Hochschule für Technik und Wirtschaft Dresden
Fakultät Informatik/Mathematik

Diplomarbeit

im Studiengang Medieninformatik

zum Thema

Entwicklung eines JPEG-Dateianalysators

Eingereicht von: Emanuel Günther, SG 17/043, MN 43618
Eingereicht am: 30.08.2021
Betreuer: Prof. Dr.-Ing. Jörg Vogt / HTW Dresden
Zweitgutachter: Prof. Dr.-Ing. Jens Schönthier / HTW Dresden

Inhaltsverzeichnis

Abkürzungsverzeichnis

2D-DCT
: zweidimensionale diskrete Kosinustransformation

AES
: Advanced Encryption Standard

AES-CM
: Counter-Modus des AES

APP0
: Application 0

AVI
: Audio Video Interleave

bzw.
: beziehungsweise

DES
: Data Encryption Standard

DHT
: Define Huffman Table(s)

DQT
: Define Quantization Table(s)

DRI
: Define restart interval

ECB
: Electronic Code Book

engl.
: englisch

EOI
: End of Image

EtC

Encryption-Then-Compression

f.

folgend

FEC

Forward Error Correction

ff.

fortfolgend

FOURCC

four-character code

HTWDD

Hochschule für Technik und Wirtschaft Dresden

JDK

Java Development Kit

JFIF

JPEG File Interchange Format

LSB

Least Significant Bit

MAC

Message Authentication Code

max.

maximal

MIME

Multipurpose Internet Mail Extensions

MJPEG

Motion JPEG

QTFF

QuickTime File Format

RIFF

Resource Interchange File Format

RST

Restart

RTCP

RTP control protocol

RTP

Real-time transport protocol

RTSP

Real-Time Streaming Protocol

S.

Seite

SDP

Session Description Protocol

SOF

Start of Frame

SOF0

Start of Frame, Baseline DCT mit Huffman-Entropiecodierung

SOI

Start of Image

SOS

Start of Scan

SRTP

Secure Real-time Transport Protocol

TLS

Transport Layer Security

vgl.

vergleiche

WAV

Waveform Audio File Format

WebRTC

Web Real-Time Communication

Glossar

atom

Basisbausteine für Informationen im QuickTime File Format

Chunk

"Blattstruktur" für Daten im Resource Interchange File Format

diskrete Kosinustransformation

Überführung von Daten in den Frequenzbereich

four-character code

32-bit unsigned Integer, der aus der Aneinanderreihung von vier ASCII-Zeichen besteht und zur Typidentifikation von Dateien und Datenblöcken verwendet wird

interleaved

Art der Datenspeicherung, bei der jeweils die codierten Blöcke der Farbkomponenten eines Bildblocks aufeinanderfolgend gespeichert werden, *siehe auch* non-interleaved

Joint Photographic Experts Group

Entwickler des gleichnamigen Kompressionsverfahrens JPEG

JPEG

Verbreitetes Format zur Bildkompression für Bilder mit natürlichen Inhalten, *siehe auch* Joint Photographic Experts Group

JPEG File Interchange Format

gebräuchliches Format zur Speicherung von JPEG-Daten

Least Significant Bit

Jenes Bit eines Bytes, welches darüber entscheidet, ob die dargestellte Zahl gerade oder ungerade ist. Die Least Significant Bits entsprechen den dem LSB am nächsten stehenden Bits, inklusive des LSBs selbst. (vgl. [33])

List

Listenstruktur im Resource Interchange File Format, die wiederum Listen oder Chunks enthält

Motion JPEG

Aus einzelnen JPEG-Bildern bestehendes Videoformat

Multipurpose Internet Mail Extensions

standardisierte Struktur zur Kennzeichnung von Daten oder Dateien hinsichtlich ihres Inhalts und ihrem Aufbau

non-interleaved

Art der Datenspeicherung, bei der die codierten Blöcke einer Farbkomponente aufeinanderfolgend gespeichert werden, *siehe auch* interleaved

NULL

Bezeichnung für die Chiffre im SRTP, welche keine Verschlüsselung durchführt

Padding

Erweiterung eines Datenfeldes mit beliebigen Daten, um eine vorgegebene Datenmenge zu erhalten

RGB-Modell

Farbmodell mit den Komponenten Rot, Grün und Blau

Thumbnail

Kleines Vorschaubild für Bild- oder Videodateien

XML

Extensible Markup Language, Sprache für die Darstellung von strukturierten Daten

YCbCr-Modell

Farbmodell, bei dem Y für die Helligkeit steht und Cb bzw. Cr die Blau- bzw. Rot-Chrominanzen (Blau- bzw. Rot-Abweichungen) beschreiben

zweidimensionale diskrete Kosinustransformation

Überführung von zweidimensionalen Daten (zum Beispiel Bilddaten) in den Frequenzbereich, *siehe auch* diskrete Kosinustransformation

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Obwohl es bereits 1994 standardisiert und seitdem nicht geändert wurde, ist JPEG das am häufigsten genutzte Bildformat. Sowohl Foto- als auch die heutzutage weitaus gebräuchlicheren Smartphonekameras nutzen es als Standardformat. Der Erfolg des Formates hängt unter anderem damit zusammen, auf welche Weise die Bilder in Dateien gespeichert werden. Das Datenformat reduziert die Größe der Bilder durch Kompression und sorgt so für kleinere Dateien. Diese ermöglichen nicht nur die effiziente Speicherung von Bilddaten auf Datenträgern. Auch im Zeitalter von Social Media tragen kleine Dateien zum schnellen Anzeigen von Inhalten bei.

Die weite Verbreitung des JPEG-Formates zeigt sich allerdings nicht nur bei der Speicherung von Bildern. Das Format bildet gleichzeitig auch die Grundlage für eine einfache Videokompression. Durch die Aneinanderreihung mehrerer JPEG-Bilder entsteht ein Videoformat, welches als Motion JPEG (MJPEG) bezeichnet wird. Das Anzeigen dieser Bilder in einer bestimmten Geschwindigkeit erzeugt ein Video. Gerade für Geräte mit wenig Rechenleistung bietet diese Art der Videocodierung die Möglichkeit, Aufnahmen in Echtzeit zu komprimieren und zu übertragen. Anwendung findet MJPEG beispielsweise als Überwachungskameras, die als „Internet of Things“-Geräte im Smart Home eingesetzt werden.

Damit internetgebundene Kameras in Echtzeit übertragen können, wird ein Protokoll für die Übertragung benötigt. Dieses enthält einige Einschränkungen, welche die Übertragung erleichtern und die zu übertragende Datenmenge reduzieren. Die von Kameras aufgezeichneten bzw. anderweitig erzeugten Videodaten müssen demnach bestimmte Kriterien erfüllen, um übertragen werden zu können.

Die Professur für Rechnernetze und Kommunikationssysteme der Hochschule für Technik und Wirtschaft Dresden (HTWDD) hat bereits ein Projekt entwickelt, welches die Übertragung von MJPEG-Videos ermöglicht. Zur Implementierung nutzt es die Programmiersprache Java und deren Softwarebibliotheken. Für die Lehre wird die entsprechende Software bereits im Modul Internettechnologien II eingesetzt. Dafür werden ausschließlich aufgezeichnete Videos und keine Livedaten verwendet, was das Testen der Funktionalität im Rahmen von Veranstaltungen deutlich vereinfacht. Es zeigte sich allerdings, dass eine erfolgreiche Übertragung eines Videos durch die Projektsoftware nur mit bestimmten MJPEG-Videodateien möglich ist. Deshalb ist ein Ziel dieser Arbeit, bestehende MJPEG-Videos hinsichtlich der Eignung zum Streaming mit dem echtzeitfähigen Real-time transport protocol (RTP) zu überprüfen. Dabei spielen sowohl das Format der entsprechenden MJPEG-Dateien als auch die Kompressionsparameter der darin enthaltenen JPEG-Bilder eine Rolle. Es soll im Vorfeld der Übertragung festgestellt werden, ob eine gegebene Videodatei mit diesem Protokoll übermittelt werden kann. Um dieselbe Entwicklungsumgebung wie im Projekt der HTWDD zu verwenden, soll ebenfalls die Programmiersprache Java verwendet werden.

Das Softwareprojekt für die Übertragung von MJPEG-Videos wird auch auf den privaten Computern der Studenten ausgeführt. Dadurch zeigte sich, dass die Prozessorauslastung durch die Nutzung der Software auf schwächerer Hardware steigt und deshalb die Videos nicht in Echtzeit angezeigt werden können. Aus diesem Grund soll das benannte Projekt hinsichtlich seiner Ausführungszeit optimiert werden. Das Ziel ist dabei, die flüssige Wiedergabe von Videos auf einer Vielzahl an Architekturen zu ermöglichen.

