsste man einen Weg finden, die Synchronisation gegenüber dem Endgerät transparent durchzuführen. Dabei nutzt man die Unwissenheit des Endgeräts über den Inhalt der ankommenden Audiosamples aus. Aufgrund seiner eingeschränkten Intelligenz besitzt das Endgerät keinerlei Informationen über die Art der Audiodaten, das heißt ob es sich nun um die stündlichen Nachrichten oder um eine Symphonie von Beethoven handelt, es würde in keinem Fall Modifikationen in den Originaldaten erkennen. Wenn zum Beispiel der Audiopuffer langsam aber beständig zur Neige geht, wiederholt man in bestimmten Abständen eine Sequenz von Events, sodass die originalen Daten länger im Puffer bestehen bleiben und dem Füllstand die Gelegenheit gegeben wird, sich durch neue Daten wieder auf Normalniveau zu regenerieren. Das Endgerät spielt nach wie vor die Samples in der anfangs übermittelten Nominalfrequenz ab und registriert die Veränderungen nicht. Natürlich muss darauf geachtet werden, dass die Daten nicht zu sehr verfälscht werden, um zu verhindern, dass das menschliche

[27] siehe Paragraph 4.1.3.4 auf Seite 66

Ohr aufgrund von Rhythmusschwankungen und drastischen Störungen in der Melodie die Veränderungen entdeckt.

Statt des Sendens einer exakten Kopie von Events könnte man auch leere Samples in das isochrone Paket integrieren, um sich die originalen Daten länger aufzusparen. Eine weitere Möglichkeit wäre die Interpolation zweier aufeinanderfolgender Samples des selben Kanals, um das daraus resultierende Sample zusätzlich dazwischen zu versenden.

Nun stellt sich nur noch die Frage, wie man mit einem aufgrund einer konstanten Steigung des Füllstands prognostizierten Überlauf des Puffers umgeht. Bei dem schon beispielhaft dargestellten sicherlich in der Mehrheit aller Unregelmäßigkeiten vorkommenden Leerlauf kann man mit Mehrfachsendungen dem Puffer auf einfache Art wieder auf die Sprünge helfen. Überläufe hingegen stellen vor das Problem, dass zu viel Information für ein bestimmtes Zeitintervall vorhanden ist, sodass man diese Informationsmenge verlustbehaftet reduzieren muss. Die einfachste Variante wäre das Verwerfen einiger Samples in bestimmten Abständen ohne zu große Informationsbrocken auf einmal zu verlieren, bis der Füllstand wieder Normalniveau erreicht hat. Genauso gut könnte man aber, anstatt nur ein Sample von zwei aufeinanderfolgenden zu senden, auch eine interpolierte Version der beiden berechnen. Ob dieses Verfahren die Qualität der Audioquelle weniger vermindert als die erste Variante, ist sicherlich von der Art der Audiodaten abhängig und bedarf weiterer Details dieses Gebiets, was jedoch den Rahmen dieser Arbeit sprengen würde.

3.7.3 Bewertung der beiden Varianten

Aufgrund der Tatsache, dass bei der zweiten Variante die originalen Audiodaten modifiziert und somit verfälscht werden, stellt diese Art der Synchronisation trotz der Transparenz gegenüber dem Endgerät keine gute Wahl für das Gateway dar. Die Aufgabe dieses Projekts liegt nur in der Konvertierung der Audiodaten, die Informationen selbst, die durch die Gesamtheit der empfangenen Daten entstehen, müssen erhalten bleiben, um das Risiko zu vermeiden, wichtige Teile davon unkenntlich zu machen oder gar zu zerstören.

Deshalb bleibt nur die Möglichkeit der Variation der Samplingrate. Wie das jeweilige Endgerät dann schließlich auf diese Veränderung der Wiedergabegeschwindigkeit reagiert und wie gut es damit zurecht kommt, bleibt ihm selbst überlassen. Professionelle Audiohardware ist eventuell dazu in der Lage, die Events unabhängig von der nominalen Samplingrate genau zu dem Zeitpunkt abzuspielen, für den sie die Gateway-Software im SYT-Feld vorgesehen hat. Ein anderer Grund, der für diese Variante spricht, wäre die Motivation, dass es für das Endgerät von höchster Wichtigkeit ist, dass die Originaldaten unverfälscht bei ihm ankommen, um sie weiteren Analysen und Berechnungen zu unterziehen. Dieses Projekt soll die Palette der endgültigen Anwendungsgebiete so breit wie möglich halten und die Anzahl auf keinen Fall einschränken.

Weitere Details zur praktischen Implementierung der Synchronisation werden in 4.4.5 beschrieben.

3.7.4 Synchronisation im MPlayer

Da die Audiodaten ja zuerst den Verarbeitungsweg des MPlayers passieren müssen, muss auch analysiert werden, wie dieser auf Über- und Leerläufe des Puffers reagiert. Die einzigen Konfigurationsmöglichkeiten, die dieses Thema betreffen, werden vor Aufruf der Software auf der Kommandozeile als optionale Parameter angeboten. Man kann die Größe des MPlayer-eigenen Zwischenspeichers und den Mindestfüllstand für den Beginn der Wiedergabe festlegen, aber die Samplingrate bleibt während der gesamten Laufzeit unverändert. Sobald keine Audiodaten mehr im MPlayer-Puffer zur Verfügung stehen, wird die Weiterleitung der Samples an das Ausgabemodul angehalten und gewartet, bis wieder neue Daten eintreffen. Der Vorteil an diesem Verfahren ist der, dass damit die in diesem Projekt entwickelte Synchronisation möglichst wenig beeinflusst wird, denn die Samples werden so schnell wie möglich (abhängig vom Ausgabemodul) und unmodifiziert vom MPlayer an den neu implementierten Teil weitergegeben.

Einer der Nachteile bei Verwendung des MPlayers ist die Tatsache, dass der IEEE 1394-Teil der Gateway-Software abgesehen von den grundsätzlichen Eigenschaften der Audiodaten, wie zum Beispiel Samplingrate usw., nur die nackten Audiosamples ohne Zusatzinformationen bekommt. Damit ist es nicht möglich, sich bei der Synchronisation nach den RTP-Timestamps zu richten, da diese vom MPlayer nicht weitergeleitet werden.

3.7.5 Praktische Synchronisation

Aus oben genannten Gründen und den Einschränkungen bei Verwendung des MPlayers bleibt somit nur die Möglichkeit, die Berechnungen für die Anpassung der Samplingrate aufgrund eines einzigen Kriteriums durchzuführen, und zwar die Geschwindigkeit, in der die Samples vom MPlayer in den Audiopuffer des IEEE 1394-Teils weitergeleitet werden, im Vergleich zur nominellen Samplingfrequenz. Dieses Verhältnis spiegelt sich im Füllstand des Audiopuffers wieder, da die Samples auf der einen Seite mit einer bestimmten Frequenz vom MPlayer in den Puffer hineingeleitet und auf der anderen Seite standardmäßig mit der nominellen Samplingrate entnommen und Richtung IEEE 1394-Bus gesendet werden. Dadurch genügt es, den Pufferfüllstand regelmäßigen Messungen zu unterziehen und bei eventuellen Abweichungen vom Normalzustand entsprechende sychronisierende Aktionen zu setzen.

3.8 Endgeräte

In den vielen verschieden ausgestatteten Endgeräten, die fähig sind, Audiodaten im AMDTP-Format zu interpretieren, passiert im Großen und Ganzen das selbe. Nachdem die IEEE 1394-Hardware einen bestimmten isochronen Kanal gewählt und die Ressourcen dafür reserviert hat, können die Daten auch schon empfangen werden. Aus den Headern der Pakete kann das Gerät die Metainformationen über die Audiodaten extrahieren, die es benötigt, um die weiteren Komponenten, wie zum Beispiel die integrierte Soundkarte zur Ausgabe über angeschlossene Lautsprecher,

korrekt zu konfigurieren. Im SYT-Feld des CIP-Headers befindet sich der notwendige Zeitstempel, der den genauen Zeitpunkt zur Präsentation der Samples darstellt. Ein standardgemäß implementiertes Endgerät sollte genau diese Information für das Timing verwenden, und nicht auf Basis der nominellen Samplingrate urteilen.

Eine Liste der Produkte aus Industrie, der Consumer- und der IT-Branche, die eine IEEE 1394-Schnittstelle enthalten, findet man in [31], wobei diese Aufzählung keinen Anspruch auf Vollständigkeit hat, aber doch einen großen Anteil an verfügbaren Produkten namhafter Hersteller umfasst.

4 Implementierungsdetails

In diesem umfangreichsten Teil der Arbeit sollen tiefergehende Details der einzelnen Teile des Projekts behandelt werden, ohne sich jedoch zu lange mit Programmiersprachen-spezifischen Einzelheiten aufzuhalten. Relevante implementierungstechnische Abläufe und Algorithmen werden zwar diskutiert, doch dabei wird ein gewisses sprachlich allgemeines Niveau aufrecht erhalten bleiben, ohne die Syntax der Programmiersprache dazu zu verwenden.

Zuerst werden alle Voraussetzungen - die Hardware- und Softwareumgebung - beschrieben, die für den produktiven Einsatz dieser Software notwendig sind respektive im Laufe der Implementierung getestet und eingesetzt wurden. Danach wird beschrieben, auf welche Art und Weise der MPlayer in die Gateway-Software beziehungsweise umgekehrt integriert wird. Weiters wird der gedankliche Werdegang bei der Suche nach einem vernünftigen Weg ins IEEE 1394-Netzwerk und schließlich die praktische Ausführung dieses Teils näher dargestellt, wobei auch Fehlentscheidungen und Sackgassen erwähnt werden, um die Entscheidungen, die zum endgültigen Aufbau der Software geführt haben, besser rechtfertigen zu können und verständlicher zu machen.

Das Verhalten während der Ausführungszeit wird hauptsächlich durch eine Konfigurationsdatei und die Parameter beim Aufruf definiert, die am Ende dieses Kapitels noch näher ausgeführt werden. Zur Überwachung der Vorgänge und eventuell auftretender Fehler wurde der grafische Log-Viewer entwickelt, der Informationen über den Pufferfüllstand, die Synchronisation und sonstige Ereignisse liefert.

4.1 Betriebsvoraussetzungen

In diesem Kapitel wird die Frage beantwortet, welche Komponenten und Voraussetzungen gegeben sein müssen, um den Versuchsaufbau nachzustellen respektive die Gateway-Software dann schließlich in der Praxis einsetzen zu können. Dabei sei darauf hingewiesen, dass keine allgemeine Aussage über die Funktionsweise des Programms in bestimmten typischen Umgebungen getroffen werden kann, da nur die genannten Ressourcen unmittelbar mit dem Gateway getestet wurden. Wie bei jeder anderen Softwarekomponente auch, muss die korrekte Funktion des Gateways in einer neuen individuellen Umgebung erst geprüft und eventuell angepasst werden und da keine zwei vollständig identen Einsatzumgebungen existieren, können auch bei scheinbar gleichen Bedingungen unvorhersehbare Fehler auftreten.

4.1.1 Datenquellen

Grundvoraussetzung für den Empfang öffentlicher Audioquellen ist natürlich ein Internetzugang, der abhängig von der Qualität der Quellen eine bestimmte Mindestgeschwindigkeit aufweisen sollte. Von den Radiosendern, die im Internet einen Live-Stream zur Verfügung stellen, werden die Audiodaten meist nur in geringen Bitraten angeboten, um möglichst viele Hörer gleichzeitig

bedienen zu können. Der Originaldatenstrom ist meistens durch eine Samplingrate von 44,1 kHz, einen oder zwei Kanäle und 16 Bit Samples charakterisiert, was eine Übertragungsgeschwindigkeit von 705,6 kBit/s respektive ungefähr 1,4 MBit/s in Stereo voraussetzen würde, wenn er ohne Komprimierung versendet werden würde. Um die Internetverbindungen bei diesen zwar mittlerweile möglichen aber doch sehr hohen Datenraten zu entlasten, werden die Audiodaten mit verschiedensten Verfahren[28] komprimiert und erst anschließend an die Clients geschickt. Streams mit den oben genannten Eigenschaften werden oftmals zu Bitraten von 64 kBit/s oder 128 kBit/s zusammengestaucht, natürlich mit teilweise erheblichen Qualitätsverlusten.

Somit würde ein Internetzugang mit einer kontinuierlichen maximalen Downloadgeschwindigkeit von 256 kBit/s vollkommen ausreichen, um sich die meisten verfügbaren Live-Streams ohne Unterbrechungen aufgrund von Pufferleerläufen anhören zu können. Wenn abgesehen von den Live-Streams noch regelmäßig andere Daten aus dem Internet heruntergeladen werden, wäre die Verwendung eines schnelleren Zugangs sicherlich sinnvoll. Bei den aktuell weit verbreiteten Breitband-Zugangstechnologien, wie zum Beispiel ADSL oder den Zugängen über die Infrastruktur der Kabelfernsehbetreiber, besteht aber ohnedies kein Problem mit den Geschwindigkeiten, da hier schon bei den Einsteigerprodukten weit höhere Downloadraten angeboten werden, als in den Einsatzgebieten dieses Projekts nötig wären. Lediglich Verbindungen über analoge Modems, ISDN-Adapter oder GSM- und GPRS-Handys erweisen sich als unzureichend für dieses Anwendungfeld. Neue mobile auf drahtloser Übertragung basierende Zugangstechnologien, wie IEEE 802.11 oder UMTS, die sich aktuell einer rasanten Verbreitung erfreuen, sind jedoch schon für das Audiostreaming geeignet und werden dafür auch immer intensiver verwendet, auch im Hinblick auf Voice-over-IP.

Echtes Quality of Service auf dem Weg zwischen der Quelle und dem Gateway-Rechner wäre sinnvoll, um das Risiko eines Pufferleerlaufs zu verringern, allerdings ist diese Garantie in einem IP-Netzwerk mit der Größe des Internets nahezu unmöglich beziehungsweise nur in bestimmten Umgebungen mit speziellen Technologien zwischen ausgewählten Partnern überhaupt realisierbar.[29]

Die Tests während der Implementierung wurden mit verschiedensten Audioquellen durchgeführt, hier seien nur einige davon genannt:

- Lokale Audiodateien auf der Festplatte des Gateway-Rechners
- MP3-Dateien von einem primitiven RTSP-Server, der von LIVE555 [37] für Testzwecke entwickelt wurde und auf einem Rechner im lokalen Netzwerk lief, der via Kabel mit brutto 100 MBit/s beziehungsweise drahtlos mit brutto 54 MBit/s (IEEE 802.11g) angebunden war.
- RealAudio-Dateien vom freien RTSP/RDT-Server Helix[30], der ebenfalls im lokalen Netzwerk situiert war.

[28] siehe Kapitel 2.3 auf Seite 17
[29] siehe Paragraph 2.5.1.1 auf Seite 33
[30] Helix Community, siehe https://helixcommunity.org

- Live-Streams von Radiosendern im Internet im RealAudio-Format, die zwar RTSP verstehen, aber als Übertragungsprotokoll nur RDT, sodass aufgrund von Inkompatibilitäten mit dem proprietären Protokoll von RealNetworks[31] nur ein Stream mit in RTSP-Paketen eingekapselten Audiodaten auf TCP-Basis möglich war:
  - Radio Salzburg: rtsp://realorf.conova.com/salzburg.ra (64 kBit/s)
  - Radio Lombardia: rtsp://81.208.58.101:554/encoder/live.rm (32 kBit/s)
  - Radio Noord: rtsp://spielberg.ision.nl/encoder/noord.rm (44 kBit/s)
- Live-Stream des Radiosenders FM4 im OGG-Vorbis-Format über HTTP:
  http://listen.fm4.amd.co.at:31337/fm4-hq.ogg (160 kBit/s)

4.1.2 Hardwareumgebung

Während der Entwicklungs- und anschließenden Testphase wurde die Infrastruktur, wie sie in Abbildung 11 gezeigt wird, verwendet. Das wichtigste Arbeitsgerät, das auch gleichzeitig als Plattform für das Gateway fungierte, war ein Laptop des Typs „Dell Inspiron 510m". Mit dem Intel Pentium M 1,7 GHz als Hauptprozessor und 512 MiByte RAM war genügend Rechenleistung und Hauptspeicher vorhanden, um die notwendige Software ohne zeitlich kritische Probleme während der Laufzeit zu betreiben. Nur in speziellen Situationen, für die sich hauptsächlich die grafische Anzeige des Log-Viewers verantwortlich zeigt, erreichte die Prozessorauslastung sehr hohe Prozentwerte[32]. Der Laptop besitzt abgesehen von einem internen WLAN-Adapter, der laut IEEE 802.11g maximal eine Datenübertragungsrate von brutto 54 MBit/s erreichen kann, und einer 100-MBit/s-schnellen Ethernetkarte einen IEEE 1394-Adapter, der auf dem Chipsatz PCI4510 von Texas Instruments basiert.