Neben den angesprochenen Verbesserungen des Projektes und seiner Nutzung soll dieses auch in seiner Funktionalität erweitert werden. Eine Möglichkeit dafür besteht in der Implementierung einer Audioübertragung, sodass Videos beim Empfänger mitsamt ihrer Tonspur abgespielt werden können. Der hier gewählte Ansatzpunkt ist allerdings die Verschlüsselung der übertragenen Daten. Diese Art der Sicherung spielt eine immer größere Rolle in realen Anwendungen, wie beispielsweise bei nicht-öffentlichen Videokonferenzen. Sensitive oder personenbezogene Daten sollen nicht in die Hände Dritter gelangen können. Genau das ist aber bei der ungesicherten Übermittlung von Daten über das Internet möglich. Ein Schutz der Daten kann mit deren Verschlüsselung erreicht werden. Auf diese Weise können nur Sender und Empfänger den Inhalt der Daten einsehen. Für Dritte bleiben diese unlesbar. Aus diesem Grund sollen in dieser Arbeit mögliche Verschlüsselungskonzepte für das Streaming von Videos untersucht werden. Der Fokus wird dabei auf die Übertragung von den im Streamingprojekt verwendeten MJPEG-Videos gelegt. Anschließend an die Untersuchung soll eine praktische Demonstration der untersuchten Verfahren und Konzepte an einer vorhandenen Streamingsoftware erfolgen, die das Protokoll Real-Time Streaming Protocol (RTSP) implementiert.

Die Grundlagen aller weiteren Erörterungen werden in der Betrachtung des JPEG-Kompressionsverfahrens und dem Aufbau von MJPEG-Dateien gelegt. Sowohl die Analyse von Videodateien als auch die Verschlüsselungen erfordern die dabei gewonnenen Erkenntnisse. Den Anschluss daran bildet die Ermittlung der Anforderungen für JPEG-Übertragungen per RTP. Diese werden in ein Analyseprogramm umgesetzt, welches MJPEG-Dateien auf die Erfüllung der erhaltenen Anforderungen überprüft. Als Übergang zur Arbeit an und mit dem Projekt der HTWDD zum RTSP-Streaming folgen daran die Optimierungen desselben. Inhalt ist dabei neben der Verkürzung der Ausführungszeiten bestimmter Operationen auch die Herstellung der Kompatibilität mit dem VLC Media Player, welches ebenfalls eine RTSP-Implementierung enthält. Ziel dieser Kompatibilität ist, die Funktionalitäten des Projektes mithilfe des externen Programmes zu überprüfen. Das sich daran anschließende Kapitel zur Vorbereitung der Verschlüsselung von Mediendaten leitet die Betrachtungen verschiedener Verschlüsselungskonzepten ein. Daran folgend wird das Protokoll Secure Real-time Transport Protocol (SRTP) analysiert, welches neben einer Verschlüsselung auch die Authentifikation von Paketen bereitstellt. Auf die Implementierung dieses Protokolls und das Testen derselben wird ebenfalls eingegangen. Ein ähnliches Vorgehen gilt für die Betrachtung der JPEG-spezifischen Verschlüsselung. Zunächst werden verschiedene Möglichkeiten für den Entwurf eines solchen Verfahrens beleuchtet. Im Anschluss daran wird ein konkretes Verfahren implementiert und getestet. Den Abschluss bildet die Zusammenfassung der gewonnenen Erkenntnisse, welche auch einen Ausblick auf mögliche Fortsetzungen der vorliegenden Arbeit umfasst.

2. JPEG-Kompression und das JFIF-Datenformat

Die Abkürzung JPEG des vielverwendeten Kompressionsverfahrens steht für die Joint Photographic Experts Group, welche das Verfahren entworfen hat. Standardisiert ist die JPEG-Kompression als ISO/IEC 10918-1 [8]. In diesem Dokument werden die Anforderungen an Encoder und Decoder beschrieben, welche für die Erzeugung und das Verarbeiten von JPEG-Daten erfüllt sein müssen. Dazu gehören neben den für die Kompression zu verwendenden Algorithmen auch das in Annex B des Standards beschriebene Datenformat für die komprimierten Daten. Auf dieses Format wird in Abschnitt 2.2 genauer eingegangen. Vorgaben bei der Verwendung bestimmter Parameter für die Algorithmen gibt es nicht. Allerdings empfiehlt der Standard bestimmte Werte und Tabellen, mit denen im Allgemeinen gute Ergebnisse erzielt werden.

Aufbauend auf dem im JPEG-Standard definierten Datenformat hat sich das JPEG File Interchange Format (JFIF)-Format entwickelt. Dieses ist als ISO/IEC 10918-5 [9] standardisiert. Das Dokument erweitert das JPEG-Format um einen Abschnitt, in welchem Daten zur Anzeigegröße des Bildes gespeichert werden und die Möglichkeit für das Speichern eines Thumbnails gegeben wird. Außerdem werden Einschränkungen bezüglich bestimmter Parameter der Kompression getroffen, zu denen unter anderem die Vorgabe eines zu verwendenden Farbmodells gehört. Aufgrund dieser Anpassungen ist das JFIF-Format mittlerweile das übliche Format zur Speicherung und Übertragung von JPEG-Kompressionsdaten.

Nachfolgend soll zunächst eine kurze Übersicht über das JPEG-Kompressionsverfahren gegeben werden. Dabei wird ausschließlich auf den Baseline-Modus der Kompression eingegangen, da nur dieser im weiteren Verlauf der Arbeit eine Rolle spielen wird. Im Anschluss daran wird der Aufbau des JFIF-Dateiformates beschrieben. Dieser wird später beim Auslesen und der Analyse der Kompressionsparameter benötigt.

2.1. Das JPEG-Kompressionsverfahren im Baseline-Modus

Im JPEG-Standard werden vier verschiedene Kompressionsmodi beschrieben. Neben dem Baseline-Modus gibt es den erweiterten, den verlustfreien und den hierarchischen Modus. Jedoch muss jeder Encoder oder Decoder für JPEG-Bilder den Baseline-Modus implementieren (vgl. [27], S. 204). Auch deshalb ist dieser der am weitesten verbreitete der JPEG-Modi. Dass der Baseline-Modus eine verlustbehaftete Kompression durchführt, ist dabei keine Einschränkung hinsichtlich seiner Verbreitung. Im Gegensatz zu verlustfreien Verfahren wird dabei nur eine Annäherung an das Originalbild codiert. So kann die zur Speicherung von Bildern benötigte Datenmenge deutlich reduziert werden. Das

komprimierte Bild ist dem Originalbild jedoch visuell sehr ähnlich, wodurch keine negativen Effekte für die Betrachtung enstehen.

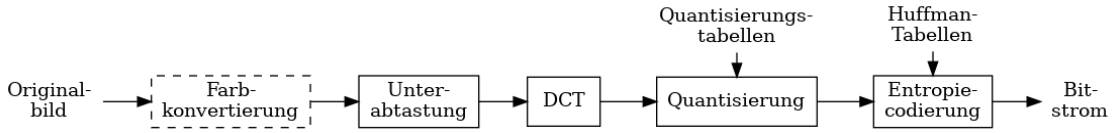


Abbildung 2.1.: Der Ablauf einer JPEG-Kompression im Baseline-Modus. Die Farbkonvertierung ist optional. Quantisierungs- und Huffmantabellen gehen als Parameter in den jeweiligen Kompressionsschritt ein. (verändert nach [27], S. 195).

Die Kompression eines digitalen Bildes erfolgt im Baseline-Modus in mehreren Schritten, die auch in Abbildung 2.1 dargestellt sind. Zu Beginn steht die optionale Konvertierung der Farbkomponenten in ein anderes Farbmodell. Die Farbkomponenten können beispielsweise im RGB-Modell vorliegen und in das YCbCr-Modell konvertiert werden. Das JPEG-Verfahren selbst ist farbenblind (vgl. [27], S. 194). Die Algorithmen funktionieren also unabhängig von der Interpretation der Farbinformationen. Im JFIF-Format ist die Interpretation jedoch auf das YCbCr-Modell festgelegt. Deshalb ist die Konvertierung der Bilddaten notwendig, sollten diese nicht im genannten Farbmodell vorliegen. Für solche Konvertierungen existieren auch standardisierte Umrechnungsvorschriften. Beispielsweise wird die Umrechnung vom RGB-Modell ins YCbCr-Modell in der Empfehlung „Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios“ ([10], S. 3) beschrieben und im JFIF-Standard ([9], S. 3) konkretisiert.

Liegen die Bildinformationen im gewünschten Farbraum vor, können sie einer Farbunterabtastung unterzogen werden. Diese macht sich zunutze, dass das menschliche Auge Helligkeitsinformationen besser wahrnimmt als Farbinformationen. Somit fällt es dem Auge nicht weiter auf, wenn nur noch ein Teil der Farbinformationen vorhanden sind. Auf diese Weise kann die Datenmenge eines Bildes deutlich reduziert werden.

Für die übliche Farbunterabtastung ist es erforderlich, dass die erste Komponente den Helligkeitswert beschreibt. Die beiden anderen Komponenten beschreiben Farbanteile. Für den Fall, dass keine Unterabtastung vorgenommen wurde, wird das Verhältnis der Komponenten als `4:4:4` beschrieben. Bei `4:2:2` wird die erste Komponente unberührt gelassen, während die beiden Farbanteile jeweils horizontal halbiert werden. Damit wird die Datenmenge um ein Drittel reduziert. Durch eine horizontale und vertikale Halbierung der Farbinformationen wird das `4:2:0`-Format erreicht. Die Helligkeitskomponente behält ihr ursprüngliches Format. Diese Unterabtastung resultiert in einer Halbierung der Datenmenge. Neben den genannten Verhältnissen gibt es noch weitere, die durch Farbunterabtastung entstehen können. Allerdings werden meist die hier beschriebenen Formate verwendet.

Im Anschluss wird das Bild in Blöcke von 8x8 Pixeln unterteilt. Das vereinfacht nicht nur die Weiterverarbeitung, sondern setzt auch den Grundstein für die weitere Kompression. Danach wird für jeden Block eine zweidimensionale diskrete Kosinustransformation (2D-DCT) ausgeführt. Diese Transformation beruht auf dem Prinzip, dass jede Funktion durch eine Reihe an Kosinusfunktionen beschrieben werden kann. Im Fall von JPEG entspricht der Bildblock von 8x8 Pixeln dieser Funktion. Die Basisfunktionen der

2D-DCT, welche zur Beschreibung eines Bildblocks zur Verfügung stehen, sind in Abbildung 2.2 dargestellt. Durch die diskrete Kosinustransformation werden die Koeffizienten der Funktionen berechnet, aus deren Summe sich das Originalbild erzeugen lässt. Der Bildblock des Originalbildes wird also lediglich in eine andere Repräsentation überführt (vgl. [27], S. 155). Das hat jedoch zur Folge, dass die einzelnen Werte des Blocks dekorreliert werden.

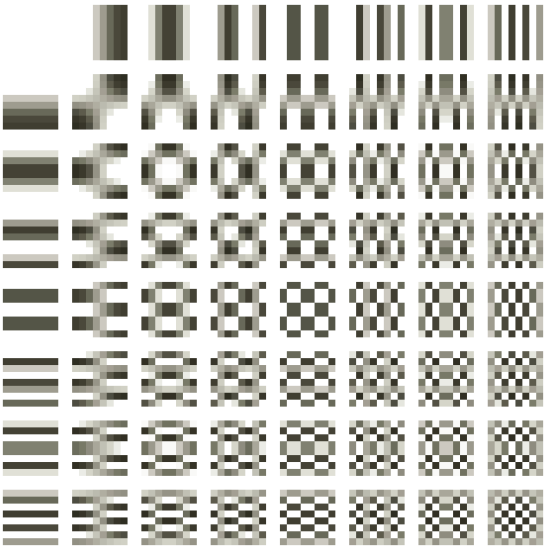


Abbildung 2.2.: Die Basisfunktionen der 2D-DCT für einen Block von 8x8 Pixeln. Der Wertebereich ist das Intervall $[-1, 1]$, wobei Werte nahe 1 hell und Werte nahe -1 dunkel dargestellt sind. (aus [27], S. 196)

Nach der Transformation des Bildblocks werden die entstandenen Koeffizienten quantisiert. Bei der Quantisierung wird jeder Koeffizient S_{uv} auf einen kleineren Wert $Sq_{uv} = round(S_{uv}/Q_{uv})$ abgebildet. Dabei wird die Größe der Quantisierungsstufen Q_{uv} aus einer Quantisierungstabelle entnommen. Auf diese Weise werden wichtige Koeffizienten beibehalten. Dazu gehören vor allem die der mittleren Helligkeit und der Grundstrukturen, die in Abbildung 2.2 oben links zu sehen sind. Die Koeffizienten unwichtiger Anteile, in Abbildung 2.2 unten rechts zu sehen, werden auf den Wert `0` reduziert.

Im Anschluss an die Quantisierung erfolgt eine Aufarbeitung der Koeffizienten für die Entropiecodierung. Statt des DC-Koeffizienten selbst, der die mittlere Helligkeit des Blockes repräsentiert, wird die Differenz zum DC-Koeffizienten des vorigen Blockes codiert. So werden die Werte für die mittlere Helligkeit benachbarter Blöcke dekorreliert. Anschließend werden die AC-Koeffizienten, das heißt alle verbleibenden Koeffizienten, umsortiert. Das geschieht nach absteigender Relevanz in einem Zick-Zack-Muster. Damit stehen der DC-Anteil und die wichtigen AC-Anteile am Anfang. Alle Koeffizienten am Ende der sortierten Daten, die Null betragen, werden weggelassen. Damit wird wiederum die Menge an Daten reduziert.

Abschließend werden die quantisierten Transformationskoeffizienten entropiecodiert. Im Baseline-Modus wird dafür das Huffman-Verfahren verwendet. Das Prinzip dieses Verfahrens ist, dass häufig auftretende Koeffizienten durch einen kurzen Code repräsentiert werden. Selten auftretende Werte erhalten einen längeren Code. Die so erzeugten

Codes werden als Huffman-Tabellen im komprimierten Bild integriert, damit dieses auch wieder decodiert werden kann. Auf diese Weise wird die Datenmenge des Bildes weiter reduziert. Ein weiterer Vorteil des Huffman-Verfahrens ist die Erzeugung von präfixfreien Codes (vgl. [21], S. 214). Das bedeutet, dass kein Codewort der Beginn eines anderen Codeworts sein kann. So kann kein Codewort durch unvollständiges Auslesen für ein anderes Codewort gehalten werden.

2.2. Das JFIF-Datenformat

Die bei der JPEG-Kompression entstandenen Daten werden im JFIF-Format in einer Datei gespeichert. Die Abkürzung JFIF steht für JPEG File Interchange Format. Das Format ist im Standard ISO10918-5 [9] definiert. Es basiert auf dem Vorschlag für ein binäres Datenformat in Annex B des JPEG-Standards [8]. Dieser Vorschlag wird um ein Datenfeld erweitert, in welchem Informationen zum Dateiformat und zur Pixeldichte enthalten sind. Außerdem besteht auch die Möglichkeit, ein Thumbnail zu integrieren. Eine weitere Festlegung des JFIF-Formats ist die Verwendung des YCbCr-Modells. Damit müssen Graustufenbilder eine und farbige Bilder alle drei Farbkomponenten enthalten. Zuletzt besteht auch die Möglichkeit für JFIF-Erweiterungen. Die im Standard definierten Erweiterungen beziehen sich ausschließlich auf verschiedene Datenformate des integrierbaren Thumbnails. Deshalb soll im weiteren Verlauf nicht weiter diese eingegangen werden. Des Weiteren wird hier, wie schon zum Beginn des Abschnitts 2 erwähnt, ausschließlich auf den Baseline-Modus des JPEG-Verfahrens eingegangen.

Eine JFIF-Datei besteht aus einer Sequenz von Segmenten. Der Beginn jedes Segmentes wird durch einen Marker gekennzeichnet, auf welchen ein Datenfeld mit der Länge des Segmentes folgt. Es gibt verschiedene Typen an Markern, die Aufschluss auf die Art der im Segment enthaltenen Daten und deren Struktur geben. Die in dieser Arbeit relevanten Marker sind in Tabelle 2.1 zu finden. Ausnahmen von dieser Struktur bilden die Start of Image (SOI)-, End of Image (EOI)- und Restart (RST)-Marker. Diese stehen für sich allein und sind nicht der Anfang eines Segmentes. SOI- und EOI-Marker beschreiben den Start bzw. das Ende eines JPEG-Bildes. Alle Informationen, die zur Decodierung des Bildes benötigt werden, müssen sich zwischen den beiden genannten Markern befinden. Die RST-Marker werden später noch genauer erläutert.

Auf den SOI-Marker folgt das Application 0 (APP0)-Segment. Dieses macht von der im JPEG-Standard gegebenen Möglichkeit Gebrauch, anwendungsspezifische Daten in das Bild zu integrieren. Das APP0-Segment ist im JFIF-Format beschrieben und enthält den nullterminierten String „JFIF“ und die Versionsnummer des verwendeten Formates. Außerdem kann das optionale Thumbnail an dieser Stelle definiert werden. Da es jedoch im weiteren Verlauf der Arbeit nicht von Bedeutung ist, folgen hier keine weiteren Ausführungen zu den damit verbundenen Inhalten. In dem genannten Segment finden sich weiterhin auch die Angaben zur horizontalen und vertikalen Pixeldichte. Diese sind entweder ohne Einheit, in Bildpunkten pro Inch oder Bildpunkten pro Zentimeter angegeben. Unabhängig von deren Einheit lässt sich aus den beiden Pixeldichten das Seitenverhältnis der Pixel berechnen. Quadratische Pixel mit dem Seitenverhältnis $1:1$ sind in der digitalen Bildverarbeitung mittlerweile üblich. Allerdings treten bei manchen Verfahren, wie beispielsweise der digitalen Codierung analoger Fernsehsysteme, nichtquadratische Pixel auf (vgl. [10]).