Grundsätzlich müsste jeder Computer, in den eine Netzwerkkarte und ein IEEE 1394-Adapter integriert ist und auf dem ein einwandfrei lauffähiges Linux-Betriebssystem installiert werden kann, als Gateway-Rechner geeignet sein. Selbstverständlich sollte das System auch die Performance und den Speicherausbau betreffend nicht zu knapp dimensioniert sein, aber für den Großteil der in den letzten Jahren produzierten Rechnern dürfte diese Voraussetzung gelten. Eindeutige Mindestkapazitäten der einzelnen verwendeten Komponenten lassen sich schwer definieren, da es immer auf die Gesamtkonstellation ankommt und sich auch Produkte mit gleicher nomineller Leistung, aber von verschiedenen Herstellern produziert, nicht immer miteinander vergleichen lassen. Es wird neben den schon genannten notwendigen Teilen eine minimale Ausstattung von 128 MiByte Hauptspeicher und einem 1 GHz schnellen Prozessor empfohlen. Die restlichen für ein Computersystem erforderlichen Komponenten sind für dieses Projekt eher nebensächlich und können frei gewählt werden. Viel wichtiger ist die Aktualität der installierten Software und eine möglichst

[31] siehe Kapitel 2.5.2 auf Seite 38 und 5.1.2 auf Seite 92
[32] siehe Kapitel 5.1.4 auf Seite 93

effiziente Konfiguration des Betriebssystems, um knappe Hardwareressourcen nicht unnötig zu verbrauchen[33].

Wie man in der Grafik 11 erkennen kann, besteht in der hauptsächlich verwendeten Testumgebung eine Verbindung über eine Outdoor-Richtfunkverbindung, die dem Standard IEEE 802.11b gehorcht, mit dem Internet-Service-Provider (ISP), wodurch sich höhere Latenzzeiten zu Servern im Internet ergeben als bei konventionellen kabelgebundenen Internetzugangstechnologien wie DSL. Bei auf UDP basierendem Streaming wirken sich diese längeren Antwortzeiten aufgrund der einseitigen Kommunikation weniger drastisch aus als bei TCP-Verbindungen, bei denen öfters auf Bestätigungen gewartet werden muss.

Im Wohngebäude des Autors wurde eine strukturierte Verkabelung auf Ethernet-Basis eingerichtet, die einen Router, der Network Address Translation (NAT) betreibt und über die Outdoor-Funkbrücke mit dem Provider kommuniziert, einerseits mit einem testweise installierten RTSP-Server und andererseits mit einem WLAN Access Point des Standards IEEE 802.11g verbindet. Die Reichweite des 54-MBit/s-schnellen Access Points ist jedoch auf die Gebäudegrenzen beziehungsweise geringfügig darüber hinausgehende Abschnitte beschränkt. Über diese drahtlose Heimnetzwerkverbindung spricht schließlich das Kernstück dieses Projekts - der Gateway-Rechner - mit der Außenwelt.

Der Client, der die Audiodaten letztendlich erhält und wiedergibt, besteht aus einem herkömmlichen PC mit mehr oder minder leistungsfähiger Hardware, deren wohl wichtigster Bestandteil der IEEE 1394-Adapter ist, um eine Verbindung mit dem Audioto1394-Gateway zu ermöglichen. Mit Hilfe geeigneter Software werden die empfangenen Audiodaten an die Soundkarte und damit an die externen Stereo-Lautsprecher weitergeleitet.

Als Alternative zu dem gerade genannten Endgerät stand am Institut für Computertechnik an der Technischen Universität Wien die HiFi-Anlage LISSA von Sony zur Verfügung, deren Komponenten untereinander über IEEE 1394 kommunizieren, sodass es auch möglich war, den Gateway-Rechner in den Bus zu integrieren. Da dieses Audiogerät den anfänglichen Verbindungsaufbau aber scheinbar über eine proprietäre Abfolge von AV/C-Kommandos bewerkstelligt, war es nur über einen Umweg möglich, den Receiver dazu zu überreden, die gesendeten Audiodaten wiederzugeben. Sobald die LISSA aber auf den richtigen isochronen Kanal zugriff, indem das Gateway vorgab, eine der anderen angeschlossenen Sony-Komponenten zu sein, funktionierte die Ausgabe problemlos, da die Audiodaten im IEC 60958-Format korrekt interpretiert wurden. Um den Verbindungsaufbau automatisiert durchführen zu können, ohne dass manuelle Modifikationen der IEEE 1394-Register des Receivers notwendig sind, muss man den asynchronen Datenverkehr zwischen den Sony-Geräten in der Implementierung der Gateway-Software nachahmen.

[33] siehe nächstes Kapitel

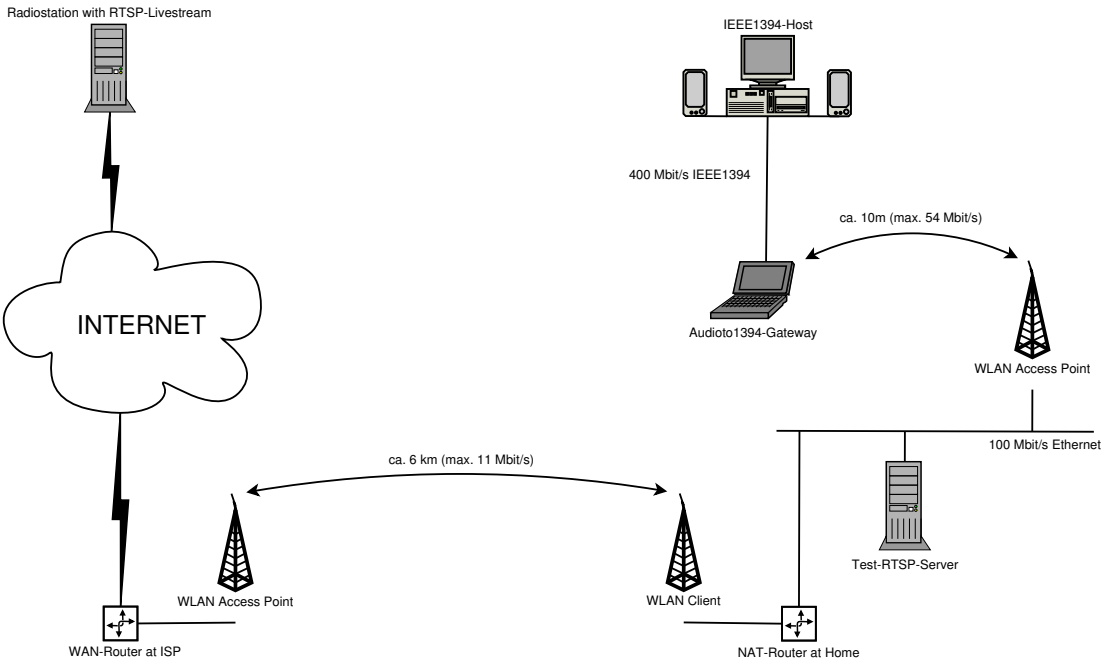

Abbildung 11: Infrastruktur in der Testphase

4.1.3 Softwareumgebung

Hier werden die softwaretechnischen Voraussetzungen für den Betrieb des Gateways beschrieben, aufsetzend auf der im letzten Kapitel dargestellten Hardware. Auch hier wird hauptsächlich auf die im Projekt verwendeten Softwareteile eingegangen, da über alternative nicht getestete Komponenten keine definitiven Aussagen gemacht werden können.

4.1.3.1 Basissystem Die gesamte Entwicklung und die Testphase fand auf dem Betriebssystem Linux statt. Nach anfänglicher Suche nach einem geeigneten Linux-System fiel die Wahl auf eine auf dem bekannten Debian-System basierende Distribution namens Ubuntu[34], die einerseits viele der bewährten Features von Debian, wie zum Beispiel die Paketverwaltung, und andererseits eine erweiterte automatische Hardwareerkennung und aktuellere Versionen der Softwarepakete mitbringt. Die im halbjährlichen Rhythmus aktualisierte Distribution wurde seit dem Start dieses Projekts ständig auf dem neuesten Stand gehalten, beginnend mit der Ausgabe 4.10 (Warty Warthog) über 5.04 (Hoary Hedgehog) bis zu 5.10 (Breezy Badger).

Updates auf einen neueren Kernel wurden in unregelmäßigeren Abständen durchgeführt, meist motiviert durch eine fehlende oder fehlerhafte Hardwareunterstützung. Während der gesamten Zeit waren jedoch nur Kernel des aktuellen Zweiges 2.6 im Betrieb, beginnend mit der Version 2.6.8.1 bei der Erstinstallation des Systems bis zur Version 2.6.12.5 bei den abschließenden Tests

[34] Canonical Ltd., siehe http://www.ubuntulinux.org

Algorithm 2 Installation der LIVE555 Library

```
# Ausführung als Benutzer root:
apt-get install liblivemedia-dev
```

und Debugging-Vorgängen. Laut dem Projekt Linux1394 [36] integriert der Kernel 2.6.12 offiziell die ersten wirklich stabilen IEEE 1394-Quellen, wobei vor der Kompilierung eines neuen Kernels immer die aktuellsten Quellen von der Projekthomepage geladen und die original in den Kernelquellen vorhandenen ersetzt wurden. Durch diese wiederholten Updates wurde die Gateway-Software spürbar stabiler, zusätzlich zur Erhöhung der Robustheit durch die ständige Weiterentwicklung der Software selbst.

Die Gateway-Software selbst läuft zwar auf der Konsole, doch während der Implementierung wurde nicht auf den Komfort einer graphischen Benutzeroberfläche verzichtet. Aus diesem Grund war der in Ubuntu mitgelieferte X-Server X.Org 6.8.2[35] mit dem Desktop-Manager GNOME in der Version 2.12.1[36] installiert und eingerichtet. Weiters muss ein X-Server zwingend zur Verfügung stehen, wenn die Verwendung des Log-Viewers aktiviert ist, der die aktuellen Vorgänge während der Laufzeit graphisch illustriert. Der Log-Viewer setzt auch noch weitere Entwicklungsbibliotheken voraus, und zwar die GTK+ Libraries in der Major-Version 2 zur Darstellung der Desktop-Elemente und die GThreads zur Verwendung von Multithreading in graphischen Umgebungen.

Das eben besprochene Basissystem mit Netzwerk- und IEEE 1394-Unterstützung muss bereits funktionstüchtig sein und möglichst stabil laufen, bevor mit der Installation der weiteren Komponenten fortgefahren werden kann. Diese notwendige Bedingung kann durch beliebige Belastungstests und die Überprüfung der nötigen Funktionen im System, wie zum Beispiel den Internetzugang und den X-Server, zumindest teilweise bestätigt werden.

4.1.3.2 Installation des MPlayer

Vor der Installation des MPlayers selbst muss zuerst die LIVE555 Multimedia Streaming Library [37] installiert werden. Diese Programmbibliothek und deren API[37] wird vom MPlayer benötigt, um Verbindungen zu Audioquellen, die auf RTP und RTSP basieren, aufbauen zu können. Falls es sich bei dem Kommunikationspartner aber um einen RTSP-Server von RealNetworks handelt, der unter anderem auch RealAudio- oder RealVideo-Streams anbietet, dann wird eine MPlayer-eigene Implementierung verwendet. In Debian-basierten Linux-Distributionen mit dem Paketverwaltungssystem „apt-get" funktioniert die Installation und Konfiguration der LIVE555 Library sehr einfach, wie man in Algorithmus 2 sehen kann. Falls in anderen Distributionen nicht ähnliche vorkonfigurierte Pakete existieren oder man die neueste Version verwenden will, muss man die Quellen manuell von der LIVE555 Homepage [37] beziehen - die aktuelle Version lautet 2006.01.05 -, selbst kompilieren und ins System integrieren.

Im nächsten Schritt müssen die Codecs für den MPlayer heruntergeladen und in ein vorgegebenes

[35] X.Org Foundation, siehe http://www.x.org
[36] GNOME: The Free Software Desktop Project, siehe http://www.gnome.org
[37] siehe Kapitel 2.2 auf Seite 14: API

Algorithm 3 Installation und Test des MPlayer

```
tar xvjf MPlayer-1.0pre7.tar.bz2
cd MPlayer-1.0pre7
./configure
make
# Tests mit Audioausgabe über lokale Soundkarte:
mplayer rtsp://realorf.conova.com/salzburg.ra
mplayer http://listen.fm4.amd.co.at:31337/fm4-hq.ogg
```

Verzeichnis kopiert werden, damit das Hauptprogramm sie bei der Konfiguration auch findet. Vorgefertigte Codec-Sammlungen findet man auf der MPlayer-Homepage [35] zum Download. Die Pakete stehen in verschiedenen Ausführungen zur Verfügung, je nachdem, welche Audioformate man zu verwenden beabsichtigt. Sobald sich die Codecs auf dem lokalen Rechner befinden, müssen sie in das Verzeichnis /usr/local/lib/codecs/ verschoben werden, um vom Konfigurationsskript auch korrekt gefunden zu werden.

Die Installation des MPlayer selbst darf nicht über die Paketverwaltung erfolgen, da auf diese Weise nur ein vorkompiliertes Binärpaket ins System integriert wird, bei dem die für dieses Projekt notwendigen Modifikationen nicht mehr durchgeführt werden können. Also bleibt nur die Option, sich die Quellen direkt von der MPlayer-Homepage [35] zu besorgen, wo man zusätzlich auch immer mit den aktuellsten Versionen bedient wird. Während der gesamten Entwicklungszeit bestand mit drei verschiedenen Versionen Kontakt, die jedoch alle Vorversionen von 1.0 waren und sich dementsprechend noch in einem gewissen unfertigen Status befanden. Trotzdem gab es an deren Stabilität und Funktionsvielfalt während der Arbeit an diesem Projekt nichts zu bemängeln. Einige Features, wie zum Beispiel die korrekte Behandlung von RTSP-Quellen mit RealNetworks-Inhalten, die in der derzeit neuesten Version 1.0pre7 noch nicht zufriedenstellend gelöst wurden, werden wohl erst in der finalen Fassung inkludiert sein.

Der Vorgang der Installation wird im Algorithmus 3 anhand der Version 1.0pre7 demonstriert. Nach der Ausführung des Konfigurationsskripts sollte in der Ausgabe des Skripts geprüft werden, ob alle gewünschten Codecs und vor allem die LIVE555 Library gefunden wurden und die Konfiguration erfolgreich beendet wurde.

Wiederum muss gewährleistet sein, dass der MPlayer bereits auf dem System einwandfrei lauffähig ist, bevor der IEEE 1394-Teil hinzugefügt wird, um danach klar differenzieren zu können, wo die Gründe für etwaige Fehler liegen. Deshalb sollte man noch einige Tests mit beliebigen Audioquellen durchführen und prüfen, ob sich diese Audiodaten über die lokale Soundkarte wiedergeben lassen. Auch die Verwendung von externen Audiodaten, wie zum Beispiel die eines RTSP- oder HTTP-Servers im Internet, wie es im Algorithmus 3 beispielhaft gezeigt wird, sollte in den Test miteinbezogen werden, um sicherzustellen, dass die Internetverbindung in Kombination mit dem MPlayer funktionsfähig ist.

Algorithm 4 Installation der LIBRAW1394 Bibliothek

```
# Ausführung als Benutzer root:
tar xvzf libraw1394-1.2.0.tar.gz
cd libraw1394-1.2.0
./configure –prefix=/usr
make
make install
```

4.1.3.3 Installation der IEEE 1394-Komponenten

In diesem Teil wird die Integration der LIBRAW1394 Bibliothek ins System diskutiert, wobei vorausgesetzt wird, dass das Kernelmodul RAW1394 bereits vorhanden und geladen ist. Falls das nicht der Fall sein sollte, muss zuvor der Kernel neu kompiliert werden und dabei die Option für dieses Modul aktiviert werden, die man im Optionsbaum der Kernelkonfiguration an folgender Stelle findet:

Device Drivers -> IEEE 1394 (FireWire) support -> Raw IEEE1394 I/O support

Für die in 4.3 diskutierte Lösungsvariante ist dieses Modul nicht notwendig, doch da diese Methode nicht zum Ziel geführt hat, werden hier sofort die vorausgesetzten Einstellungen für die endgültige Implementierung getroffen, die auch die korrekte Funktionsweise des RAW1394-Moduls beinhalten.

Mit dem Befehl „lsmod", der eine Liste der ins System geladenen Module in der Konsole ausgibt, kann man sicherstellen, dass das notwendige RAW1394-Modul bereits aktiv ist. Ähnlich dem AMDTP-Modul[38] wird der Zugriff auf die IEEE 1394-Ressourcen über ein Device namens „/dev/raw1394" realisiert, das dem üblichen Rechtemanagement unter Linux gehorcht. Das bedeutet, dass auch normalen Benutzern Lese- und Schreibrechte auf dieses Device verliehen werden müssen, damit die Ausführung der Gateway-Software nicht allein dem Systemadministrator „root" vorbehalten ist.

Jetzt kann die LIBRAW1394 Bibliothek mit der aktuellen Version 1.2.0 von der Linux1394 Homepage [36] heruntergeladen und, wie im Algorithmus 4 beschrieben, installiert werden. Der Grund für den zusätzlichen Parameter beim Aufruf von „configure" liegt in einer Eigenheit von Ubuntu-Linux, denn in dieser Distribution werden die Bibliotheken standardmäßig in /usr/lib/ und nicht in /usr/local/lib/ gesucht, deshalb muss der korrigierte Präfix übergeben werden.

4.1.3.4 Testsoftware und -hardware

Auf dem FireWire-Host, der die isochronen Audiodaten empfängt, muss eine Softwarekomponente aktiv sein, die den Datenstrom von der IEEE 1394-Schnittstelle entgegennimmt und zum Beispiel der Soundkarte zur Wiedergabe übergibt. Genau diese Aufgabe erledigt das Programm amdtpplay, das man von der Linux1394 Homepage [36] beziehen kann.

Mit dem Utility 1394commander, das ebenfalls von der genannten Homepage zur Verfügung gestellt wird, kann man einige Hintergrundinformationen über die Nodes auf dem IEEE 1394-Bus

[38] siehe Kapitel 4.3 auf Seite 71

erfahren und auch manuell Befehle absetzen, um zum Beispiel einen Bus-Reset zu initiieren oder bestimmte Register neu zu setzen.

Die anschaulichste Demonstration dieses Projekts stellt aber sicherlich die Wiedergabe von Internetradio-Streams auf der i.LINK-HiFi-Anlage von Sony namens LISSA dar, die für Testzwecke im Institut für Computertechnik zur Verfügung gestellt wurde. Denn damit wird eines der Ziele der Gateway-Software klar ersichtlich: Die Erweiterung der Zugriffsmöglichkeiten auf Audioquellen ausgehend von stark eingeschränkten Endgeräten.

4.1.3.5 Entwicklungswerkzeuge Für die Entwicklung der Gateway-Software wurden noch weitere softwaretechnische Hilfsmittel verwendet, von denen die einen unumgänglich für den Erstellungsprozess waren und die anderen zur Erhöhung der Effizienz dienten.

- gedit 2.12.1: ein Editor, der für den GNOME-Desktop entwickelt wurde und Syntax-Highlighting beherrscht
- Anjuta 1.2.4: eine effiziente IDE (Integrated Development Environment) für den GNOME-Desktop
- GCC 3.4.5: ein Präprozessor, Compiler, Assembler und Linker für die Programmiersprachen C und C++
- make 3.80: ein Werkzeug zur Interpretation und Ausführung der Makefiles
- Ethereal 0.10.12: ein Netzwerksniffer, der Netzwerkpakete komplett mitprotokolliert, sie in übersichtlicher Form darstellt und dadurch bei der Analyse von RTP- und RTSP-Paketen geholfen hat

4.2 Integration in den MPlayer-Quellcode

Ursprünglich wurde der Gedanke verfolgt, den äußerst umfangreichen Quellcode des MPlayers so zu überarbeiten und zu vereinfachen, dass alle für dieses Projekt unnötigen Komponenten, wie zum Beispiel die Videoverarbeitung und -wiedergabe, in der endgültigen Version des Gateways nicht mehr vorkommen und somit die Fehleranfälligkeit und Komplexität verringert wird. Im Prinzip sollten ausschließlich die Teile extrahiert werden, die unbedingt für den Betrieb der Gateway-Software notwendig sind. Dazu wurde der MPlayer Schritt für Schritt gekürzt, um einerseits die Funktionsweise besser zu verstehen und andererseits die korrekte Ausführbarkeit nicht zu gefährden. Kurz zusammengefasst wurden dabei folgende Aktionen durchgeführt:

1. Für diese Projekt nicht relevante Verzeichnisse und Dateien wurden entfernt.

2. Viele der übrig gebliebenen Quellcodedateien wurden gekürzt und modifiziert. Zum Beispiel wurde beim ersten Versuch der Lösungsvariante mit dem AMDTP-Kernelmodul das Hauptprogramm mplayer.c radikal gekürzt und an die Verwendung der Funktionen, die das AMDTP-Modul ansprechen, angepasst.

3. Die Konfigurationsdateien und Makefiles wurden an die neuen Begebenheiten angepasst.

Dieser äußerst aufwendige Prozess der Vereinfachung führte im Endeffekt zu wenig Erfolgen, da die einzelnen Codefragmente im MPlayer sehr stark miteinander verwoben und dadurch nur schwer oder überhaupt nicht zu trennen waren. Weiters findet man bei genauerer Analyse des Codes sehr viele verschiedene Programmierstile und wenig Dokumentation, was sich zusätzlich erschwerend auf das Verständnis auswirkte. Im weiteren Testverlauf zeigten sich dann öfters Fehler und Abstürze, deren Ursache undefinierbar war, die sich aber mit großer Wahrscheinlichkeit auf die drastischen Änderungen des MPlayer-Codes zurückführen ließen.

Aus den gerade genannten Gründen wurde dann schließlich beschlossen, einen anderen Weg zu gehen, und zwar die Anpassung des eigenen Codes an die Struktur des MPlayers, wobei möglichst wenig Modifikationen an den originalen Quellcode-Dateien durchgeführt werden sollten. Dadurch ergibt sich ein gravierender Vorteil, und zwar die relativ einfache Integration des Sourcecodes für das IEEE 1394-Gateway in zukünftige MPlayer-Versionen, sofern sich nicht drastische Änderungen in der Architektur des MPlayers ergeben. Sobald also neue Audioformate von den Programmierern des MPlayer-Teams integriert respektive Fehler in vorhandenen Implementierungen behoben werden, bedeutet es nur geringen Aufwand, die vorhandene Version durch diese erweiterte zu ersetzen. Durch neue Audioquellen wächst somit auch der Funktionsumfang der Gateway-Software, ohne diese explizit anpassen zu müssen.

Der Clou bei dieser Art und Weise der Integration ist die Vorspiegelung falscher Tatsachen gegenüber dem MPlayer, denn während der Einrichtung des Gateways wurde das MPlayer-Modul, das ursprünglich für die Ausgabe von Audiodaten in eine Datei verantwortlich war, durch eine modifizierte Version ersetzt, in der die eingehenden Daten an spezielle Funktionen zur Behandlung der IEEE 1394-Schnittstelle weitergeleitet werden. Somit wird nur an zwei Stellen direkt in den Originalcode des MPlayers eingegriffen, erstens in das Ausgabemodul ao_pcm.c und zweitens in das allgemeine Makefile, um die zusätzlichen IEEE 1394-Bibliotheken einzubinden. Bei allen Modifikationen, die detailliert noch in den nächsten drei Unterabschnitten diskutiert werden, galt als oberstes Motto: Möglichst unabhängig vom MPlayer-Quellcode bleiben!

Nachdem alle neu implementierten Bibliotheken zum Verzeichnisbaum des MPlayers hinzugefügt und alle sonstigen Code-Veränderungen durchgeführt wurden, genügt es, mit einem „make” im Hauptverzeichnis den Compiler und Linker zu starten, um die Integration schließlich zu vollenden und das fertig ausführbare Gateway-Programm zu bilden. Der MPlayer ist somit auf normale Art und Weise, so wie es die offizielle Dokumentation beschreibt, verwendbar. Das neue Feature des Schreibens auf den IEEE 1394-Bus ersetzt zwar die Ausgabefunktion in eine Datei, jedoch wird

es in einer der nächsten Versionen der Software in der Konfigurationsdatei die Möglichkeit geben, mit Hilfe eines Parameters zwischen dem Originalzustand und dem modifizierten zu wechseln.

4.2.1 Ausgabemodul ao_pcm.c

Wie schon zuvor kurz erwähnt, ist der Zweck der originalen Datei ao_pcm.c, die ihren Standort im Unterverzeichnis /libao2 des MPlayer-Hauptverzeichnisses hat, durch die Ausgabe der Audiodaten in eine Datei definiert. Übergibt man beim Aufruf des MPlayers den optionalen Parameter „-ao pcm", dann wird dieses Ausgabemodul in Kraft gesetzt und statt standardmäßig der Soundkarte die dekodierten Audiosamples zu übergeben, werden sie in eine Datei geschrieben. Zusätzlich wird am Anfang der Datei noch ein konformer Waveheader eingefügt, der über das Format der Audiodaten informiert, damit die entstandene unkomprimierte WAV-Datei auch von beliebiger Audiosoftware abgespielt werden kann. Benötigt man nur die rohen Samples ohne zusätzliche Informationen, wie es in diesem Projekt der Fall ist, muss man den Parameter auf „-ao pcm:nowaveheader" erweitern.

Natürlich gäbe es die Möglichkeit, ein eigenes Ausgabemodul zu erstellen und es dem MPlayer bekannt zu machen. Dafür müsste man jedoch weitaus tiefer in die Struktur des MPlayers eindringen und weit mehr Modifikationen am Originalcode durchführen, was aus schon genannten Gründen nicht sinnvoll ist. Der Hauptgrund, warum die Wahl genau auf dieses Modul als Ausgangspunkt für weitere Modifikationen fiel, besteht darin, dass es auf sehr einfache Art die Arbeitsweise des MPlayers demonstriert und es sich leicht an die neue Situation - den Übergang ins IEEE 1394-Netzwerk - anpassen lässt.

Abgesehen von den Parametern, die beim Aufruf des Programms übergeben werden[39], stellt diese Datei die einzige programmiertechnische Kommunikationsmöglichkeit mit dem Hauptmodul des MPlayers dar. Deswegen wird hier die Bedeutung der wichtigsten vorkommenden Funktionen genauer beschrieben [29]:

- init(): In dieser Funktion wird das Ausgabegerät initialisiert. Zu diesem Zweck erhält sie vom MPlayer Informationen über die kommenden Audiodaten, und zwar die Samplingrate, die Anzahl der Kanäle und das Sampleformat. Weiters wird festgelegt, wie groß die Datenblöcke - die Bursts - maximal sein sollen, die bei jedem Aufruf der play()-Funktion übergeben und anschließend an den IEEE 1394-Audiopuffer weitergeleitet werden. Der hier festgelegte Wert kann in der Konfigurationsdatei[40] angepasst werden. Zurückgeliefert wird der Erfolg der Initialisierung.

- play(): Hier werden die Audiodaten in den Puffer des jeweiligen Ausgabegeräts zur Weiterverarbeitung geschrieben. Die Anzahl der tatsächlich verwendeten Bytes des aktuellen Datenblocks wird zurückgegeben. Da in diesem Projekt die Daten an den Audiopuffer des

[39] siehe Kapitel 4.6.1 auf Seite 88
[40] siehe Kapitel 4.5 auf Seite 83

IEEE 1394-Teils übergeben werden und dort immer genügend Platz für einen vollen Datenblock zur Verfügung steht, wird hier immer die gesamte Größe des eingehenden Datenblocks zurückgeliefert.

- uninit(): Alle belegten Ressourcen werden hier freigegeben und das Ausgabegerät wird geschlossen.
- get_delay(): Wenn der MPlayer diese Funktion aufruft, muss ihm die Zeitspanne zurückgegeben werden, die bezogen auf die Wiedergabezeit zwischen dem ersten und dem letzten Sample im Audiopuffer des Ausgabegeräts liegt.
- get_space(): Diese Funktion meldet dem MPlayer, wieviel verfügbarer Speicherplatz noch im Puffer des Ausgabegeräts vorhanden ist, damit er festlegen kann, wie groß der nächste Datenblock für die play()-Funktion sein soll, ohne den Puffer zu überfüllen. Nach der Anpassung dieser Funktion an den IEEE 1394-Teil liefert sie aus dem Grund, der schon bei der play()-Funktion beschrieben wurde, immer die maximale in der init()-Funktion definierte Datenblockgröße zurück.

4.2.2 Makefile

Das zentrale Makefile des MPlayers, das sich in dessen Hauptverzeichnis befindet, muss modifiziert werden, um die neu hinzugefügten Bibliotheken in die endgültige ausführbare Datei einzubinden. Zusätzlich müssen auch die für den Betrieb von GTK+-Anwendungen[41] notwendigen Bibliotheken integriert werden, um die Anzeige des Log-Viewers zu ermöglichen.

Im Algorithmus 5 findet man die nötigen Ergänzungen, die an beliebiger Stelle nach den erstmaligen Definitionen der Variablen COMMON_LIBS, COMMON_DEPS und PARTS eingefügt werden können. Die vorausgesetzten Bibliotheken für GTK+ und die GThreads werden mit dem Befehl „pkg-config" direkt beim Kompiliervorgang eruiert und in die Kommandozeile eingesetzt.

4.2.3 lib1394

Das Verzeichnis /lib1394, das ins Hauptverzeichnis des MPlayers kopiert werden muss, enthält den gesamten Quellcode, der für die IEEE 1394-Komponente des Gateways verantwortlich ist, abgesehen von der LIBRAW1394-Bibliothek, die bereits im System installiert sein muss. Auch die originalen und modifizierten Teile der LIBIEC61883-Bibliothek befinden sich darin. Durch ein eigenes Makefile in diesem Verzeichnis für alle neu hinzugefügten Komponenten wird ein höheres Maß an Flexibilität und Unabhängigkeit erreicht. Dieses erzeugt ein Archiv namens lib1394.a, das im zentralen Makefile des MPlayers komplett integriert wird.

[41] The Gimp Toolkit, siehe http://www.gtk.org

Algorithm 5 Ergänzungen zum MPlayer-Makefile

```
include lib1394/config.mak
ifeq ($(GTKVIEWER),yes)
        GTK2_LIBS = `pkg-config –libs gtk+-2.0` `pkg-config –libs gthread-2.0`
else
        GTK2_LIBS =
endif
IEEE1394_LIBS = lib1394/lib1394.a -lraw1394 -L/usr/lib
COMMON_LIBS += $(IEEE1394_LIBS) $(GTK2_LIBS)
COMMON_DEPS += $(IEEE1394_LIBS)
PARTS += lib1394
lib1394/lib1394.a:
        $(MAKE) -C lib1394
```

Algorithm 6 Laden des AMDTP-Kernelmoduls

```
# Ausführung als Benutzer root:
modprobe amdtp
mknod -m 666 /dev/amdtp -c 171 48
```

4.3 Lösungsvariante AMDTP-Kernelmodul

Bei diesem ersten Lösungsansatz wurde der IEEE 1394-Teil des Gateways mit Hilfe des AMDTP-Kernelmoduls des aktuellen 2.6er Kernel realisiert. In den meisten Fällen wird es notwendig sein, den Kernel - in diesem Fall die Version 2.6.8.1 - neu zu kompilieren und davor in der Konfiguration den AMDTP-Support am besten modular zu aktivieren. Zu finden ist diese Kernel-Option auf folgendem Ast im Konfigurationsbaum:

Device Drivers -> IEEE 1394 (FireWire) support -> IEC61883-1 Plug support -> IEC61883-6 (Audio transmission) support

Sobald der Kernel neu gebildet und das AMDTP-Modul somit vorhanden ist, kann es auch gleich ins System eingebunden und auch das zugehörige Device erzeugt werden, wie es im Algorithmus 6 dargestellt ist, falls das nicht schon automatisch vom System durchgeführt wurde. Wichtig dabei ist die korrekte Konfiguration der Rechte des Devices, denn standardmäßig kann nur der Benutzer root es öffnen und mit ihm kommunizieren.

In der Datei amdtp_handler.c wurden die neuen Funktionen für die Behandlung des AMDTP-Moduls implementiert. Dazu benötigte Headerdateien waren amdtp.h und ieee1394-ioctl.h, die der Kernel mitbringt beziehungsweise auch auf der Linux1394-Homepage [36] in den Kernelquellen zu finden sind. Der gesamte Zugriff auf die IEEE 1394-Schnittstelle besteht nur aus drei kurzen Funktionen:

- amdtp_init(): Hier wird die IEEE 1394-Schnittstelle mit dem IOCTL-Kommando initialisiert und zuvor einige Parameter gesetzt, wie die Samplingrate, die Kanalanzahl, den iso-

chronen Kanal, das AMDTP-Format der Audiosamples und den isochronen Sendemodus[42]. Dabei wird ein gewöhnlicher Filestream erzeugt, auf den man auf übliche Art und Weise Daten schreiben kann.

- amdtp_write(): Mit dieser einzeiligen Funktion wird die übergebene Menge an Bytes mit der Standardfunktion fwrite() auf den Stream geschrieben. Der Kernel regelt alle weiteren IEEE 1394-spezifischen Vorgänge.
- amdtp_close(): Der Stream muss am Ende auch ordnungsgemäß geschlossen werden.

Die Vorteile in der Verwendung des AMDTP-Kernelmoduls liegen klar auf der Hand: die sehr simpel gehaltene Benutzerschnittstelle, die der allgemeinen Funktionsweise von Devices unter Linux entspricht, erfordert wenig bis gar kein Zusatzwissen über den verwendeten IEC-Standard 61883-6. Man behandelt dieses Device genauso wie eine Soundkarte und muss daher nur die Audio-Parameter - Samplingrate, Sampleformat und die Anzahl der Kanäle - und den isochronen Kanal, auf dem die Audiodaten gesendet werden, festlegen, und schon kann man mit dem Schreiben der Samples beginnen.

Doch diese Einfachheit geht einher mit dem Verlust des kontrollierten Zugriffs auf die niederen Ebenen, in denen das isochrone Paket selbst aus seinen Einzelteilen zusammengesetzt wird, und da in diesem Projekt sehr gewissenhaft auf das präzise Timing der Samples geachtet werden muss, kann diese Vorgehensweise das vorliegende Problem nur teilweise und unzufriedenstellend lösen. Der Grund dafür ist die nicht modifizierbare Samplingrate und der automatisch gesetzte Wert des SYT-Feldes während der Ausgabe, die keine anpassungsfähige Synchronisation erlauben. Das heißt, dass die Anzahl der Samples pro isochronem Paket vom Kernelmodul vorgegeben wird und nicht dynamisch an die Geschwindigkeit der liefernden Quelle angepasst werden kann.