Hexadezimalcode	Symbol	Beschreibung
0xFFC0	SOF_0	Start of frame, Baseline DCT
0xFFC4	DHT	Define Huffman table(s)
0xFFD0 - 0xFFD7	RST_m*	Restart with modulo 8 count „m“
0xFFD8	SOI*	Start of image
0xFFD9	EOI*	End of image
0xFFDA	SOS	Start of scan
0xFFDB	DQT	Define quantization table(s)
0xFFDD	DRI	Define restart interval
0xFFE0 - 0xFFEF	APP_n	Reserved for application segments

Tabelle 2.1.: Wichtige Marker in JFIF-Dateien. Symbole mit Asterisk (*) beschreiben Marker, auf die kein Segment folgt. (verändert nach [8], S. 32)

Auf diese Informationen folgen Define Quantization Table(s) (DQT)- und Define Huffman Table(s) (DHT)-Segmente. Diese beschreiben die bei der JPEG-Kompression verwendeten Quantisierungs- und Huffman-Tabellen. Der Aufbau ist dabei für beide Segmente ähnlich: Bei Quantisierungstabellen wird zunächst die Präzision der Daten definiert, gefolgt von einer Identifikationsnummer. Darauf folgen die Einträge der Tabelle. Die Einträge von Huffman-Tabellen besitzen eine definierte Genauigkeit von 8 Bit. Dafür wird neben der Identifikationsnummer auch eine Klasse angegeben, welche die Tabelle für die Verwendung für Gleich- bzw. Wechselanteile qualifiziert. Daran anschließend folgen wieder die Inhalte der Tabelle. Sowohl für DQT- als auch für DHT-Segmente gilt: Es können mehrere dieser Segmente im Bild enthalten sein und jedes Segment kann mehrere Tabellen definieren. Im Fall von mehreren Tabellen pro Segment wiederholt sich der eben beschriebene Aufbau für jede Tabelle.

Sind die bei der Kompression verwendeten Tabellen definiert, folgt im Baseline-Modus das Segment Start of Frame, Baseline DCT mit Huffman-Entropiecodierung (SOF0). Neben diesem gibt es noch weitere Start of Frame (SOF)-Marker, die jeweils den bei der Kompression verwendeten Modus angeben. Der SOF0-Marker beschreibt im Baseline-Modus komprimierte Daten, die mit dem Huffman-Verfahren entropiecodiert wurden. In den meisten Fällen darf ein JPEG-Bild nur ein SOF-Segment besitzen. Die einzige Ausnahme bildet dabei die Kompression im hierarchischen Modus, auf den hier jedoch nicht weiter eingegangen wird.

In jedem SOF-Segment werden die Präzision der komprimierten Daten sowie die Dimensionen des Bildes angegeben. Zusätzlich dazu wird die Anzahl der Komponenten im Bild definiert, auf welche die Beschreibung jeder einzelnen Komponente folgt. Die Beschreibung einer Komponente besteht dabei aus ihrer Identifikationsnummer, der horizontaler und vertikaler Abtastrate sowie der Identifikationsnummer der verwendeten Quantisierungstabelle. Werden die Abtastraten aller Komponenten miteinander verglichen, lässt sich daraus das aus der Farbunterabtastung entstandene Verhältnis ermitteln.

Nach den Informationen des SOF-Segments folgen ein oder mehrere Start of Scan (SOS)-Segmente. Wie viele dieser Segmente im Bild enthalten sind, hängt von der Reihenfolge der codierten Blöcke im Datenstrom ab. Werden die Blöcke der Farbkomponenten getrennt voneinander gespeichert, sodass alle Blöcke einer Komponente aufeinander folgen, heißt die Art der Speicherung non-interleaved. Daraus resultieren mehrere SOS-

Segmente, da in jedem Scan alle codierten Blöcke einer Komponente gespeichert werden. Für jede Komponente des Bildes würde so ein SOS-Segment entstehen. Alternativ dazu können die Daten interleaved gespeichert werden. Das bedeutet, dass die codierten Blöcke aller Farbkomponenten eines Bildblockes aufeinanderfolgend gespeichert werden. Aus dieser Art der Speicherung entsteht ein einziges SOS-Segment, da die Blöcke der Farbkomponenten immer abwechselnd nacheinander folgen. Eine Visualisierung des Unterschieds zwischen den beiden genannten Arten der Speicherung ist in Abbildung 2.3 zu sehen.

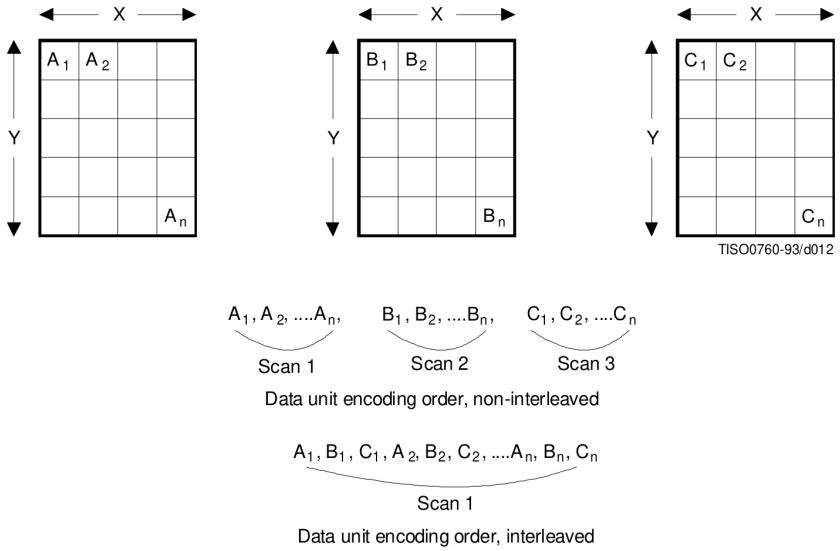


Abbildung 2.3.: Unterschied zwischen interleaved und non-interleaved bei der Speicherung von Datenblöcken. (aus [8], S. 24)

Ein SOS-Segment gibt die Anzahl der Farbkomponenten in diesem Scan an. Danach wird für jede Komponente spezifiziert, welche der zuvor definierten Huffman-Tabellen zur Codierung von Gleich- bzw. Wechselanteilen genutzt werden soll. Im Anschluss daran folgen die Kompressionsdaten. Nach jedem in diesen Daten natürlich vorkommenden `0xFF`-Byte wird ein Null-Byte ergänzt, um die Verwechslung mit den im Datenformat verwendeten Markern zu verhindern.

Auf eine weitere Funktionalität des JFIF-Dateiformates soll hier noch eingegangen werden. In den entropiecodierten Daten des SOS-Segments können RST-Marker vorkommen. Diese Marker signalisieren, dass die folgenden Daten ohne Wissen über die davor liegenden Daten decodiert werden können. Dafür muss allerdings ein Define restart interval (DRI)-Segment vor dem SOS-Segment vorkommen, welches die Abstände dieser Marker definiert. Ohne ein solches Segment sind RST-Marker nicht zulässig. Von Vorteil sind diese Marker, wenn Daten während einer Datenübertragung verloren gehen und nicht erneut übertragen werden können. Ein Beispiel dafür ist die Übertragung von Echtzeitstreams. Außerdem können sie gegebenenfalls Fehler reduzieren, die durch die Prädiktion der DC-Koeffizienten entstehen.

3. Aufbau von Motion JPEG-Daten

Im Vergleich zu einzelnen Bildern besitzen Videos neben Höhe und Breite noch die Dimension Zeit. Für die Kompression von Videodaten kann deshalb neben der Ähnlichkeit von Bildausschnitten ein und desselben Bildes ausgenutzt werden, dass aufeinanderfolgende Bilder meist ebenfalls Ähnlichkeiten besitzen. Das ist zum Beispiel bei einem sich bewegenden Objekt vor stillem Hintergrund oder einem Kameraschwenk der Fall. Diese Ähnlichkeit wird zeitliche Redundanz genannt. In modernen Videokompressionsverfahren wird sie ausgenutzt, um die Informationsmenge durch die Vorhersage von Bildinhalten zu reduzieren. Dafür werden meist umfangreiche Berechnungen durchgeführt, was zu hohen Kompressionsraten führt.

Eine andere Gruppe an Videokompressionsverfahren nutzt die zeitliche Redundanz nicht aus. Stattdessen komprimieren diese Verfahren jedes Bild eines Videos einzeln. Eines dieser Verfahren ist MJPEG, welches dem Namen nach Einzelbilder im JPEG-Format komprimiert. Im Vergleich zu den modernen Videokompressionsverfahren erreichen Verfahren wie MJPEG keine hohen Kompressionsraten. Dafür bieten sie andere Vorteile. Durch die relativ geringe Verarbeitung entstehen kaum Kompressionsfehler. Außerdem sind die Einzelbilder durch die voneinander unabhängige Kompression sehr leicht auszulesen und zu verarbeiten. Beides ist wichtig für den Videoschnitt. Durch die Unabhängigkeit der Bilder voneinander kann an beliebigen Positionen geschnitten werden, ohne dass aufgrund der Kompression ein Qualitätsverlust zu verzeichnen ist. Auch für Echtzeitübertragungen ist diese Art von Videokompression geeignet. Jedes Bild kann einzeln codiert werden, ohne dass es von vorigen oder folgenden Bildern abhängig ist. Dass die dafür durchzuführenden Berechnungen deutlich weniger aufwändig sind als die für moderne Kompressionsverfahren, ist vor allem in Anwendungsfällen mit weniger leistungsfähiger Hardware von Vorteil. Dazu gehören beispielsweise Überwachungskameras, die ihre Bilder in Echtzeit in ein lokales Netzwerk übertragen.