Ein weiterer gravierender Nachteil ist die Instabilität des Kernelmoduls, die sich schon während der wenigen Tests damit in Form von Programmabstürzen und fehlender Datenübertragung offenbarte. Auf Verbesserungen und Behebung dieser auftretenden Fehler darf man aber nicht mehr hoffen, da das Modul nicht mehr intensiv weiterentwickelt wird. Die Aufgabe des AMDTP-Kernelmoduls wird vollständig der Bibliothek LIBIEC61883 übertragen, die im nächsten Kapitel für die Weiterführung dieses Projekts verwendet wird.

4.4 Lösungsvariante LIBIEC61883

Die Bibliothek LIBIEC61883 setzt im Unterschied zum AMDTP-Kernelmodul auf dem Kernelmodul RAW1394 auf, mit dessen Hilfe man auf tieferer Ebene auf die IEEE 1394-Schnittstelle zugreifen kann. Der Nachteil dieses flexibleren Verfahrens besteht darin, dass der Programmierer die isochronen Pakete inklusive Header selbst zusammenstellen und dazu natürlich mit den verwendeten Standards bestens vertraut sein muss.

[42] siehe Paragraph 2.4.5.2 auf Seite 29

Algorithm 7 Include-Anweisungen für die LIBRAW1394 und LIBIEC61883

```
#include <libraw1394/raw1394.h>
#include <libiec61883/iec61883.h>
```

In den schon fertig implementierten benutzerfreundlichen Funktionen der LIBIEC61883 werden dem Programmierer schon viele dieser standard-spezifischen Entscheidungen abgenommen, jedoch wiederum zum Preis des Verlustes der Eingriffsmöglichkeiten, das heißt auch hier ist die Anzahl der Samples pro Paket standardmäßig nicht frei wählbar. Gegenüber dem AMDTP-Kernelmodul ist die Modifikation dieser vorgegebenen Algorithmen jedoch wesentlich einfacher durchzuführen, da die Bibliotheksfunktionen auf User-Level und nicht auf Kernelebene eingebunden werden. So war es möglich, eigene Ideen in die Funktionen, die für den Aufbau des CIP-Headers und des Sampleformats zuständig sind, einzubauen und so die für dieses Projekt unumgängliche Variation der Samplingrate während der Laufzeit zu ermöglichen.

Voraussetzung für eine korrekte Funktionsweise der LIBIEC61883 ist eine ordnungsgemäße Installation einer aktuellen Version der LIBRAW1394, die in Algorithmus 4 demonstrativ durchgeführt wurde. Zusätzlich zur allgemeinen Headerdatei der LIBIEC61883 namens iec61883.h muss auch die Datei für die Behandlung des RAW1394-Moduls, die durch die Installation der LIBRAW1394-Bibliothek dem System hinzugefügt wurde, in die zuständigen Quellen eingebunden werden, wie in Algorithmus 7 gezeigt wird.

Da der Quellcode der LIBIEC61883 aber aus schon genannten Gründen teilweise modifiziert werden muss, werden die Dateien nur innerhalb eines Unterverzeichnisses des Pfades /<MPlayer-Hauptverzeichnis>/lib1394/ gepflegt, und aus Gründen der zu starken Spezialisierung für dieses Projekt nicht systemweit als Bibliothek installiert. Sollte deshalb eine andere Anwendung die LIBIEC61883 zur korrekten Ausführung benötigen, muss die unveränderte Version von [36] bezogen und laut den offiziellen Anweisungen installiert werden.

4.4.1 Funktionsweise der LIBIEC61883

Die Bibliothek setzt die Funktionen der LIBRAW1394 ein, um IEEE 1394 um den zusätzlichen Standard IEC 61883 zu erweitern und dem Programmierer die Möglichkeit zu geben, auf einfache Art und Weise diesen Standard in eigene Projekte zu integrieren.

Nach der Initialisierung der IEEE 1394-Schnittstelle und der Definition der Audioparameter für die Header kann der Sendevorgang auch schon gestartet werden. Die Integration der Audiodaten in die einzelnen isochronen Pakete wird über eine Callbackfunktion realisiert. Das bedeutet, dass eine hardwarenahe Funktion der LIBRAW1394 in regelmäßigen Intervallen diese Callbackroutine aufruft, in der der Programmierer die Audiosamples für genau ein Paket an eine übergebene Speicheradresse kopiert, auf die die ursprüngliche Funktion Zugriff hat. Diese stellt die Pakete mit allen Headern dann endgültig zusammen und sendet durchschnittlich alle 125 μs eines davon in den dafür vorgesehenen isochronen Kanal auf den IEEE 1394-Bus.

Die korrekte Zusammenstellung des CIP-Headers übernimmt in der unveränderten Version ein Teil der LIBIEC61883, jedoch wird für die Timestamps im SYT-Feld die nominelle Samplingrate als Basis zur Berechnung herangezogen, was, wie schon erwähnt, für dieses Projekt unzureichend ist. Aus diesem Grund wurde der originale Quellcode in der Datei cip.c der Bibliothek an die Anforderungen angepasst.

4.4.2 Beschreibung der Quellcodedateien

In diesem Unterkapitel wird kurz die Verzeichnis- und Dateistruktur der zum MPlayer hinzugefügten Quellcodedateien beschrieben:

Im Hauptverzeichnis des MPlayers wurde ein Unterverzeichnis /lib1394 angelegt, in dem sich alle Codesequenzen befinden, die für die Kommunikation mit der IEEE 1394-Hardware und die Verwaltung der sonstigen notwendigen Komponenten verantwortlich sind:

- **amdtp_transmit.c:** In diesem zentralen Modul befinden sich die Initialisierung der IEEE 1394-Schnittstelle, die Callbackfunktion für das Befüllen der isochronen Pakete, die Verwaltung des Multithreading und des Audiopuffers, die Schnittstelle zum MPlayer und die Kontrolle über den Log-Viewer[43].
- **readcfg.c:** Hier werden die Parameter aus der Konfigurationsdatei[44] eingelesen, ausgewertet und in Variablen gespeichert. Man könnte hier natürlich auf die schon von MPlayer mitgelieferten Funktionen zum Einlesen von Textdateien zurückgreifen, aber diese könnten in zukünftigen Versionen modifiziert werden und damit Probleme bei der Verarbeitung der hier verwendeten projektspezifischen Parameter hervorrufen. Außerdem soll soviel Abstand wie möglich vom MPlayer gehalten werden und so wurde eine eigene primitive Funktion für diesen Zweck entwickelt.
- **ringbuffer.c:** Die grundlegenden Funktionen zur Verwendung des Ringpuffers, wie das Initialisieren, Lesen, Schreiben und Abfragen des Füllstands, werden in dieser Datei gespeichert.
- **statout.c:** Alle Funktionen, die die Statusausgabe und die Verwaltung des Log-Viewers betreffen, findet man hier.

Im Verzeichnis /<MPlayer-Verzeichnis>/lib1394/libiec61883/ werden die modifizierten Dateien der LIBIEC61883 gespeichert, wobei folgende Dateien vom Originalzustand abweichen:

- **amdtp.c:** Änderungen wurden in der Funktion amdtp_xmit_handler() vorgenommen, um eine detailliertere Auswertung der Rückgabewerte der Callbackfunktion zu ermöglichen.

[43] siehe Kapitel 4.6.3 auf Seite 90
[44] siehe Kapitel 4.5 auf Seite 83

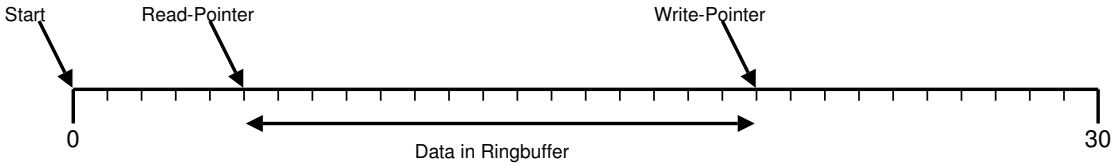

Abbildung 12: Ringpuffer

- **cip.c:** Durch die Modifikationen wird eine präzisere Synchronisation des Audiopuffers erreicht.

4.4.3 Audiopuffer

Der Puffer, der die Audiosamples vom MPlayer in Empfang nimmt, zwischenspeichert und schließlich dem IEEE 1394-Teil der Gateway-Software übergibt, muss als FIFO[45]-Puffer ausgelegt sein, um den vorliegenden Ansprüchen zu entsprechen. Die Idee des Ringpuffers eignet sich dafür ausgezeichnet, da er ohne unnötige Speicherzugriffe und -verschiebungen die ankommenden Schreib- und Leseanfragen effizient abarbeitet und schnell beantwortet.

Dafür verantwortlich ist eine Datenstruktur, bestehend aus einem Zeiger[46] auf den Beginn des zuvor fix reservierten Speicherbereichs und zwei zusätzlichen Zeigern, die jeweils auf die aktuelle Lese- und Schreibposition verweisen. Wenn neue Daten empfangen werden, werden diese an der aktuellen Schreibposition innerhalb des Puffers angehängt und danach der Schreib-Zeiger entsprechend angepasst. Ähnlich verhält es sich beim Lesevorgang, bei dem, nachdem eine bestimmte Menge an Daten gelesen wurden, der Lese-Zeiger einfach weitergeschoben wird ohne die Daten tatsächlich zu löschen. Die Bezeichnung Ringpuffer stammt aus der Tatsache, dass, sobald die Zeiger ans Ende des reservierten Bereichs gelangen, sie wieder an den Anfang gesetzt werden. Sobald sich Nutzdaten im Puffer befinden, muss sich aus logischen Gründen die Leseposition immer links von der Schreibposition befinden, wenn man sich den Speicherbereich als waagrechte Linie mit der Startadresse am linken Ende vor Augen führt, wie in der Abbildung 12 beispielhaft demonstriert wird.

Eine Ausnahme dieser Grundregel ergibt sich in der speziellen Situation, in der der Schreib-Zeiger bereits zum Anfang zurückgesprungen ist und der ihm folgende Lese-Zeiger noch nicht. Diese Eigenschaften müssen in den Funktionen, die im Ringpuffer operieren und auch Statusinformationen wie den noch freien Speicherplatz und die Konsistenz zurückliefern, berücksichtigt werden.

Die für die Verwendung des Ringpuffers notwendigen Funktionen und Variablen befinden sich in den Dateien ringbuffer.c und der zugehörigen Headerdatei ringbuffer.h. Die Größe des für den Puffer zu reservierenden Speicherbereichs kann in der Konfigurationsdatei[47] vor dem Start des

[45] siehe Kapitel 2.2 auf Seite 14: FIFO
[46] siehe Kapitel 2.2 auf Seite 14: Zeiger
[47] siehe Kapitel 4.5 auf Seite 83

Gateways fixiert werden.

4.4.4 Multithreading

Für das Multithreading und die Synchronisation zwischen den zur gleichen Zeit laufenden Komponenten werden POSIX[48] Threads und deren Bibliotheksfunktionen verwendet. Nahezu die gesamte Thread-Programmierung findet in der Datei amdtp_transmit.c statt, bis auf den Thread für die Statusausgabe im Log-Viewer, der in statout.c implementiert wird. Da der MPlayer selbst kein Multithreading verwendet, sollte auch keine seiner Quellcodedateien damit in Berührung kommen.

Grundsätzlich konkurrieren drei gleichzeitige Threads in der LIB1394 um die vorhandenen Ressourcen, vor allem um den Zugriff auf den gemeinsamen Audiopuffer:

1. **Transmit-Thread:** Dieser Thread wird bereits beim Initialisieren der RAW1394-Schnittstelle gestartet und ist dafür verantwortlich, dass die Daten vom Audiopuffer an die IEEE 1394-Schnittstelle weitergegeben werden. Dabei überwacht er den Status des RAW1394-Dateideskriptors und ruft bei Bedarf die Funktion raw1394_loop_iterate() auf, die sich ihrerseits um die hardwarenahen Aufgaben und im Zusammenhang damit auch um die Aufrufe der Callbackfunktion kümmert. Im Gegensatz zum zweiten Thread ist dieser während der gesamten Laufzeit der Gateway-Software aktiv.

2. **MPlayer-Thread:** Sobald dem MPlayer neue dekodierte Audiosamples von der Quelle zur Verfügung stehen und er sie dem Ausgabemodul übergeben möchte, nimmt eine Funktion der LIB1394 diese Daten entgegen und kopiert sie mit Rücksicht auf andere Zugriffe in den Audiopuffer. Ist dies erledigt, wird die Kontrolle dem MPlayer-Hauptprogramm wieder zurückgegeben. Falls der Audiopuffer schon einen Füllstand erreicht haben sollte, bei dem der aktuelle Datenblock vom MPlayer keinen Platz mehr findet, wird der Thread für eine bestimmte Zeitspanne angehalten, bis sich der Puffer durch die Aktivitäten des Transmit-Threads wieder geleert hat.

3. **Statout-Thread:** Der Thread zur Aktualisierung der Statusausgabe wird wie der Transmit-Thread schon zu Beginn gestartet und läuft durchgehend. Die grafische Ausgabe der Statusmeldungen und aufgetretenen Fehler erfolgt im Log-Viewer[49], der aus GTK-Komponenten für den X-Server unter Linux besteht. Hauptsächlich besteht die Aufgabe des Threads darin, Ereignisse, die auf die grafischen Elemente wirken, zu behandeln und dementsprechende Aktionen zu starten. Diese Events bestehen aus Aktivitäten, die erstens intern von der Gateway-Software selbst initiiert werden, wie die Aktualisierung der Pufferfüllstandsanzeige, und zweitens durch Benutzereingriffe verursacht werden.

[48] nähere Informationen zu POSIX (Portable Operating System Interface for UniX) findet man in [1], ein Tutorial zu den POSIX Threads wird in [30] zur Verfügung gestellt

[49] siehe Kapitel 4.6.3 auf Seite 90

Beim Zugriff auf den Audiopuffer muss präzise darauf geachtet werden, dass Dateninkonsistenzen aufgrund von „Race Conditions" vermieden werden. Das bedeutet, dass mehr als ein Schreibvorgang oder ein Lesevorgang zur gleichen Zeit nicht durchgeführt werden darf. Alle Zugriffe auf den gemeinsamen Speicher müssen atomar ablaufen, das heißt durch keinen anderen Vorgang gestört werden. Zu beachten ist auch die Tatsache, dass Lesezugriffe auf Ringpuffer aufgrund des konsumierenden Charakters den Speicher ebenfalls modifizieren, sodass im Prinzip bei dieser Art von Audiopuffer nur Schreibvorgänge existieren. Eine Ausnahme bildet allerdings das oft benötigte Auslesen des Pufferfüllstands für Statusanzeigen und Synchronisations- und Überwachungszwecke, das rein lesend fungiert und keine Änderungen vornimmt.

Um diese Bedingungen zu gewährleisten, werden spezielle Variablen aus der Bibliothek der POSIX Threads verwendet, mit denen der Puffer für eine bestimmte Zeit gesperrt werden kann. Während dieser Zeitspanne - auch kritischer Bereich genannt - erledigt der sperrende Thread seine Tätigkeiten und gibt den Zugriff danach wieder frei. Will nun ein anderer Thread den Audiopuffer während einer Sperre okkupieren, wird er in einen Wartezustand versetzt, bis der erste Thread die Freigabe durchführt.

4.4.5 Synchronisationskalkulationen

Die Synchronisation des Audiopuffers basiert im Prinzip nur auf zwei Kriterien, die in die Berechnungen mit einbezogen werden: Die nominelle Samplingrate und der Pufferfüllstand, der immer in Prozent der gesamten Puffergröße gemessen wird. Es wird immer versucht, den Füllstand genau im Mittel zu halten, sodass genau fünfzig Prozent belegt sind. Weicht er durch externe Einflüsse[50] von diesem Gleichgewicht ab, dann wird die Samplingrate, mit der die Audiodaten auf den IEEE 1394-Bus geschickt werden, geringfügig modifiziert, sodass durch schnelleres oder langsameres Senden der Samples der Puffer wieder auf Normalniveau gebracht wird. Dabei muss immer darauf geachtet werden, keine zu radikalen Modifikationen vorzunehmen, die das menschliche Gehör wahrnehmen könnte.

In bestimmten Intervallen, die durch eine Anzahl isochroner Pakete in der Konfigurationsdatei definiert werden, wird der Pufferfüllstand überprüft und basierend auf einem Mittelwert der vergangenen Füllstände ein Faktor errechnet, mit dem die nominelle Samplingrate multipliziert und somit geringfügig verändert wird. Die detaillierte Beschreibung dieser Berechnung erfolgt in 4.4.5.1 und 4.4.5.2.

Die Resultate der Synchronisation wirken sich dann im SYT-Feld des isochronen Pakets aus. Da sich der Zeitstempel in diesem Feld aus den unteren 16 Bit des Cycle Time Registers[51] zusammensetzt und sich auf das Event im Paket bezieht, das ein Vielfaches des SYT-Intervalls darstellt[52], wird der Wert auf folgende Art und Weise gebildet:

[50] siehe Kapitel 3.7 auf Seite 53
[51] siehe Kapitel 2.4.3 auf Seite 22
[52] siehe Kapitel 2.4.5.1 auf Seite 28

Die Differenz zwischen zwei Timestamps ist bei gleichbleibender Samplingrate immer konstant und kann leicht aus Samplingfrequenz und SYT-Intervall berechnet werden. Sie wird in den kleinsten Zeiteinheiten im IEEE 1394-Standard angegeben, die ein Taktgeber mit 24,576 MHz produziert und die auch die Basis der Cycle Time darstellen.

$$TicksPerSytIntervall = 24576000 * \frac{SytIntervall}{SamplingRate}$$

$$TicksPerSytIntervall \quad ... \quad \text{Zeiteinheiten pro SYT} - \text{Intervall}$$

Um nun den aktuellen SYT-Wert zu bilden, wird zum letzten Wert diese Differenz addiert und an das Format des Cycle Time Registers angepasst. Da sich in diesem Projekt die Samplingrate aus Synchronisationszwecken aber ändern kann, muss die Differenz nach jeder Modifikation neu kalkuliert werden. Um bei den Berechnungen möglichst präzise zu bleiben und weniger Rundungsfehler aufkommen zu lassen, wird in der Implementierung im Großteil der Fälle mit echten Brüchen anstatt mit Fließkommazahlen gearbeitet.

4.4.5.1 Berechnung des Füllstandsmittelwerts Damit die Variation der Samplingrate als Resultat der Synchronisation nicht zu sprunghaft vonstatten geht, wird als Basis der Berechnungen nicht der aktuelle Pufferfüllstand verwendet, sondern ein Mittelwert aus den vergangenen Messwerten gebildet, der rapide Füllstandsänderungen kompensiert und somit eine glattere Kurve im Samplingfrequenz-Zeit-Diagramm bewirkt. Da der MPlayer zum Beispiel die Samples dem Audiopuffer nur in Blöcken mit einer konstanten konfigurierbaren Größe übergibt, kann es kurzzeitig zu starken Schwankungen im Füllstand kommen, die sich zwar von selbst wieder legen, aber zu ungewollten Synchronisationsergebnissen führen können, falls im falschen Augenblick nahe einem Extremwert eine Messung durchgeführt wird.

Mit folgender Formel wird der aktuelle Durchschnittswert berechnet, der abhängig vom Mittelwert am letzten Messzeitpunkt und dem Prozentwert ist, der den Einfluss des alten Werts auf den neuen festlegt:

$$\begin{aligned} AvgFillstate(t) = {} & AvgFillstate(t-1) * OldAverageFactor + \\ & CurrentFillstate(t) * (1 - OldAverageFactor) \end{aligned}$$

$$\begin{aligned} AvgFillstate(t) \quad & ... \quad \text{Füllstandsmittelwert zum Zeitpunkt t} \\ OldAverageFactor \quad & ... \quad \text{Gewichtung des Füllstandsmittelwerts zum Zeitpunkt t} - 1 \\ CurrentFillstate(t) \quad & ... \quad \text{genauer Füllstand zum Zeitpunkt t} \end{aligned}$$

Der wählbare Faktor wird vor dem Start der Gateway-Software in der Konfigurationsdatei[53] festgelegt. Bei hohen Werten geht der aktuelle Messwert immer mehr unter und abrupte Änderungen des Füllstands, die dann zu einem konstanten neuen Pegel führen, wirken sich zeitlich gesehen erst später auf die Resultate der Synchronisationsberechnungen aus. Andererseits dürfte die Synchronisation aufgrund ihrer Spezialisierung auf die Kompensation geringer Füllstandsabweichungen in solchen Extremfällen sowieso wenig erfolgversprechend sein. Wegen der hohen Empfindlichkeit des menschlichen Gehörs darf die Samplingrate auch nur leicht korrigiert werden, sodass auch bei unerwarteten ruckartigen Änderungen keine zu drastischen Maßnahmen ergriffen werden dürfen. Diese maximalen Toleranzgrenzen, die im nächsten Kapitel in die Berechnungen miteinfließen, können ebenfalls in der Konfigurationsdatei definiert werden.

4.4.5.2 Berechnung des Änderungsfaktors Der Multiplikator, der die Samplingrate variiert, ist linear proportional zum durchschnittlichen Pufferfüllstand und zur prozentuellen Toleranzgrenze. Diese Tatsachen sind in folgender Formel auch klar ersichtlich:

$$Factor = 1 - [(1 - 2 * AvgFillstate) * SyncTolerance]$$

$Factor$ ... Multiplikator für die Samplingrate
$AvgFillstate$ ... Füllstandsmittelwert
$SyncTolerance$ ... maximale Abweichung in Prozent

Im Diagramm 13 sieht man den linearen Zusammenhang zwischen dem Pufferfüllstand und dem daraus berechneten Multiplikator für die Modifikation der Samplingrate. Für die drei verschiedenen Graphen wurden vernünftige Beispiele für maximale Toleranzgrenzen bei der Synchronisation gewählt und mit unterschiedlichen Farben gekennzeichnet. Bei einem Füllstand von fünfzig Prozent, der als Optimum gilt und das ständig im Auge behaltene Ziel der Synchronisation darstellt, wird die Samplingfrequenz unabhängig vom Toleranzparameter nicht verändert, weshalb sich auch alle Linien in diesem Punkt treffen. Die unterbrochene Horizontale zeigt sozusagen die nominelle Samplingrate, von der desto mehr abgewichen wird, je größer der Abstand zum normalen halben Füllstand ist. In den Extremfällen - dem leeren und dem vollen Puffer - werden die maximalen Toleranzen erreicht, dazwischen wird linear gemittelt.

Die lineare Änderungskurve stellt nur eine mögliche Lösung für die Multiplikatorberechnung dar, man könnte dem Anwender zusätzlich zur Definition der Steilheit auch die Freiheit geben, zwischen verschiedenen Kurventypen zu wählen, wie zum Beispiel einer Exponentialfunktion oder einer Funktion höheren Grades. Damit würde erreicht werden, dass die Synchronisation bei geringen Abweichungen vom Optimalpunkt fast keine Änderungen an der Samplingrate vornimmt,

[53] siehe Kapitel 4.5 auf Seite 83

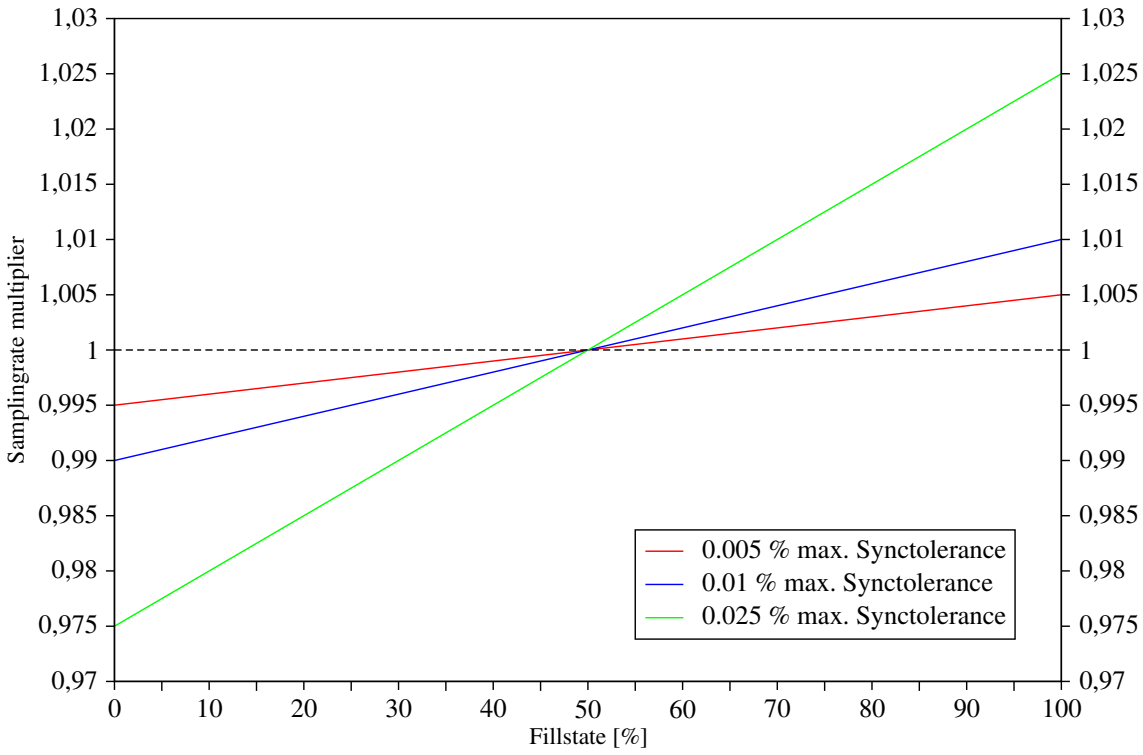

Abbildung 13: Multiplikator-Füllstand-Diagramm

die sich jedoch rasch vergrößern können, wenn sich der Füllstand den beiden Extremsituationen nähert.

4.4.6 Synchronisationsverlauf

In diesem Unterkapitel wird anhand eines Beispiels der Verlauf einer Anpassung an eine asynchron laufende Quelle demonstriert. Die Abbildung 14 und die restliche Diskussion dieser Synchronisation basieren jedoch auf theoretischen Überlegungen und Berechnungen und nicht auf praktischen Beobachtungen während der Laufzeit der Gateway-Software. Deshalb müssen vorab einige Voraussetzungen definiert und Parameter fixiert werden, die diese fiktive Umgebung charakterisieren:

- Die interne Uhr der sendenden Audioquelle läuft mit einer konstanten Asynchronität schneller oder langsamer als die Uhr des Gateway-Rechners. Das bedeutet, dass pro Sekunde um einen gleichbleibenden Faktor mehr oder weniger Samples beim Gateway ankommen. Da in der Realität keine zwei Uhren existieren, die absolut synchron laufen, ist diese vereinfachte Annahme nicht weit von der Wirklichkeit entfernt.
- Auf dem Weg von der Quelle zum Gateway treten keine Unregelmäßigkeiten oder sonstige Störungen in der Übertragung der Audiodaten auf. Diese Annahme muss getroffen werden, um die Komplexität der Synchronisationsberechnungen zu verringern.

- Der MPlayer-Teil der Software dekodiert die Audiodaten in Echtzeit und leitet die Samples sofort einzeln an den Puffer weiter, anstatt zu warten, bis eine bestimmte Datenblockgröße erreicht ist. Aus diesem Grund und wegen der zweiten Voraussetzung verliert auch die Berechnung eines Füllstandsmittelwerts an Bedeutung, weshalb nur der aktuelle Wert berücksichtigt wird, das heißt der Parameter *OldAverageFactor* wird auf Null gesetzt.

Trotz dieser nicht ganz praxistauglichen Annahmen kann man in den nachfolgenden allgemeinen Berechnungen und dem abschließenden Beispiel deutlich die Arbeitsweise der Gateway-Software erkennen, obwohl die Vorgänge durch die perfekt arbeitende Quelle und den störungsfreien Kommunikationsfluss stark vereinfacht wurden.

Mit der folgenden Gleichung wird die absolute und relative Füllstandsabweichung vom Normalwert, der der Hälfte des Gesamtpuffers entspricht, ermittelt.

$$
\begin{aligned}
absFillstateDeviation(t) &= [nsr * (1+p) * ch * ss * t] - [nsr * k(t) * ch * ss * t] \\
&= nsr * ch * ss * t * [1 + p - k(t)] \\
relFillstateDeviation(t) &= \frac{absFillstateDeviation(t)}{Buffersize} \\
&= \frac{nsr * ch * ss * t * [1 + p - k(t)]}{nsr * ch * ss * sec} \\
&= \frac{t}{sec}[1 + p - k(t)]
\end{aligned}
$$

$absFillstateDeviation(t)$	...	absolute Füllstandsabweichung in Bytes zum Zeitpunkt t
$relFillstateDeviation(t)$	...	relative Füllstandsabweichung zum Zeitpunkt t
nsr	...	nominale Samplingrate
p	...	relative Sendegeschwindigkeitsabweichung der Quelle
ch	...	Anzahl der Audiokanäle
ss	...	Größe eines Samples in Bytes
$k(t)$	...	Änderungsfaktor der Synchronisationsberechnungen
sec	...	Audiopuffergröße in Sekunden

Ausgehend davon und mit Hilfe der Formel für die Berechnung des Änderungsfaktors aus dem Kapitel 4.4.5.2 kann der Füllstand mit folgender Gleichung für jeden Zeitpunkt t ermittelt werden:

$$
\begin{aligned}
relFillstate(t) &= 0.5 + relFillstateDeviation(t) \\
relFillstate(t) &= 0.5 + \frac{t}{sec}\{1 + p - \{1 - [1 - 2 * relFillstate(t)] * st\}\}
\end{aligned}
$$

$$
\begin{aligned}
relFillstate(t) &= 0.5 + \frac{t}{sec}\{p + [1 - 2 * relFillstate(t)] * st\} \\
relFillstate(t) &= 0.5 + \frac{t * (p + st)}{sec} - \frac{t * 2 * st}{sec} relFillstate(t) \\
relFillstate(t) * (1 + \frac{t * 2 * st}{sec}) &= 0.5 + \frac{t * (p + st)}{sec} \\
relFillstate(t) &= \frac{0.5 + \frac{t*(p+st)}{sec}}{1 + \frac{t*2*st}{sec}} \\
relFillstate(t) &= \frac{0.5 * sec + (p + st) * t}{sec + 2 * st * t}
\end{aligned}
$$

$relFillstate(t)$ ... relativer Füllstand zum Zeitpunkt t

st ... maximale relative Abweichung von der nominellen Samplingrate

Diese Formeln setzen jedoch voraus, dass sowohl der Empfang der Audiodaten von der Quelle als auch das Senden der Samples an den IEEE 1394-Bus gleichzeitig aktiv sind. Zu Beginn muss aber der Audiopuffer zuerst bis zur Hälfte gefüllt werden, bevor das Weiterleiten der Daten gestartet wird. Diese Zeitspanne errechnet sich wie folgt:

$$
\begin{aligned}
0.5 &= \frac{nsr * ch * ss * TimeToFill * (1 + p)}{nsr * ch * ss * sec} \\
&= \frac{TimeToFill * (1 + p)}{sec} \Rightarrow \\
TimeToFill &= 0.5 * \frac{sec}{1 + p}
\end{aligned}
$$

$TimeToFill$ ... Zeitspanne zum Auffüllen des Puffers auf 50%

Nach diesem Intervall beginnt die Zeit für die obigen Gleichungen zu laufen, wie man in der Abbildung 14 demonstrativ sehen kann. Mehr oder weniger zufällig wurden folgende Parameter für die Erstellung der Grafik festgelegt:

- Die maximale Abweichung von der nominellen Samplingfrequenz beträgt 2,5%.
- Die Sendegeschwindigkeit der Quelle ist um 0,7% gegenüber der nominellen Samplingfrequenz des Gateways erhöht.
- Die Länge des Audiopuffers beträgt 5 Sekunden.

In der Abbildung sind zwei Kurven in den Farben Rot und Blau dargestellt. Zu Beginn verlaufen beide Linien deckungsgleich, jedoch nur bis der Puffer zu 50% gefüllt ist, was ungefähr 2,5 Sekunden dauert. Diese sehr steil ansteigende Gerade von 0% bis 50% ist aufgrund des groß gewählten

Maßstabs der Zeitachse nicht sichtbar. Nach dieser gemeinsamen Startphase wird nur ein Parameter variiert, und zwar die minimale Füllstandsdifferenz zur letzten Synchronisationsaktion[54], die gegeben sein muss, um die Samplingrate auf Seiten des IEEE 1394-Busses erneut modifizieren zu dürfen.

Bei der roten Variante wurde diese Differenz auf 0% gesetzt, das heißt, die Samplingfrequenz wird bei jeder Überprüfung des Pufferfüllstands, die standardmäßig in Intervallen von 1000 isochronen Pakete stattfindet, angepasst. Dadurch ergibt sich eine über den gesamten Zeitraum streng monoton steigende Kurve, die sich asymptotisch an den optimalen Füllstand annähert, der in diesem Beispiel genau 64% beträgt und allgemein durch folgende Grenzwertberechnung ermittelt werden kann:

$$\lim_{t \longrightarrow \infty} \frac{0.5 * sec + (p + st) * t}{sec + 2 * st * t} = \frac{p + st}{2 * st}$$

Das bedeutet, dass sich an diesem Punkt der Füllstand nicht mehr ändert, da sich die Sendegeschwindigkeit der Audioquelle und die des Gateways bei der Weiterleitung genau die Waage halten. Dieser ideale Wert kann aber mit Hilfe des hier verwendeten Synchronisationsalgorithmus nie erreicht werden.