Eine Schwierigkeit beim Verarbeiten von MJPEG-Daten besteht darin, dass kein einheitliches Format für die Speicherung dieser Daten definiert ist. Die eigentlichen Videodaten werden zwar als Strom aufeinanderfolgender JPEG-Bilder gespeichert, allerdings unterscheiden sich die Dateiformate in der Struktur und Definition ihrer Informationsblöcke. Im einfachsten Fall enthält eine Datei mit der Endung `mjpeg` oder `mjpg` nur den Strom aufeinanderfolgender Bilder, ohne Angaben zu beispielsweise Bildwiederholrate, Abspieldauer oder Datenrate zu machen. Andererseits existieren Containerformate wie QuickTime File Format (QTFF) und Audio Video Interleave (AVI), welche zwar die genannten Videodaten beinhalten, diese aber in eine interne Struktur einbetten und daneben noch Datenfelder für aus dem Video abgeleitete Informationen beschreiben. Diese Formate können grundsätzlich von verschiedenen Kompressionsverfahren erzeugte Datenströme enthalten. Deshalb stellt ihr Aufbau eine Strukturierung bereit, in der neben den Daten selbst auch Angaben wie die Art der Daten, das verwendete Kompressionsformat und die Beschreibung des Handlers zum Dekomprimieren der Daten Platz finden.

Mitunter implementieren die genannten Formate auch Felder, die mit dem Begriff „Metadaten“ benannt sind. Diese enthalten beispielsweise einen Urheberrechtshinweis oder den Autor der Datei. Da sich auf ein Video beziehende Informationen wie die Bildwiederholrate oder die Abspieldauer ebenfalls als „Metadaten“ bezeichnet werden können, erfährt dieser Begriff eine Bedeutungsüberladung. Deshalb soll hier darauf hingewiesen werden, dass die Bezeichnung „Metadaten“ im weiteren Verlauf auf die aus dem Video abgeleiteten Informationen und nicht auf dateibezogene Angaben bezieht.

Im Anschluss soll für drei verschiedene Formate erläutert werden, welche Möglichkeiten jeweils zur Speicherung von MJPEG-Daten bestehen. Bei den Formaten handelt es sich um das MJPEG-Format des VLC Media Players, das QTFF und das AVI-Format, welche in dieser Reihenfolge betrachtet werden. Für die beiden Containerformate QTFF und AVI werden jedoch nur die Grundstruktur der Formate sowie relevante Metadaten betrachtet. Eine vollständige Beschreibung dieser beiden Dateiformate ist nicht Gegenstand der nachfolgenden Betrachtungen.

3.1. Das MJPEG-Format des VLC Media Players

Der VLC Media Player [29] ist ein Programm zum Abspielen von Dateien mit multimedialen Inhalten. Dabei werden verschiedene Formate und Medien unterstützt, wie beispielsweise DVDs und Audio-CDs. Auch einige Streaming-Protokolle sind implementiert. Die Software wird von der gemeinnützigen Organisation VideoLAN frei und quelloffen entwickelt und läuft auf allen gängigen Plattformen. Neben dem Abspielen von Mediendaten ist das Programm auch in der Lage, Dateien in verschiedene Formate zu konvertieren. Dazu gehört auch die Konvertierung von Videodaten ins MJPEG-Format.

Zur Speicherung von MJPEG-Daten verwendet der VLC Media Player ein auf den Multipurpose Internet Mail Extensions (MIME) basierendes Format. MIME ist eine standardisierte Form, auf den Inhalt und Aufbau von Daten und Dateien hinzuweisen. Die entsprechenden Informationen befinden sich meist am Beginn der Daten oder Datenabschnitte. Während die Grundstruktur des MIME-Formats in RFC 2045 [6] standardisiert ist, gibt RFC 2046 [7] Aufschluss über die möglichen Medientypen. Zu letztgenannten gehört auch der „multipart“-Typ. Er beschreibt, dass die nachfolgenden Daten aus mehreren Einheiten bestehen. Beispielsweise können die aufeinanderfolgenden JPEG-Bilder eines Videos einen solchen Typ bilden. Die Trennung der Einheiten voneinander erfolgt mit sogenannten „boundaries“. Diese bestehen aus zwei Bindestrichen, auf die eine Zeichenkette von maximal 70 Zeichen folgt. Auf diese Weise wird die Trennung zweier Einheiten gekennzeichnet. Dabei ist die verwendete Zeichenkette innerhalb der gesamten Daten identisch. Danach folgen MIME-übliche Angaben zum Inhalt der nächsten Dateneinheit. Beispiele dafür sind der Typ der Daten, der im Falle eines MJPEG-Videos `image/jpeg` lautet, sowie deren Länge. Im direkten Anschluss an diese Informationen beginnen die eigentlichen Daten der Einheit. Auf diese Weise können mehrere MIME-Einheiten aufeinander folgen, jedoch immer durch die „boundaries“ abgegrenzt. Die „multipart“-Einheit muss an ihrem Ende noch einmal mit der Zeichenkette zur Trennung der Einheiten abgeschlossen werden. Jedoch folgen auf die Kennzeichnung noch einmal zwei Bindestriche.

Bei dem vom VLC Media Player verwendeten MIME-Format werden die Kennzeichnung als „multipart“-Typ sowie die abschließende „boundary“ weggelassen. So kann die

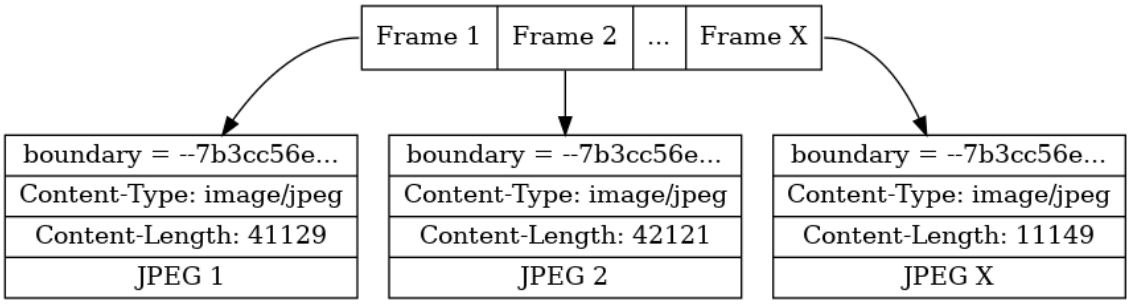


Abbildung 3.1.: Schematische Darstellung einer vom VLC Media Player konvertierten MJPEG-Datei.

Datenmenge etwas reduziert werden. Jedoch ist das Format weiterhin als MIME erkennbar. Ein Beispiel für eine vom VLC Media Player konvertierte MJPEG-Datei ist in Abbildung 3.1 dargestellt. Die zusätzlichen Informationen dieses Datenformates enthalten außerdem keine Daten, die den im JPEG-Format verwendeten Markern entsprechen. Damit können vom VLC Media Player erzeugte MJPEG-Dateien als aufeinanderfolgende JPEG-Bilder gelesen werden. Denn die MIME-Informationen zwischen den Bildern können ohne Weiteres übersprungen werden.

3.2. Motion JPEG im QuickTime File Format

Das QTFF ist ein von Apple entwickeltes proprietäres Containerformat. Eine Dokumentation des Formates existiert auf den Webseiten der Apple Developer [1]. Diese dient allerdings ausschließlich informationellen Zwecken und stellt keine für die Nutzung des Dateiformates erforderliche Lizenz bereit.

Aufgebaut ist das QTFF aus Datenblöcken, die Atoms genannt werden. Diese können ineinander verschachtelt werden und ergeben auf diese Weise eine QuickTime-Videodatei. Neben einem Datenfeld für seine Größe enthält jedes Atom ein aus vier ASCII-Zeichen bestehendes Typfeld, mit welchem es identifiziert werden kann. Im Anschluss an diese beiden Felder folgen die typabhängigen Daten. Deren Länge und Aufbau unterscheidet sich je nach Atom-Typ.

Eine QuickTime-Videodatei besteht immer aus mindestens zwei verschiedenen Atoms. Der Datenblock vom Typ „File type“ wird mit den Zeichen `'ftyp'` gekennzeichnet. Er enthält Informationen, welche das Dateiformat als QTFF identifizieren und die Unterscheidung zu anderen Dateiformaten ermöglicht. Das kann vor allem bei Dateien nützlich sein, deren Dateinamensendung nicht mit den tatsächlichen Inhalt übereinstimmt. Auf die Informationen zum Dateityp folgen die Videodaten im „Movie data atom“. Es ist durch das Typfeld `'mdat'` identifizierbar. Neben den Bewegtbilddaten sind in diesem Datenblock auch Audiodaten oder die Untertitel zu finden.