In der blauen Kurve kommt eine minimale Füllstandsdifferenz von 5% zur Anwendung, sodass aufgrund des Startpunkts bei 50% nur in der Nähe der Punkte 55%, 60% und 65% eine Modifikation durchgeführt wird. Dadurch pendelt der Füllstand die ganze restliche Zeit zwischen 60% und 65%, wobei die streng monoton fallenden Abschnitte wesentlich länger dauern als die Steigungen. Der Grund dafür besteht darin, dass die obere Grenze näher am optimalen Wert liegt und die hier stattfindende Änderung der Samplingfrequenz besser an die Sendegeschwindigkeit der Quelle angepasst ist, sodass die Füllstandsabweichungen geringer ausfallen.

Der einzige aber in der Praxis gravierende Vorteil der blauen Variante, in der die Wahl auf eine höhere Minimaldifferenz fällt, ist die Entlastung der Ressourcen des Gateway-Rechners, der durch die Synchronisationskalkulationen abhängig von seiner Ausstattung doch sehr beansprucht wird.

4.5 Konfigurationsdatei

Zur Feineinstellung der verschiedenen Parameter des Gateways wird eine Konfigurationsdatei verwendet, die aufgrund des standardisierten ASCII-Formats mit jedem beliebigen Texteditor modifiziert werden kann. Pfad und Name der Datei sind im Quellcode des Gateways fixiert, weshalb die Verwendung eines alternativen Standorts nur durch Änderung in der Datei */<mplayer-dir>/libao2/ao_pcm.c* möglich ist. In */etc/lib1394/config* findet man standardmäßig die Konfiguration. Existiert keine Konfigurationsdatei respektive fehlen in der vorhandenen Datei einige Pa-

[54] Parameter BUF_FILL_DIFFERENCE in der Konfigurationsdatei

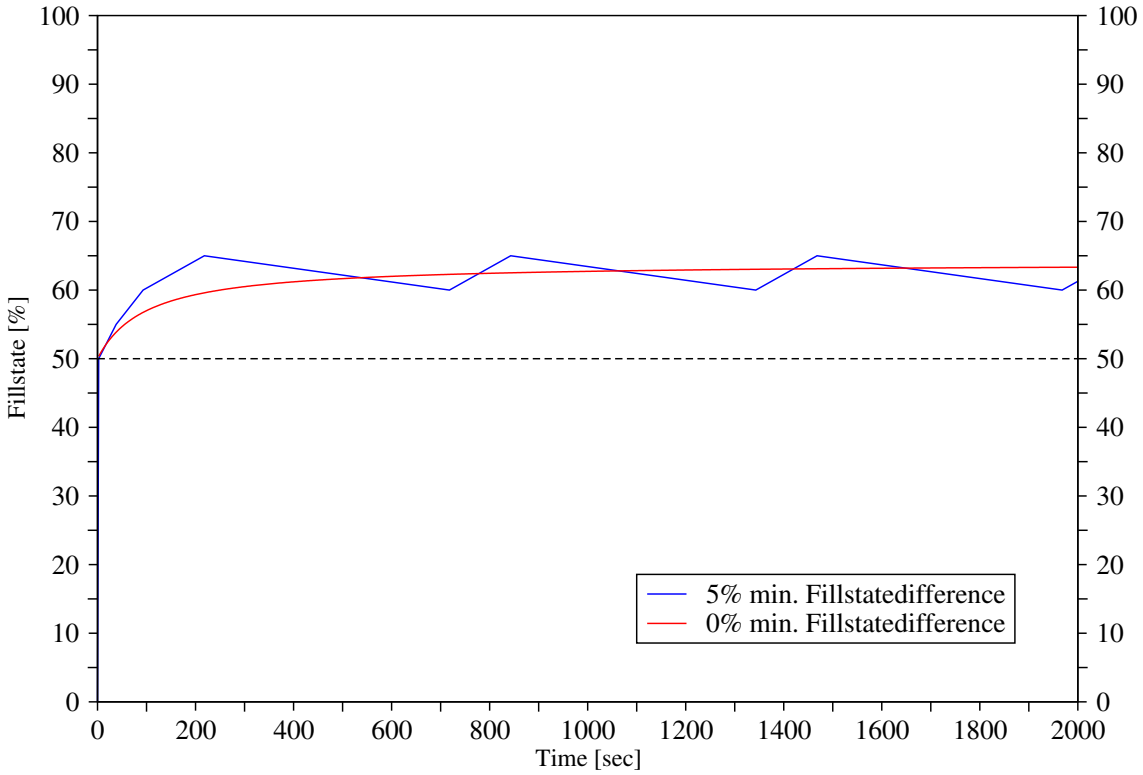

Abbildung 14: Synchronisationsverlauf bei 0.7% schnellerer Quelle

rameter, dann werden die bei der detaillierten Beschreibung in Klammer stehenden Standardwerte eingesetzt, die für die Mehrzahl der Einsatzgebiete vernünftig und geeignet sein sollten.

Folgende Parameter, die auch in der oben genannten Datei selbst noch kurz beschrieben werden und in vier Bereiche gegliedert werden können, beeinflussen das Verhalten der Gateway-Software und können in gewissen Grenzen variiert werden:

4.5.1 Parameter für die Kommunikation mit dem MPlayer

- **BURSTSIZE** (16384): Dieser Wert, dessen Einheit Bytes ist, gibt die Größe der Datenblocks an, die der MPlayer dem ao_pcm-Modul in einem Stück übergibt. Somit wartet der MPlayer so lange mit der Weiterleitung der Audiosamples, bis er die hier angegebene Datenmenge von der externen Quelle bezogen hat. Je kleiner dieser Wert gewählt wird, desto schneller erreichen zwar neue Daten den Audiopuffer des IEEE 1394-Teils, aber desto mehr Overhead wird durch die MPlayer-eigenen Routinen erzeugt. Wählt man den Wert zu hoch, dann läuft der Puffer aus, bevor neue Daten nachkommen. Der Standardwert von 16384 Bytes erwies sich während der Testphasen als geeignet.

4.5.2 Parameter des Audiopuffers

- **BUFFERSIZE** (5): Wie groß der Audiopuffer werden soll, entscheidet dieser Wert in der Einheit Sekunden. Das bedeutet, dass sich bei vollständiger Füllung des Puffers genau soviele Audiodaten in ihm befinden, um eine Wiedergabezeit von der hier festgelegten Länge zu erreichen. Um dies möglich zu machen, muss die tatsächliche Größe in Bytes aber immer beim Start der Gateway-Software neu berechnet werden, abhängig von den Eigenschaften der genutzten Quelle. In die Kalkulation fließen die Samplingrate, die Anzahl der Kanäle und die Größe eines Samples wie in folgender einfacher Formel mit ein:

$$Buffersize \quad = \quad Samplingrate * Channels * Samplesize * Seconds$$

  Mit dem Standardwert von 5 Sekunden und den üblichen Audioeigenschaften von 44,1 kHz, 16 Bit große Samples und Stereoton ergibt sich somit eine Größe von 882000 Bytes oder umgerechnet circa 0,84 MiByte für den Puffer.

- **BUF_START_FILL** (50): Nach dem Start des Gateways wird der Audiopuffer solange mit Daten befüllt, bis der hier festgelegte Prozentwert in Relation zur Gesamtgröße erreicht ist. Erst dann beginnt die Software mit dem Weiterleiten der Samples an die IEEE 1394-Schnittstelle. Der Standardwert von fünfzig Prozent ist deswegen sinnvoll, weil ja während der gesamten Laufzeit versucht wird, diese Mitte mit Hilfe der Puffer-Synchronisation zu

halten. Würde der Startwert sehr stark von diesen fünfzig Prozent abweichen, dann würde schon anfangs die Synchronisation einsetzen und sich langsam zum Idealwert vorarbeiten. Der Hauptgrund für den Einsatz dieses Parameters stellt die Simulation von Extremzuständen wie den Pufferleerlauf und der darauf einsetzenden Gegenmaßnahmen dar. Für den Normalgebrauch sollte der Standardwert unverändert bleiben.

4.5.3 Parameter des IEEE 1394-spezifischen Teils

- **IEEE1394_PORT** (0): Dieser Parameter legt fest, mit welchem IEEE 1394-Adapter gearbeitet werden soll, wobei eine Änderung auf einen höheren Wert natürlich das Vorhandensein von mehreren Schnittstellenkarten voraussetzt. Der Standardwert 0 bezeichnet immer den vom System als ersten registrierten Adapter.

- **ISOCHANNEL** (27): Hier wird der isochrone Kanal, auf dem die Audiodaten gesendet werden sollen, gewählt. Wie schon in 2.4.3.2 erwähnt, stehen 64 Kanäle zur Verfügung, und zwar bei 0 beginnend. Der hier voreingetragene Standardwert von 27 wurde rein zufällig ohne weitere Hintergedanken festgelegt.

- **RCV_NODE** (63): Wird hier ein Wert von 0 bis 62 gewählt, dann versucht die LIBIEC61883 mit dem Setzen der Plug Control Register [15] eine Point-to-Point-Verbindung mit dem Node der hier angegebenen ID herzustellen. Damit weiß nur dieser Knoten, auf welchem isochronen Kanal die Audiodaten gesendet werden. Die standardmäßige Broadcastadresse 63 ermöglicht allen Geräten am lokalen IEEE 1394-Bus ohne vorherige Aushandlungen den Zugang zu den Daten.

- **AMDTP_FORMAT** (1): Dieser Parameter legt fest, in welchem Format die Audiodaten innerhalb eines isochronen Pakets aufgebaut sind. Zwei Möglichkeiten stehen dazu zur Auswahl, einerseits der AM824-RAW Modus, der in Kraft tritt, wenn AMDTP_FORMAT auf 0 gesetzt wird, und der beliebig viele Audiokanäle erlaubt, und andererseits der IEC 60958 konforme Modus, der dem Wert 1 entspricht, aber nur maximal zwei Kanäle erlaubt. Details zu den beiden Modi wurden in 2.4.5.3 diskutiert.

- **TRANSFER_DELAY** (9000): Die Transferverzögerung bezeichnet das Zeitintervall zwischen dem Eintreffen eines Events beim Transmitter und dem Zeitpunkt, in dem es der Empfänger erhält. Dieser Wert kann natürlich nicht genau bestimmt werden, weil er von zu vielen Faktoren abhängig ist, aber ein grober Schätzwert, der sicherlich über der tatsächlichen Übertragungszeit liegt, sind 9000 Ticks des Taktgebers eines IEEE 1394-Chips, also fast drei Cycles. Um sich mehr darunter vorstellen zu können, wären das in der uns vertrauten Zeiteinheit ungefähr 366 Mikrosekunden. Diese Zeitspanne wird zum berechneten SYT-Wert hinzugefügt, um den Präsentationszeitpunkt der Events in die Zukunft zu verlegen. Damit bleibt dem Endgerät noch genug Spielraum, um die rechtzeitige Ausgabe der

Audiosamples zu gewährleisten. Da der maximale Wert im SYT-Feld dezimal 49151 beträgt, darf TRANSFER_DELAY diesen nicht überschreiten, da es sonst bei der Addition zu einem Überlauf kommen würde. Um Missverständnisse in der Interpretation beim Endgerät zu vermeiden, sollte der Parameter vielmehr weit unterhalb der Hälfte des Maximalwerts liegen.

4.5.4 Parameter zur Steuerung der Synchronisation

- **SYNC_ENABLE** (1): Mit diesem einfachen Schalter kann die Synchronisation aktiviert (1) oder deaktiviert (0) werden. Bei Deaktivierung werden keine Modifikationen der Samplingrate vorgenommen und somit kann es schon bei geringfügig asynchron laufenden Uhren oder sonstigen kleinen Unregelmäßigkeiten vorkommen, dass der Audiopuffer nach einiger Zeit entweder leer- oder überläuft.

- **SYNC_CHECK_INTERVAL** (1000): Diese Anzahl an isochronen Paketen gibt das Intervall an, wie oft der Pufferfüllstand überprüft wird, und legt damit fest, in welchen Abständen die Samplingrate geändert werden kann. Je kleiner dieser Wert ausfällt, desto öfters wird der Füllstand kontrolliert und desto mehr Rechenleistung wird verbraucht. Bei sehr hohen Werten kann die Synchronisierung eventuell nicht schnell genug auf abrupte Änderungen reagieren.

- **SYNC_TOLERANCE** (250): Der Wert definiert die maximal zulässige Abweichung von der nominellen Samplingrate in Hundertstel Prozent. Wird der Prozentsatz zu gering gewählt, kann die Synchronisierung nicht richtig greifen und damit den Pufferfüllstand nicht schnell genug wieder normalisieren. Zu hohe Werte verändern die Wiedergabegeschwindigkeit so stark, dass das menschliche Gehör den Unterschied merken kann. Der Standardwert von 250 - also 2,5% Toleranz von der nominellen Samplingfrequenz - scheint ein vernünftiges Mittel zu sein.

- **BUF_FULL_SLEEPTIME** (1000000): Falls die Daten zu schnell vom MPlayer in den Puffer gelangen und er voll laufen sollte, dann wird die Anlieferung vom MPlayer genau für diese Zeit in Mikrosekunden blockiert. In dieser Wartezeit wird die Weiterleitung ins IEEE 1394-Netzwerk natürlich fortgesetzt, wodurch der Füllstand automatisch wieder verringert wird.

- **BUF_FILL_DIFFERENCE** (5): Dieser Prozentsatz legt die minimale Differenz zwischen dem Pufferfüllstand bei der letzten Synchronisierung und dem aktuellen fest, der gegeben sein muss, um erneut die Samplingrate verändern zu dürfen. Angenommen, dieser Wert beträgt 5 und der Füllstand bei der letzten Modifikation stand bei 63%, dann wird die Samplingfrequenz erst wieder geändert, wenn der Füllstand kleiner 58% oder größer 68% beträgt. Damit wird verhindert, dass nach jedem SYNC_CHECK_INTERVAL die Rate korrigiert wird, und somit Rechenleistung eingespart.

- **OLD_AVG_FACTOR** (95): Da die Synchronisierungsberechnungen nicht rein auf dem aktuellen Pufferfüllstand aufbauen, sondern auf einem gemittelten Wert über die vergangenen Füllstände[55], muss festgelegt werden, wie stark die alten Werte in die momentane Kalkulation miteinfließen. Dieser Faktor in Prozent definiert somit, wieviel Gewicht der zuletzt berechnete Mittelwert im neuen Durchschnitt hat. Je höher der Wert gewählt wird, desto weniger Einfluss auf die Synchronisation haben Ausreißer, die den Füllstand für kurze Zeit stark verringern oder erhöhen, dann aber wieder Normalniveau erreichen. Bei rapiden Füllstandsänderungen ohne Rückkehr zum Normalzustand kann sich aber genau diese Tatsache nachteilig auswirken, denn in diesem Fall merkt die Synchronisation diese Situation durch die starke Mittelung erst sehr spät. Trotz dieses Nachteils schien ein Standardwert von 95% in den meisten Situationen angemessen.

4.6 Start und Steuerung

Dieses abschließende Unterkapitel beschreibt den Vorgang zum Starten der Gateway-Software und die möglichen Benutzereingriffe während der Laufzeit. Zur Überwachung der Ereignisse wurde ein Log-Viewer entwickelt, der in einer grafischen Umgebung läuft und den aktuellen Status präsentiert.

4.6.1 Aufruf der Gateway-Software

Da die Ausgabe auf den IEEE 1394-Bus nur als Modul für den MPlayer entwickelt wurde, wird die Software abgesehen von einigen speziellen Parametern auf die gleiche Art und Weise aufgerufen, wie man es standardmäßig vom MPlayer gewohnt ist. Durch die üblichen zusätzlichen Optionen, die beim Aufruf des Kommandos angegeben werden können, wird jedoch nur der Teil des MPlayers beeinflusst, der die Dekodierung der Audiodaten von der externen Quelle übernimmt. Alle Vorgänge, die nach der Übergabe der Samples an den neu implementierten Audiopuffer passieren, können nur durch die schon beschriebene Konfigurationsdatei angepasst werden.

Die vollständige Dokumentation aller möglichen Parameter kann in den Manpages eines Systems, in dem der MPlayer ordnungsgemäß systemweit installiert wurde, mit Hilfe des Befehls „man mplayer” abgerufen werden. An dieser Stelle werden nur zwei wichtige Optionen demonstriert.

Im Algorithmus 8 findet man außer der ausführbaren Datei und der RTSP-Audioquelle noch zwei zusätzliche Optionen:

- **„-cache <x>”:** Dieser Parameter definiert die Größe des Puffers in KB für die originalen MPlayer-Komponenten, das heißt dieser Speicher entspricht nicht dem neu hinzugefügten Audiopuffer, von dem in dieser Arbeit oftmals die Rede ist. Der MPlayer verwaltet einen

[55] siehe Kapitel 4.4.5 auf Seite 77

Algorithm 8 Beispielhafter Aufruf der Gateway-Software

```
/<MPlayer-Hauptverzeichnis>/mplayer -cache 64 -ao pcm:nowaveheader
    rtsp://realorf.conova.com/salzburg.ra
```

ähnlichen Puffer, den er vor der Wiedergabe der Audiodaten befüllt, um unregelmäßige Paketempfangszeiten von der Datenquelle zu kompensieren.