Optional auf der obersten Gliederungsebene der Datei ist das „Movie atom“ mit der Kennzeichnung `'moov'`. Ist es in einer Datei vorhanden, muss es sich zwischen den beiden genannten Atoms befinden. Dieser Datenblock speichert Metadaten zu den im „Movie data atom“ enthaltenen Daten und deren Struktur. Deshalb ist seine Definition ungeachtet seiner fakultativen Natur sinnvoll. Weitere Atoms sind auf der obersten Strukturebene im QTFF nicht erlaubt. Alle konkreteren Informationen zum enthaltenen

Video werden in speziellen Atoms als Substrukturen der bisher vorgestellten Datenblöcke gespeichert.

Nachfolgend soll zunächst auf die Speicherung bestimmter Metadaten im QTFF eingegangen werden. Im Anschluss daran wird eine Übersicht darüber gegeben, in welchen Formaten MJPEG-Daten im QTFF gespeichert werden können.

3.2.1. Metadaten im „Movie atom“

Das „Movie atom“ enthält eine ausgeprägte hierarchische Struktur, in der Metadaten gespeichert werden. Für jede Hierarchieebene gibt es ein spezielles „header atom“, in welchem allgemeine Informationen für die entsprechende Ebene vorgehalten werden. Teilweise werden dabei Daten aus niedrigeren Ebenen akkumuliert. Dazu gehören beispielsweise die Abspieldauer, der Zeitpunkt der letzten Modifikation und die verwendete Zeitskala. Daraus ergibt sich, dass die Angaben höherer Hierarchieebenen nicht auf alle Unterstrukturen zutreffen. Im Zweifelsfall sind deshalb niedrigere Ebenen als Informationsquelle vorzuziehen.

Der hierarchische Aufbau der Metadaten spiegelt die strukturelle Gliederung der audiovisuellen Inhalte in QuickTime-Videodateien wieder. Diese ist in vereinfachter Form in Abbildung 3.2 dargestellt. Das in der Datei enthaltene Video ist in verschiedene Tracks unterteilt. Ein Track enthält beispielsweise einen Video- oder einen Audiostream. Demzufolge besteht ein Track aus einem Medium. Für den Fall, dass es sich bei dem Medium um Bewegtbilder handelt, spezifiziert das QTFF dafür eine Unterstruktur: Das „Video media information atom“. Diese enthält wiederum eine Tabellenstruktur, aus der sich Informationen zum verwendeten Kompressionsformat und zur Bildwiederholrate ermitteln lassen.

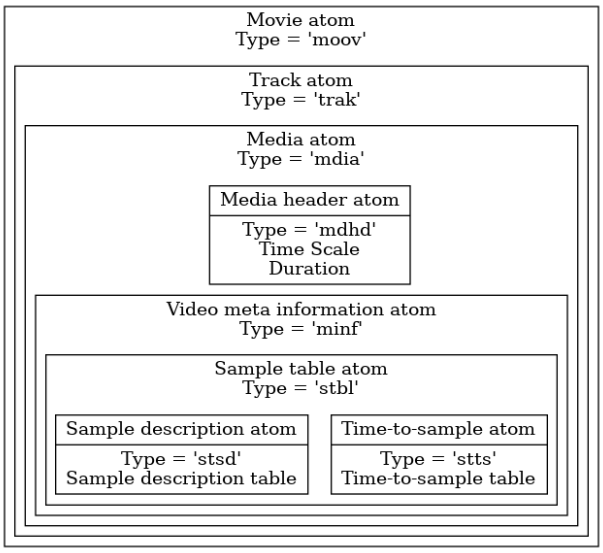


Abbildung 3.2.: Die vereinfachte hierarchische Struktur des „Media atoms“ im QTFF für den Fall von einem Video-Track. Aufgeführt werden nur die hier benötigten Atoms. (verändert nach [1], Abschnitt „Overview of Movie Atoms“)

In dem hier betrachteten Fall des MJPEG-Videos gibt es pro Video nur einen Track. Dieser enthält außerdem nur das Medium Video und keinen Audiostream, da sich das MJPEG-Format nur auf Bewegtbilddaten beschränkt. Im „Media header atom“ befinden sich unter anderem die Informationen zur verwendeten Zeitskala, die in Zeitschritten pro Sekunde gemessen wird. Außerdem ist darin die Dauer des Medienstroms zu finden, die in Einheiten der Zeitskala angegeben wird. Weiter unten in der Hierarchie finden sich Daten zu den Abtastwerten, die als Atoms in der Tabellenstruktur gespeichert werden. Dazu gehören die Beschreibung der Daten, welche unter anderem über das verwendete Kompressionsverfahren Aufschluss geben, und die Zuweisung von Zeitspannen zu einzelnen Videoframes. Letztgenannte findet im „Time-to-sample atom“ statt. Dieses enthält eine Tabelle, in der für jedes Frame des Videostreams die Abspieldauer festgelegt wird. Für konstante Bildwiederholungsraten können die Einträge zusammengefasst werden. Der Gesamtanzahl der Bilder im Video steht dann die Zeit gegenüber, die jedes dieser Bilder beim Abspielen angezeigt wird. Diese Dauer ist in Einheiten der Zeitskala angegeben, die im "Media header atom“ festgelegt wurde. Die Division der Zeitschritte pro Sekunde durch die Anzeigedauer eines Bildes ergibt schließlich die Bildwiederholrate.

3.2.2. MJPEG-Daten im „Movie data atom“

Videodaten werden im QTFF im „Movie data atom“ gespeichert. Für das Auslesen und Verarbeiten dieser Daten sind die Metadaten notwendig, die im „Movie atom“ angegeben werden. Es gibt verschiedene Formate, in denen MJPEG-Daten gespeichert werden können. Als erstes ist die Speicherung der Daten im JPEG-Format möglich. Dabei werden die einzelnen Bilder nacheinander in die Datei geschrieben. Das so entstehende Datenformat entspricht exakt dem Konzept von MJPEG. In den Metadaten wird dieses Format mit `'jpeg'` gekennzeichnet.

Die weiteren Möglichkeiten nutzen im QTFF definierte Motion JPEG-Formate. Motion JPEG Format A beschreibt JPEG-Bilder, die einen eigens definierten APP1-Marker verwenden. Dieser Marker enthält ausschließlich Informationen zu den Offsets bestimmter Felder und Informationen zur Größe des JPEG-Bildes. Damit beschreibt dieses Format valide JPEG-Bilder, die der Definition im JPEG-Standard entsprechen. Das zweite Format mit der Benennung Motion-JPEG Format B wählt einen anderen Ansatz. Anstatt das JPEG-Format mit einem APP-Marker zu erweitern, kapselt es das Bild und entfernt dabei Informationen wie die SOI- und EOI-Marker. Das so entstandene Format ist also nicht valide im Sinne des JPEG-Standards. Aus der Datei extrahierte Bilder im Motion-JPEG Format B können also nicht von üblichen Bildbetrachtungsprogrammen decodiert werden.

3.3. Motion JPEG im AVI Format

Das AVI-Format wurde von der Microsoft Corporation auf der Basis des Resource Interchange File Format (RIFF)-Formats entwickelt. Es ist in der Entwicklerdokumentation für die Softwarebibliothek DirectShow [32] beschrieben. RIFF ist ein Containerformat für Audio- und Videodaten und ist unter anderem auch Grundlage vom Waveform Audio File Format (WAV). Es stellt die Struktureinheiten Chunk und List zur Verfügung. Ein Chunk ist ein Datenblock, äquivalent zu den Atoms beim QTFF. Allerdings beginnt ein

Chunk im Gegensatz zu den Atoms mit dem Typen, auf den die Größe des Blockes und anschließend die typspezifischen Daten folgen. Der Typ wird als four-character code (FOURCC) angegeben, der als Text interpretiert leicht für Menschen lesbar ist. Hierarchische Strukturen werden mit Lists erzeugt. Diese sind mit dem FOURCC `'LIST'` gekennzeichnet. Im Anschluss an diese Zeichen kommen die Größe des Blocks, der Typ der List und wiederum die spezifischen Daten. Eine RIFF-Datei besteht aus einer beliebigen Menge und Anordnung dieser beiden Strukturen, denen ein Header vorangestellt ist. Dieser beginnt mit der FOURCC-Kennzeichnung `'RIFF'`. Auf diesen Code folgen die Dateigröße und der Dateityp.

Im Anschluss soll zunächst auf Grundstruktur von AVI-Dateien und die von dieser bereitgestellten Metadaten eingegangen werden. Darauf folgen Erläuterungen zu den „OpenDML AVI File Format Extensions“, in denen insbesondere die verschiedenen Speichermöglichkeiten von MJPEG-Videos in AVI-Dateien Betrachtung finden.

3.3.1. Die Grundstruktur des von Microsoft entwickelten AVI-Formates

Die Spezifikation des AVI-Formates erweitert das RIFF-Format insofern, als dass weitere Vorgaben bezüglich der Struktur aus Chunks und Lists gemacht werden. Diese sind in Abbildung 3.3 visualisiert. Die Erweiterung gibt vor, dass eine AVI-Datei jeweils eine List-Struktur für Header und Daten enthalten muss. In den Headerinformationen sind Metadaten zu den enthaltenen Streams gespeichert. Die Videodaten befinden sich im mit `'movi'` gekennzeichneten Chunk. Optional ist ein Index-Chunk, welcher Referenzen zur Position der Daten in der Datei enthält. Diese können für den schnellen Zugriff auf bestimmte Bereiche genutzt werden. Da die Struktur im Folgenden nicht weiter von Bedeutung ist, wird sie hier nicht näher erläutert.

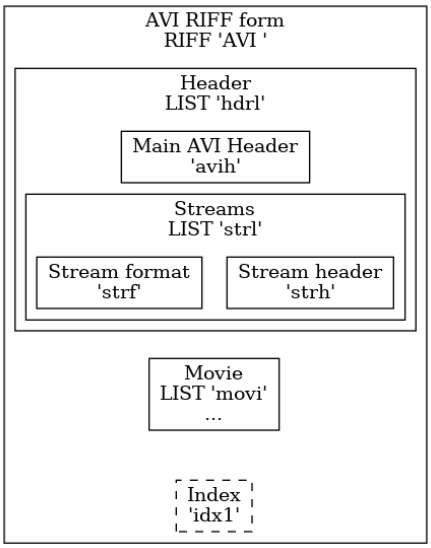


Abbildung 3.3.: Die auf dem RIFF-Format basierende Struktur einer AVI-Datei. Für diese Arbeit nicht relevante Inhalte wurden weggelassen. Dargestellt ist die Struktur für eine Datei mit einem Stream.

Im AVI-Header stehen Angaben zur maximalen Datenrate des Videos, der Anzahl der in der Datei enthaltenen Streams sowie der Höhe und Breite des Videos. Ähnlich wie beim QTFF gilt dabei auch: Der Header fasst Informationen aus den einzelnen Streams zusammen. Deshalb können die Angaben der Streams davon abweichen. Werden genaue Werte benötigt, sind diese deshalb aus den jeweiligen Streamheaders auszulesen.

Auf diese Informationen folgt ein AVI-Stream-Header für jeden in der Datei enthaltenen Stream. Im Fall eines MJPEG-Videos ist beispielsweise nur ein einziger Stream vorhanden. Der Stream-Header macht unter anderem die Angaben „Scale“ und „Rate“. Diese sind im Bezug auf die zeitliche Skala zu sehen. Die Division von „Scale“ durch „Rate“ ergibt die Bildwiederholrate des Streams. Diese Einheit wird auch verwendet, um aus der Angabe „Length“ des Headers die Abspieldauer des Streams zu ermitteln.

Das „Stream format“-Chunk macht Angaben zum Datenformat der komprimierten Bilder. Dazu zählen unter anderem die Höhe und Breite der Bilder. Aber auch die verwendeten Bits pro Pixel, das genutzte Kompressionsformat und die horizontale und vertikale Auflösung der Bilder werden in diesem Datenblock festgehalten.

3.3.2. Die OpenDML AVI File Format Extensions

Das AVI-Dateiformat kann als Containerformat mehrere verschiedene Arten an Daten speichern. Dennoch legt das Format bestimmte Einschränkungen fest. Dazu gehört beispielsweise eine maximale Dateigröße von 4 Gigabyte. Diese resultiert aus dem Datentyp, der für die Größe von Chunks verwendet wird. So können nur 2^{32} Byte adressiert werden. Eine zweite Einschränkung ist die Struktur des optionalen Indexes. Laut der AVI-Spezifikation befindet sich dieser am Ende einer Datei. Zum einen bedeutet das eine große Verschiebung des Lese-Schreibkopfes der Datei am Anfang des Lesevorgangs, da der Index zum effizienten Auslesen der Daten benötigt wird. Zum anderen muss die Position des Indexes verschoben werden, wenn die Daten des Streams erweitert werden. Die wesentlichste Einschränkung besteht jedoch bei den unterstützten Datenformaten. Verschiedene Videoformate und auch JPEG-Bilder können gespeichert werden. Jedoch gibt es kein einheitliches Format für MJPEG-Daten. Stattdessen werden solche Daten als Strom einzelner Bilder gespeichert.

Diese Restriktionen wurden in den „OpenDML AVI File Format Extensions“ [17] von einer eigens zu diesem Zweck ins Leben gerufene Arbeitsgruppe adressiert. Diese Gruppe gehörte allerdings nicht zu Microsoft, sondern zum Unternehmen Matrox Electronic Systems Ltd. Das führte zu einem als „AVI 2.0“ benannten Format, welches erst später von Microsofts Softwarebibliotheken unterstützt wurde. Üblich sind jedoch weiterhin beide AVI-Formate.

In der genannten Erweiterung des Dateiformates wird ein neuer Chunk vom Typ `'hdrx'` vorgestellt. Dieser weist in einer RIFF-Struktur darauf hin, dass sich in der Datei eine weitere Struktur desselben Typs befindet. Auf diese Weise lässt sich eine AVI-Datei unabhängig vom Datentyp der Chunk-Größe beliebig erweitern. Für die Indizierung wird vorgeschlagen, eine hierarchische Indexstruktur zu nutzen. So können sich zwischen den Videodaten Index-Chunks befinden. Diese werden wiederum in einem „Super-Index“ erfasst, der sich wie der vorige Indextyp am Ende der Datei befindet. Auf diese Weise kann die Datei kontinuierlich erweitert werden. Außerdem wird ein schnellerer Zugriff auf die Daten ermöglicht.

Für die Speicherung von MJPEG-Daten wird eine Variante des JPEG-Formates vorgeschlagen. In Abschnitt B.4 der JPEG-Spezifikation [8] wird die Möglichkeit für ein verkürztes Datenformat gegeben. Dazu werden Tabellendefinitionen aus den Daten entfernt und außerhalb der Bilder definiert. Die „OpenDML AVI File Format Extensions“ nutzt diese Variante und gibt vor, dass die zur Kodierung verwendeten Huffman-Tabellen außerhalb der JPEG-Bilder definiert werden müssen. Ansonsten werden die Bilder, wie bei MJPEG üblich, aufeinanderfolgend als Datenstrom gespeichert. Ähnlich wie beim Motion JPEG Format B des QTFFs bedeutet das wiederum, dass keine selbstständigen JPEG-Bilder aus dem Datenstrom extrahiert werden können.

4. JPEG-Transport über RTP

Für die Echtzeitübertragung von Daten wurde von der Internet Society das RTP [23] standardisiert. Weitere Standards definieren die Übertragung bestimmter Datentypen über das RTP. Neben Videokompressionsstandards wie H.264 (vgl. [30]) und HEVC (vgl. [31]) wird so auch das Übertragungsformat für JPEG-Daten standardisiert. Die entsprechenden Ausführungen sind im RFC 2435 [3] zu finden. Außerdem werden die Einschränkungen dieses Standards für die Übertragung von MJPEG-Videos im ersten Abschnitt dieses Kapitels näher betrachtet. Im Vergleich zu den anderen Videokompressionsverfahren hat die Übertragung solcher Videos den Vorteil, dass einzelne Bilder unabhängig voneinander codiert werden. Auf diese Weise können die Bilder sofort komprimiert und übertragen, aber auch vom Empfänger wieder decodiert werden. Diese Unabhängigkeit der Bilder voneinander resultiert in der Echtzeitfähigkeit des Verfahrens.

Neben der Übertragung von in Echtzeit aufgenommenen und codierten Bildern lässt das in RFC 2435 standardisierte Verfahren aber auch zu, dass aufgezeichnete MJPEG-Daten übertragen werden. Für die Speicherung dieser Daten stehen verschiedene Formate zur Verfügung, die in Kapitel 3 schon betrachtet wurden. Da kein einheitlicher Standard für das Datenformat MJPEG existiert und somit für die Kompression der einzelnen Bilder keine Parameter vorgegeben sind, können entsprechende Dateien nicht ohne Weiteres für die Übertragung verwendet werden. Die Nutzung von inkompatiblen Daten für den Transport solcher Videos über RTP würde dazu führen, dass das Medium auf der Seite des Empfängers nicht decodiert werden kann. Deshalb ist eine Offline-Analyse von MJPEG-Videos enthaltenden Dateien sinnvoll. Die Bezeichnung „offline“ bedeutet dabei, dass die Analyse noch vor dem Start des Übertragungsprogrammes stattfindet. Im zweiten Abschnitt wird deshalb eine Implementierung eines solchen Analysewerkzeuges erläutert. Als Konsequenz der vorangegangenen Betrachtung schließt sich die Überlegung an, welche Konvertierungsprogramme valide Daten im Sinne des RFC 2435 [3] erzeugen können. Diese wird den Abschluss dieses Kapitels bilden.