- **„-ao pcm:nowaveheader":** Durch diese Option wird das Ausgabemodul aktiviert, das ursprünglich für das Schreiben in eine Datei vorgesehen war und in diesem Projekt durch den IEEE 1394-Teil ersetzt wurde. Der Zusatz „nowaveheader" bedeutet, dass es nicht notwendig ist, einen standardkonformen Header für Dateien im Wave-Format am Beginn des Ausgabevorgangs zu bilden, da ja eigentlich keine richtige Datei erstellt wird.

4.6.2 Steuerung während der Laufzeit

Um das Verhalten des Gateways während der Laufzeit zu kontrollieren respektive zu ändern, existieren grundsätzlich zwei Möglichkeiten, erstens mit Hilfe von Befehlen vom Endgerät aus über den IEEE 1394-Bus und zweitens durch Benutzereingaben direkt am Gateway über die Tastatur.

Grundsätzlich sollte der Gateway-Rechner unangetastet bleiben und ohne Unterbrechung die Audiodaten konvertieren und ins FireWire-Netzwerk weiterleiten. Der einzige Benutzereingriff, der gerechtfertigt wäre, ist die Wahl der Audioquelle.

1. **Steuerung vom Endgerät:**

   Speziell für die Steuerung von IEEE 1394-Geräten wurden Standards entwickelt, die auf einem asynchron übertragenen Request-Response-Protokoll basieren. Die Kontrollbefehle werden dabei im Function Control Protocol (FCP), das im Standard IEC 61883-1 definiert wird, transportiert. Die beiden wichtigsten dieser Standards sind Audio/Video Control (AV/C) [19] von der 1394 Trade Association und Home Audio Video Interoperability (HAVi) [20], der aus einem Zusammenschluss mehrerer großer Konzerne der Consumer-Electronics-Branche hervorgegangen ist.

   Die derzeitige Implementierung unterstützt noch keine der beiden Standards, jedoch ist geplant, Teile des AV/C-Standards zu integrieren, um innerhalb einer Playlist zu navigieren oder sonstige Audioeigenschaften zu variieren.

2. **Steuerung über Tastatur:**

   Eingaben über die üblichen Kanäle, wie zum Beispiel Tastatur, Maus oder Fernwartungssitzungen, werden momentan ausschließlich vom MPlayer behandelt und betreffen deshalb auch nur dessen Funktionen. Das bedeutet, dass innerhalb von Playlists[56] zwischen

[56] zum Beispiel einfache Textdateien, die pro Zeile eine Audioquelle enthalten

den Quellen gewechselt oder innerhalb einer Quelle vor- und zurückgespult werden kann, falls dieses Feature vom gerade genutzten Media-Server unterstützt wird. Weiters kann die Übertragung angehalten und später wieder fortgesetzt werden, was aber während einer Streaming-Sitzung eine nicht unbedingt sinnvolle Aktion auf dem Gateway darstellt. Der Einsatz solcher Aktivitäten wäre direkt bei den Endgeräten weit vernünftiger, da dann jeder selbst entscheiden kann, wie er mit den empfangenen Audiodaten umgeht, und somit keine globalen den gesamten IEEE 1394-Bus betreffenden Konsequenzen entstehen.

4.6.3 Log-Viewer

Für die Überwachung der Geschehnisse während der Laufzeit der Gateway-Software wurde ein Log-Viewer entwickelt, der mit grafischen Komponenten auf Basis des X-Servers unter Linux den aktuellen Status sowie vergangene Meldungen, Warnungen und Fehler präsentiert. Ein Screenshot in Abbildung 15 zeigt den Pufferfüllstand zu diesem Zeitpunkt, die gesamte Anzahl an bisher gesendeten isochronen Paketen, eine Liste der Initialisierungsparameter, die beim Start von der Konfigurationsdatei eingelesen wurden und im mittleren der drei großen Bereiche die Synchronisationsaktivitäten mit den veränderten Samplingraten.

Notwendig für die Lauffähigkeit des Log-Viewers ist ein aktiver X-Server und die Unterstützung von GTK-Komponenten und GThreads[57]. Falls eine dieser Kriterien nicht erfüllt sein sollte, muss vor dem Kompiliervorgang in folgenden zusätzlichen Konfigurationsdateien, die sich im Verzeichnis /<MPlayer-Hauptverzeichnis>/lib1394/ befinden, der Log-Viewer deaktiviert werden:

- **config.mak:** Die einzige Variable GTKVIEWER in dieser Datei kann entweder mit „yes" oder „no" belegt sein und wird in den Makefiles ausgewertet. So kann bei Nichtvorhandensein der vorausgesetzten Bibliotheken im System mit dem Wert „no" die Integration der GTK- und GThread-Dateien in die ausführbare Datei verhindert werden ohne eine Fehlermeldung auszugeben und damit den Erstellungsvorgang abzubrechen.
- **config.h:** Die Präprozessoranweisung „#define GTKVIEWER 0" würde bewirken, dass bestimmte Teile im Quellcode, die GTK- und GThread-spezifische Funktionen verwenden und auch als solche gekennzeichnet sind, ausgeblendet und somit nicht in den Kompiliervorgang miteinbezogen werden. Der Standardwert „1" bindet diese Code-Komponenten jedoch mit ein und ermöglicht somit die Anzeige des Log-Viewers.

Die Grund für die Existenz von zwei Konfigurationsdateien, die im Prinzip den selben Zweck verfolgen und auch synchron geschaltet werden müssen, ist der, dass sie an verschiedenen Punkten im Programmerstellungsprozess ansetzen und deshalb eine geringfügig unterschiedliche Variablendefinition voraussetzen.

[57] siehe Paragraph 4.1.3.1 auf Seite 63, Kapitel 4.2.2 auf Seite 70 und 4.4.4 auf Seite 76

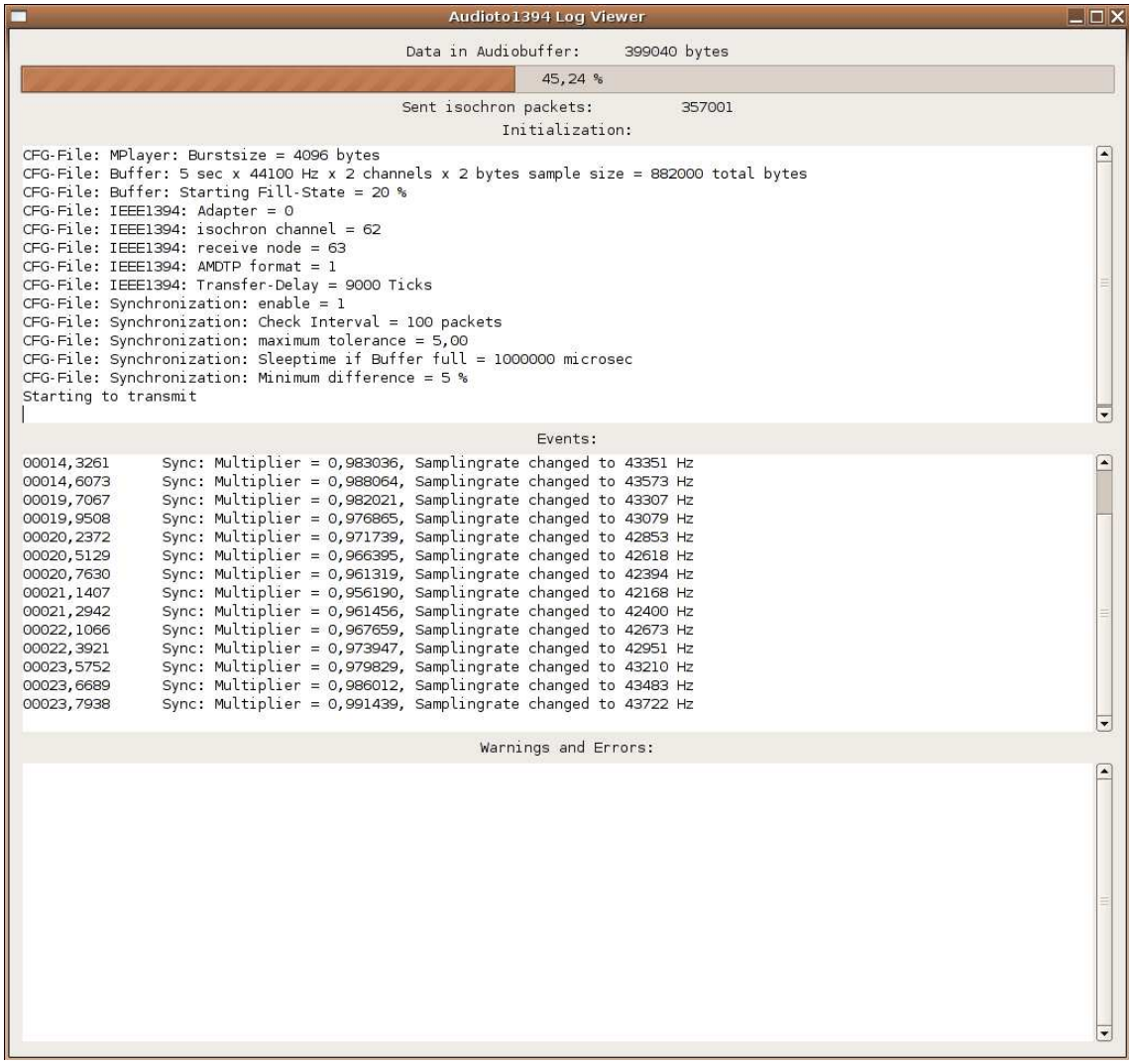

Abbildung 15: Audioto1394 Log Viewer

5 Kritik und Erweiterungen

Die Abhängigkeit von externen Softwareteilen, wie zum Beispiel hier der Quellcode des MPlayers, führt dazu, dass man einige Kompromisse eingehen muss, um den eigenen Code an die vorhandenen Schnittstellen anzupassen. Dadurch konnte noch nicht alles in die aktuelle Implementierung integriert werden, was ursprünglich in der Zielsetzung enthalten war. Die einzige Möglichkeit, die eingegangenen Kompromisse zu umgehen, wäre die Anpassung des MPlayer-Quellcodes an die eigenen Vorstellungen beziehungsweise die Neuentwicklung der zentralen Teile des MPlayers. Abgesehen vom zeitlichen Problem hätte man dann sicherlich die enorme Vielfalt der Formate, die der MPlayer momentan unterstützt und in zukünftigen Versionen unterstützen wird, einschränken und damit Einbußen in der Kompatibilität mit öffentlichen Audioquellen in Kauf nehmen müssen.

5.1 Offene Probleme

Hier werden kurz einige Probleme diskutiert, die in der momentanen Softwareversion noch auftauchen, aber grundsätzlich lösbar sind und in zukünftigen Releases behoben werden können.

5.1.1 Dynamische Neukonfiguration des IEEE 1394-Busses

Sobald sich während der Ausführung des Programms die Node-ID des Gateway-Rechners durch Bus-Resets, die entweder absichtlich durchgeführt oder durch Modifikationen in der Busstruktur verursacht werden, ändert, wird die Callback-Funktion des Transmit-Threads nicht mehr aufgerufen, wodurch in weiterer Folge keine isochronen Pakete mehr gesendet werden und der Audiopuffer überläuft. Lösungsansätze für dieses Problem könnte man auf Basis folgender Idee entwickeln.

Im Programm selbst könnte man in regelmäßigen Abständen beziehungsweise sobald erkannt wird, dass keine Daten mehr an die RAW1394-Schnittstelle weitergeleitet werden können, eine Neukonfiguration der relevanten IEEE 1394-Komponenten durchführen, ohne den Audiopuffer und den MPlayer neu zu initalisieren.

Komfortabler wäre eine auch nach einer strukturellen Änderung des Busses gleichbleibende Node-ID, doch diese kann aufgrund der automatischen Konfigurationsalgorithmen auf einer sehr hardwarenahen Ebene nicht gewährleistet werden.

5.1.2 RealAudio über RTP/UDP

Da RealAudio eines der am weitesten verbreiteten Audioformate im Internet und RTP über UDP das am besten geeignetste Protokoll zur Übertragung von Echtzeitdaten zum Gateway-Rechner darstellt, läge es natürlich nahe, beides innerhalb eines Streams zu unterstützen. Jedoch arbeiten alle dem Autor bekannten Server, die im Internet Daten im RealAudio-Format anbieten, nur mit dem schon in 2.5.2 erwähnten proprietären Protokoll RDT auf UDP-Basis.

Da in diesem Projekt der MPlayer die Formatkonvertierung der Audiodaten übernimmt und diese Komponente mit geringem Aufwand auch durch eine neue Version ersetzt werden kann, würde eine zusätzliche Unterstützung des RDT-Protokolls vom MPlayer dieses Problem am besten lösen. Im Prinzip ähnelt RDT dem standardisierten RTP in vielerlei Hinsicht, sodass es genauso gut für Echtzeitdatenströme geeignet wäre. Der ebenfalls vom Hersteller RealNetworks entwickelte Multimediaplayer RealPlayer verwendet RDT je nach Konfiguration auch mit UDP als Transportprotokoll, sodass man diesen als Anhaltspunkt bei der Protokollanalyse nehmen und so das nicht offen gelegte RDT nachbilden könnte.

5.1.3 Mehr Informationen vom MPlayer

In der vorliegenden Implementierung wird dem MPlayer beim Aufruf die Quelle der Audiodaten übergeben und nachdem dem IEEE 1394-Teil zu Beginn kurz die grundlegenden Audioparameter, wie Samplingrate, Kanalanzahl und Sampleformat, mitgeteilt wurden, werden anschließend nur mehr die rohen Audiosamples ohne Zusatzinformationen übergeben. Einem Ausgabemodul, wie es die IEEE 1394-Komponente in diesem Projekt darstellt, sollen aber naturgemäß auch nicht mehr Daten übermittelt werden, um die Weiterverarbeitung möglichst simpel zu halten.

Da hier aber zusätzlich ein Synchronisationsteil entwickelt wurde, von dem der MPlayer natürlich nichts weiß, wären Informationen wie die RTP-Timestamps aus dem ursprünglichen auf IP laufenden Audiostream sehr hilfreich. Damit könnte man die Verbindung zwischen der Audioquelle und dem Gateway besser analysieren und die Resultate dann in die Synchronisationsentscheidungen mit einfließen lassen.

Um das zu erreichen, müssen die IEEE 1394-Funktionen entweder über eine andere Schnittstelle mit dem MPlayer verknüpft werden, oder es wird dem MPlayer bekannt gemacht, was genau in diesem modifizierten Ausgabemodul passiert, um dementsprechend mehr Detailinformationen vom Dekoder zu verlangen.

5.1.4 Optimierung des Log-Viewers

Aufgrund der momentan noch nicht optimierten Multithreading-Programmierung kann die Anzeige des Log-Viewers die Prozessorauslastung teilweise sehr stark in die Höhe treiben, wodurch kurzzeitige Verarbeitungsengpässe in den echtzeitkritischen Komponenten des Gateways eintreten können. Um so entstehende Aussetzer bei der Weiterleitung der Audiosamples auf den IEEE 1394-Bus zu verhindern, müssen den Threads Prioritäten zugeordnet werden, wobei dem Log-Viewer aufgrund seiner geringen Bedeutung für die Funktionsfähigkeit des Gateways nur ein niedriger Status eingeräumt werden darf.

5.2 Erweiterungen

Dieses Kapitel beschreibt einige Möglichkeiten zur Erweiterung der Software um neue Features respektive wird auch eine alternative Schnittstelle zwischen MPlayer und der IEEE 1394-Komponente kurz diskutiert.

5.2.1 Steuerung des Gateways über IEEE 1394

Eine sinnvolle Erweiterung der Gateway-Software wäre die Implementierung einer bidirektionalen Kommunikation mit den Endgeräten. Damit wäre es möglich, vom Endgerät aus zum Beispiel die Audioquelle zu wählen oder andere Informationen an das Gateway zurückzuliefern. Mit Hilfe der Standards AV/C oder HAVi wären dafür bereits passende und vor allem standardisierte Steuerbefehle definiert.

Momentan wird im IEEE 1394-Teil ausschließlich das Senden auf isochronen Kanälen unterstützt. Um den Endgeräten die Steuerung des Gateways zu ermöglichen, muss jedoch zusätzlich ein asynchroner Sende- und Empfangsteil implementiert werden. In der LIBRAW1394-Bibliothek sind ohne spezielle Berücksichtigung der oben genannten Standards dafür auch schon einige Funktionen vorbereitet, wobei das Linux1394-Projekt [36] sogar ein erweitertes API für den AV/C-Standard zur Verfügung stellt.

5.2.2 Parameter zum Umschalten auf MPlayer-Originalbetrieb

Um dem Anwender auch zu ermöglichen, die ursprüngliche Funktion des adaptierten Ausgabemoduls - das Schreiben in eine Datei - nutzen zu können, muss man eine Option integrieren, die, anstatt den neuen Quellcode abzuarbeiten, wieder zum alten wechselt. Für diese Erweiterung existieren zwei grundsätzliche Realisierungen:

1. Man belässt beide Quellcodeversionen in demselben Ausgabemodul und wechselt zwischen ihnen mit einer Art Schalter in der Konfigurationsdatei[58] ohne Kenntnis des MPlayers über diesen Vorgang.

2. Man gibt dem MPlayer ein zusätzliches Ausgabemodul bekannt, sodass das Schreiben in eine Datei und die Ausgabe auf den IEEE 1394-Bus jeweils in einer eigenen Datei behandelt werden. Dazu muss aber tiefer in den MPlayer-Code eingegriffen werden, und zwar an mehreren Stellen, was der Aufrechterhaltung der Kompatibilität mit zukünftigen Versionen entgegen wirkt.

[58] siehe Kapitel 4.5 auf Seite 83

5.2.3 Unterstützung zusätzlicher AMDTP-Formate

Im Standard IEC 61883-6 werden außer den in dieser Implementierung behandelten Audioformaten Multi-bit Linear Audio und IEC 60958 noch weitere spezifiziert, deren Integration in dieses Projekt eine Vervollständigung des Standards bedeuten würde:

- One-bit Audio
- MIDI (Musical Instrument Digital Interface) konforme Daten
- SMPTE (Society of Motion Picture and Television Engineers) Time Code
- Sample Count Data
- High-Precision Multi-bit Linear Audio
- verschiedenste standardisierte Zusatzinformationen zu den übertragenen Audiodaten

5.2.4 Alternative „Standalone Program"

Anstatt das Ausgabemodul ao_pcm.c des MPlayers zu modifizieren und damit in den originalen Quellcode einzugreifen, könnte man die LIB1394 auch zu einem eigenständigen Programm weiterentwickeln, das die Audiosamples entweder über eine Pipe vom MPlayer erhält oder von einer Datei einliest, in der sie kurz zuvor vom MPlayer geschrieben wurden. Diese Variante birgt sowohl Vorteile als auch Nachteile gegenüber der aktuell gewählten Implementierung.

- **Vorteile:**
  - Durch die vollständige Abtrennung vom MPlayer-Quellcode würden zukünftige Versionen des MPlayers noch leichter in das Gateway zu integrieren sein.
  - Ein eigenständiges Programm würde mehr Kontrolle über die Ressourcen und Abläufe während der Laufzeit bedeuten.
  - Es würden kein Verständnis der Interna des MPlayers mehr erforderlich sein.
- **Nachteile:**
  - Bei der Benutzung von Pipes oder Dateien als Datenquelle für die LIB1394 kommen zusätzliche Datenpuffer zum Einsatz, die mehr Verzögerung in der Verarbeitung der Daten verursachen und auf deren Parameter und Verhalten auch nicht modifizierend eingegriffen werden kann, da das System die Kontrolle darüber hat und keine Modifikationen erlaubt.

- Es würde die Möglichkeit verloren gehen, auf das Verhalten des MPlayers rückzuwirken, wie es über die Funktionen get_space(), play() und get_delay() im Ausgabemodul ao_pcm.c realisiert werden kann.
- Ein Bewertungskriterium in den Synchronisationsberechnungen verliert etwas an Zuverlässigkeit, und zwar die Geschwindigkeit, in der der MPlayer die Samples an die IEEE 1394-Komponente liefert und die in der aktuellen Implementierung eine entscheidende Rolle spielt. Bei Verwendung einer eigenständigen Software würde das System die Daten über bestimmte Mechanismen durchreichen, die die Analyse verfälschen und somit die Variation der Samplingrate negativ beeinflussen könnten. Die Geschwindigkeitsmessung kann zwar weiterhin in die Berechnung miteinbezogen werden, doch je mehr Stationen die Daten passieren müssen, desto mehr Zwischenfälle während der Verarbeitung können auch eintreten.

6 Zusammenfassung und Resultate

Zusammenfassend wäre zu sagen, dass das Vorhaben, eine Brücke zwischen zwei grundverschiedenen Kommunikationsstandards zu errichten, erfolgreich abgeschlossen wurde. AMDTP auf dem IEEE 1394-Bus und RTP auf IP-Basis besitzen zwar einige Gemeinsamkeiten, wie zum Beispiel die Verwendung in Umgebungen, in denen multimediale Datenströme in Echtzeit transportiert werden müssen, aber trotzdem wurden sie ursprünglich für verschiedene Bereiche entwickelt. IEEE 1394 wird hauptsächlich in überschaubaren Netzwerken zur effizienten unidirektionalen Übertragung von Audio- und Videodaten eingesetzt, IP hingegen findet seinen Zweck in der weltweiten bidirektionalen Kommunikation zwischen gleichberechtigten Partnern, die alle Arten von Informationen austauschen. Dennoch besteht die Motivation, diese beiden Welten zusammenzuführen, vor allem, um dem eingeschränkten Blickfeld der IEEE 1394-Geräte das Tor zu den enormen audiovisuellen Datenmengen des Internets zu öffnen.

Das Ziel des vorliegenden Projekts war es, sich mit den Vorgängen auseinanderzusetzen, die beim Transportieren von kontinuierlichen Datenströmen aus einem IP-basierten Netzwerk auf den FireWire-Bus und deren Anpassung und Konvertierung durchgeführt werden müssen. Die zeitliche Synchronisation der beiden Netzwerke bedeutete dabei die größte Schwierigkeit und musste einer genaueren Betrachtung unterzogen werden. Schlussendlich wurde ein Algorithmus gewählt, der auf Basis des Füllstands des Puffers, der an der Schnittstelle zwischen den beiden Standards positioniert wurde, die Präsentationsgeschwindigkeit der Daten im IEEE 1394-Netzwerk reguliert.

Die erste Version der Implementierung der theoretisch behandelten Themen konzentriert sich ausschließlich auf die Übertragung und Konvertierung von Audiodaten, jedoch geht es hauptsächlich um die allgemeine Aufdeckung der möglichen auftretenden Probleme und um die Suche nach einer Lösung zu diesen. Die tatsächliche Softwarekomponente stellt nur eine der praktischen Umsetzungen der Theorie dar und soll die gefundenen Algorithmen nur anschaulich demonstrieren. Deshalb bleibt auch noch viel Spielraum für Erweiterungen und Verbesserungen.

Eine wichtige tragende Rolle in der Implementierung spielte das Open-Source-Projekt MPlayer, das für die Rückkonvertierung der komprimierten Audiodaten in die ursprüngliche bei der Aufnahme entstandene Form verantwortlich ist. Der Grund für die Entscheidung für diese Engine liegt in der hervorragenden Unterstützung aktueller Multimediaformate, die zwar noch nicht perfekt ist, aber in zukünftigen Versionen sicherlich erweitert und von Fehlern befreit wird. Das entwickelte Ausgabemodul für den IEEE 1394-Bus ist aufgrund seiner engen Zusammenarbeit mit dem MPlayer in allen Linux-Systemen anwendbar, in denen auch der MPlayer reibungslos funktioniert. Somit ist auch eine gewisse Flexibilität gegeben, die aber aufgrund der Komplexität aktueller Systeme nicht garantiert werden kann.

Abschließend wäre noch zu sagen, dass dieses Projekt einen Beitrag zur Förderung von IEEE 1394 darstellen soll, das heißt, dass es dem Autor ein Anliegen wäre, dass dieser äußerst effiziente und gut gelungene Standard einen größeren Bekanntheitsgrad erreicht und somit auch verstärkt

eingesetzt wird. Je mehr Anwender und Programmierer sich mit FireWire beschäftigen, desto mehr werden die Hersteller in die Entwicklung von Produkten mit IEEE 1394-Unterstützung investieren, wodurch auch die Verbreitung ansteigt und erneut das Interesse in diese Technologie gesteigert wird.

Abkürzungsverzeichnis

AAC	Advanced Audio Coding
(A)DSL	(Assymetric) Digital Subscriber Line
AIFF	Audio Interchange File Format
AMDTP	Audio and Music Data Transmission Protocol
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
AV/C	Audio/Video Control
AVP	Audio Video Protocol
CIP	Common Isochronous Packet
CRC	Cyclic Redundancy Check
CSR	Control and Status Register
CSRC	Contributing Source Identifier
DBC	Data Block Count
DBS	Data Block Size
DLL	Dynamic Link Library
DTCP	Digital Transmission Content Protection
DVB	Digital Video Broadcasting
DVD	Digital Versatile Disc
FCP	Function Control Protocol
FDF	Format Dependent Field
FIFO	First In - First Out
FLA(C)	Free Lossless Audio (Codec)
FTP	File Transfer Protocol
GCC	GNU Compiler Collection
GMT	Greenwich Mean Time
GNOME	GNU Network Object Model Environment
GNU	GNU's Not Unix
GOF	Glass Optical Fiber
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communications
GTK	Gimp Toolkit
HAVi	Home Audio Video Interoperability
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronical Engineers
IETF	Internet Engineering Taskforce
IOCTL	Input/Output Control

IP	Internet Protocol
IPSec	Internet Protocol Security
IRM	Isochronous Resource Manager
ISDN	Integrated Services Digital Network
ISO	International Organization for Standardization
ISP	Internet Service Provider
MIDI	Musical Instrument Digital Interface
MMS(T)	Microsoft Media Server (TCP)
MPEG	Moving Picture Experts Group
MPLS	Multiprotocol Label Switching
NAT	Network Address Translation
NTP	Network Time Protocol
OSI	Open System Interconnection
PCM	Puls Code Modulation
POF	Plastic Optical Fiber
POSIX	Portable Operating System Interface
QoS	Quality of Service
RA	RealAudio
RAM	Random Access Memory
RDT	Real Data Transport
RFC	Request for Comments
RM	RealMedia
RTCP	RTP Control Protocol
RTSP	Real-Time Streaming Protocol
RTP	Realtime Transport Protocol
SDP	Session Description Protocol
SID	Source ID
SIP	Session Initiation Protocol
SMPTE	Society of Motion Picture and Television Engineers
SSRC	Synchronization Source Identifier
SVCD	Super-Video Compact Disc
SYT	Synchronization Timestamp
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications System
UPNP	Univeral Plug aNd Play
USB	Universal Serial Bus
UTC	Coordinated Universal Time

VCD	Video Compact Disc
VoIP	Voice over IP
WLAN	Wireless Local Area Network
WMA	Windows Media Audio

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Literatur

[1] Institute of Electrical and Electronical Engineers: *IEEE Information Technology - Portable Operating System Interface (POSIX®)*, IEEE 1003.1-2001, 2001

[2] Institute of Electrical and Electronical Engineers: *IEEE Standard for a High Performance Serial Bus*, IEEE 1394-1995, 1995

[3] Institute of Electrical and Electronical Engineers: *IEEE Standard for a High Performance Serial Bus - Amendment 1*, IEEE 1394a-2000, 2000

[4] Institute of Electrical and Electronical Engineers: *IEEE Standard for a High-Performance Serial Bus - Amendment 2*, IEEE 1394b-2002, 2002

[5] Postel J.: *Internet Protocol, DARPA Internet Program, Protocol Specification*, RFC 791, 1981. Verfügbar unter: http://www.ietf.org/rfc/rfc0791.txt [abgerufen am 17.01.2006]

[6] Deering S.: *Host Extensions for IP Multicasting*, RFC 1054, 1988. Verfügbar unter: http://www.ietf.org/rfc/rfc1054.txt [abgerufen am 17.01.2006]

[7] Mills D.L.: *Network Time Protocol (Version 3), Specification, Implementation and Analysis*, RFC 1305, 1992. Verfügbar unter: http://www.ietf.org/rfc/rfc1305.txt [abgerufen am 17.01.2006]

[8] Deering S., Hinden R.: *Internet Protocol, Version 6 (IPv6) Specification*, RFC 1883, 1995. Verfügbar unter: http://www.ietf.org/rfc/rfc1883.txt [abgerufen am 17.01.2006]

[9] Schulzrinne H., Casner S., Frederick R., Jacobson V.: *RTP: A Transport Protocol for Real-Time Applications*, RFC 1889, 1996. Verfügbar unter: http://www.ietf.org/rfc/rfc1889.txt [abgerufen am 17.01.2006]

[10] Fielding R., Gettys J., Mogul J., Frystyk H., Berners-Lee T.: *Hypertext Transfer Protocol – HTTP/1.1*, RFC 2068, 1997. Verfügbar unter: http://www.ietf.org/rfc/rfc2068.txt [abgerufen am 17.01.2006]

[11] Schulzrinne H., Rao A., Lanphier R.: *Real Time Streaming Protocol (RTSP)*, RFC 2326, 1998. Verfügbar unter: http://www.ietf.org/rfc/rfc2326.txt [abgerufen am 17.01.2006]

[12] Handley M., Jacobson V.: *SDP: Session Description Protocol*, RFC 2327, 1998. Verfügbar unter: http://www.ietf.org/rfc/rfc2327.txt [abgerufen am 17.01.2006]

[13] Rosenberg J., Schulzrinne H., Camarillo G., Johnston A., Peterson J., Sparks R., Handley M., Schooler E.: *SIP: Session Initiation Protocol*, RFC 3261, 2002. Verfügbar unter: http://www.ietf.org/rfc/rfc3261.txt [abgerufen am 05.02.2006]

[14] Schulzrinne H., Casner S., Frederick R., Jacobson V.: *RTP: A Transport Protocol for Real-Time Applications*, RFC 3550, 2003. Verfügbar unter: http://www.ietf.org/rfc/rfc3550.txt [abgerufen am 17.01.2006]

[15] International Electrotechnical Commission: *Consumer audio/video equipment - Digital interface - Part 1: General*, 61883-1, 1st Edition, 1998

[16] International Telecommunication Union: *Packet-based multimedia communications systems*, Recommendation H.323 (07/03), 2003

[17] Hitachi, Intel Corporation, Matsushita Electric Industrial, Sony Corporation, Toshiba Corporation: *5C Digital Transmission Content Protection (White Paper)*, Revision 1.0, 1998

[18] 1394 Trade Association: *TA Document 2001024, Audio and Music Data Transmission Protocol 2.1*, 2002

[19] 1394 Trade Association: *TA Document 1998001, AV/C Digital Interface Command Set General Specification*, 1998

[20] HAVi, Inc.: *The HAVi Specification: Specification of the Home Audio/Video Interoperability (HAVi) Architecture*, 2001

[21] Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, Philips: *Universal Serial Bus Specification*, 2000, Verfügbar unter http://www.usb.org/developers/docs/usb_20_02212005.zip [abgerufen am 23.01.2006]

[22] Tanenbaum A.S., Van Steen M.: *Distributed Systems, Principles and Paradigms*, 2002, Prentice Hall, 803 Seiten, ISBN: 0-13-088893-1 (Hardcover)

[23] Baron G., Kirschenhofer P.: *Einführung in die Mathematik für Informatiker, Band 3*, 2. Auflage, 1996, Springer, 191 Seiten, ISBN: 3-211-82797-8 (Paperback)

[24] Mindshare Inc., Anderson D.: *FireWire System Architecture, Second Edition, IEEE1394a*, 1999, Addison-Wesley, 509 Seiten, ISBN: 0-201-48535-4 (Paperback)

[25] Stallings W.: *Operating Systems: Internals and Design Principles, Third Edition*, 1998, Prentice-Hall International, 781 Seiten, ISBN: 0-13-917998-4 (Paperback)

[26] Weihs M., Ziehensack M.: *Convergence between IEEE 1394 and IP for real-time A/V transmission (White Paper)*, 2003, Institute of Computer Technology, Vienna University of Technology

[27] Zuser W., Biffl S., Grechenig T., Futschek G.: *Software Engineering, Grundlagen, Methoden und Anleitung zum erfolgreichen Softwareprojekt*, Skriptum zur Vorlesung Software Engineering 1 & 2, Version 1.0, 2000, 301 Seiten

[28] Dusan Zivadinovic: *Schöne neue Funk-Welt*, c't Magazin für Computertechnik 02/05, S.128, Heise Verlag

[29] MPlayer-Dokumentation: *libao2: this control audio playing*, /<MPlayer-Hauptverzeichnis>/DOCS/tech/libao2.txt

[30] Ippolito G.: *YoLinux Tutorial: POSIX thread (pthread) libraries* [Online], Verfügbar unter: http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html [abgerufen am 23.01.2006]

[31] 1394 Trade Association: *1394 Products* [Online], Verfügbar unter: http://www.1394ta.org/About/products/index.html [abgerufen am 23.01.2006]

[32] Wikipedia: *Codec* [Online], Verfügbar unter http://de.wikipedia.org/wiki/Codec [abgerufen am 23.01.2006]

[33] Wikipedia: *Audiokompression* [Online], Verfügbar unter http://de.wikipedia.org/wiki/Audiokompression [abgerufen am 23.01.2006]

[34] Free Software Foundation, Inc.: *GNU GENERAL PUBLIC LICENSE, Version 2*, 1991, Verfügbar unter http://www.gnu.org/licenses/gpl.txt [abgerufen am 23.01.2006]

[35] The MPlayer Project: *MPlayer - The Movie Player* [Online], Verfügbar unter http://www.mplayerhq.hu [abgerufen am 23.01.2006]

[36] The Linux1394 Project: *IEEE 1394 for Linux* [Online], Verfügbar unter http://www.linux1394.org [abgerufen am 23.01.2006]

[37] Live Networks, Inc.: *LIVE555.COM: Internet Streaming Media, Wireless, and Multicast technology, services & standards* [Online], Verfügbar unter http://www.live555.com [abgerufen am 23.01.2006]