4.1. Anforderungen des RFC 2435

Um die Kompatibilität der MJPEG-Videoübertragung zwischen verschiedenen Systemen zu gewährleisten, trifft der RFC 2435 [3] viele Einschränkungen. Dazu gehören zum einen die Festlegung bestimmter Parameter für die JPEG-Kompression, zum anderen wird ein Format für den Transport der Daten geschaffen. Da der Fokus dieser Arbeit auf der Bereitstellung der benötigten Videodaten liegt, wird nicht weiter auf das Übertragungsformat eingegangen. Nachfolgend sollen die Einschränkungen der Kompressionparameter betrachtet werden. Tabelle 4.1 gibt eine kompakte Übersicht über dieselben. Einige Festlegungen des Übertragungsstandards werden bereits durch das im JPEG-Standard definierte Dateiformat oder durch den JFIF-Standard gefordert. Diese sind in der Übersicht mit dem Symbol $*$ gekennzeichnet.

RFC-Abschnitt	Einschränkung	Feststellung in JFIF-Daten
2.	sequenzieller Baseline-Modus, Huffman-Entropiecodierung	genau ein SOF0- und kein anderer SOF-Marker
2.	ein interleaved Scan	genau ein SOS-Marker
3.1.5., 3.1.6.	max. jeweils 2040 Pixel Höhe und Breite	Y- und X-Werte im SOF-Segment
3.1.8.	$*$ Quantisierungswerte mit 8 oder 16 Bit Genauigkeit	P_q im DQT-Segment
3.1.9.	$*$ Kennzeichnung von `0xff`-Bytes in den Daten durch ein nachfolgendes `0x00`-Byte	-
4.1.	8 Bit Genauigkeit für Abtastwerte	P im SOF-Segment
4.1.	quadratische Pixel	Berechnung aus $Hdensity$ und $Vdensity$ im APP0-Segment
1., 4.1.	$*$ drei Komponenten im YCbCr-Modell	N_f im SOF-Segment
4.1.	$*$ Scan nutzt Komponenten 1 (Y), 2 (Cb), 3 (Cr) in dieser Reihenfolge	-
4.1.	zwei Quantisierungstabellen	T_q-Werte im DQT-Segment
4.1.	Huffmantabellen aus JPEG-Standard [8], Annex K.3	T_c- und T_h-Werte im DHT-Segment
4.1.	Unterabtastung `4:2:2` oder `4:2:0`	H_i- und V_i-Werte im SOF-Segment

Tabelle 4.1.: Einschränkungen des RFC 2435 [3] bezüglich der Kompressionsparameter. Ist die Einschränkung am Beginn mit $*$ gekennzeichnet, wird diese auch vom Datenformat des JPEG- bzw. dem JFIF-Standard gefordert. Die Einhaltung derselben ist demnach schon für die Erstellung eines standardkonformen Datenstromes erforderlich.

Wie in Kapitel 2 schon ausgeführt, stellt das JPEG-Verfahren verschiedene Kompressionsmodi zur Verfügung. Um eine möglichst gute Interoperabilität zu gewährleisten, schränkt der Standard verwendbaren Modi auf den Baseline-Modus ein. Denn dieser Modus muss von allen Decodern implementiert werden (vgl. [27], Seite (S.) 204). Im Baseline-Modus ist außerdem festgelegt, dass die Dekorrelation auf einer diskrete Kosinustransformation basiert und zur Entropiecodierung ein Huffman-Code eingesetzt wird. Des Weiteren unterstützt dieser Modus ausschließlich Abtastwerte von 8 Bit pro Pixel und pro Komponente. Die Verwendung des Baseline-Modus kann im JPEG-Bild nachgeprüft werden, indem dieses nur ein SOF0-Segment enthält.

Eine weitere Festlegung wird bezüglich der Anzahl im Bild enthaltener Scans getroffen. Damit das Bild von Decodern auf verschiedenen Hardware-Konfigurationen dekomprimiert werden kann, darf es nur ein einziges SOS-Segment enthalten (vgl. [3], S. 2). In diesem Scan werden die Daten der Farbkomponenten interleaved gespeichert. Ob ein Bild dieser Forderung genügt, lässt sich demzufolge durch die Anzahl der enthaltenen SOS-Marker bestimmen.

Aus dem SOF0-Segment lassen sich noch weitere Parameter auslesen, die für die Übertragung bestimmte Werte vorweisen müssen. Dazu gehören unter anderem die Höhe und Breite des Bildes. Deren Einschränkung ergibt sich aus der Größe der Datenfelder im JPEG-Header, welcher für die Übertragung der Daten genutzt wird. Bei einer Genauigkeit von 8 Bit pro Wert wird die Dimension jeweils als Vielfaches von 8 Pixeln angegeben. Daraus resultiert eine maximale Höhe und Breite von jeweils 2040 Pixeln. Die Farbunterabtastung lässt sich anhand der Informationen zu den enthaltenen Komponenten bestimmen, die sich im SOF0-Segment befinden. Der Standard zur Übertragung von JPEG-Daten über RTP gibt dabei vor, dass diese entweder `4:2:2` oder `4:2:0` entsprechen muss.

Die Speicherung der Bilddaten im JFIF-Format erfordert bereits, dass die Farbkomponenten des Bildes im YCbCr-Modell vorliegen müssen. Dabei besteht jedoch die Option, nur eine Komponente einzubinden und so ein Graustufenbild zu speichern. Die Bestimmungen des RFC 2435 [3] gehen diesbezüglich noch weiter. So müssen alle drei Komponenten des genannten Farbmodelles vorliegen. Ein Graustufenbild müsste deshalb auch die Chrominanzen integrieren. Die für diese Überprüfung benötigten Informationen liegen ebenfalls im SOF0-Segment vor. Außerdem ist für die Übertragung festgelegt, dass quadratische Pixel verwendet werden müssen. Aus dem APP0-Segment lassen sich dafür die horizontale und vertikale Pixeldichte auslesen. Anschließend kann das Seitenverhältnis der Pixel aus diesen Angaben bestimmt werden.

Der Standard RFC 2435 [3] macht ebenfalls Angaben zu den Quantisierungstabellen. So muss jeweils eine Tabelle für die Luminanz- und Chrominanzwerte vorhanden sein. Deren Anzahl im Bild lässt sich durch die Menge an DQT-Segmenten und den darin enthaltenen Tabellen entnehmen.

Um die Menge an zu übertragenden Daten etwas zu reduzieren, wurde für die Entropiecodierung festgelegt, dass die Huffman-Tabellen aus dem JPEG-Standard Annex K.3 [8] verwendet werden müssen. Dabei handelt es sich um vier Tabellen, die für Luminanz- und Chrominanzwerte jeweils DC- und AC-Komponenten abdecken. Durch diese Festlegung müssen die Huffman-Tabellen nicht mit den Daten übertragen werden.

4.2. Implementierung eines JPEG-Analyseprogramms

Die Aufgabe des Analyseprogramms für MJPEG-Dateien liegt in der Überprüfung der Metadaten der enthaltenen Bilder auf die in Abschnitt 4.1 beschriebenen Anforderungen. Eine grafische Benutzeroberfläche ist dafür nicht notwendig, weshalb die entwickelte Software ausschließlich Text auf die Kommandozeile ausgeben kann.

Wie bereits in Abschnitt 1 erwähnt wurde, soll die Entwicklung des Programmes in derselben Umgebung wie der des Projektes zur Übertragung von MJPEG-Videos stattfinden. Aus diesem Grund erfolgt die Realisierung des Analyseprogrammes in der Programmiersprache Java. Der Entwurf des Programmes nutzt als Kernkomponente die von der `javax.imageio`-Bibliothek bereitgestellten Funktionalität, Metadaten aus JPEG-Bildern auszulesen. Auf diese Weise wird zum einen die Komplexität der Software gering gehalten, während gleichzeitig ihre Fehleranfälligkeit durch die Verwendung einer viel getesteten Bibliothek reduziert wird.

Damit entspricht die im Analyseprogramm zu realisierende Funktionalität der Steuerung des Einlesens von JPEG-Bildern sowie der Verarbeitung der extrahierten Meta-

daten. Der Einlesevorgang kann als separate Subfunktionalität ausgegliedert werden, sodass das Hauptprogramm neue Daten ohne Kenntnis über die Implementierung des Auslesens abrufen kann. Die Analyse der von der Bibliotheksfunktionalität erhaltenen Metadaten beschreibt ebenfalls eine Unteraufgabe. Ihre Grundlage bildet die strukturierte Darstellung der Daten sowie deren Vergleich mit den in Abschnitt 4.1 analysierten Vorgaben. Dazu sind die entsprechenden Angaben in der verwendeten Struktur vorzuhalten und eine Vergleichsoperation zu implementieren.

Um verschiedene Betrachtungsfälle möglich zu machen und die Nutzerschnittstelle des Programmes erweiterbar zu halten, werden verschiedene Kommandozeilenparameter unterstützt. Diese werden in Abschnitt 4.2.1 betrachtet. Die interne Funktionsweise des Programmes, und insbesondere die Extraktion und Aufarbeitung der Metadaten sowie die Überprüfung derselben auf die Vorgaben des RFC 2435 [3], findet in Abschnitt 4.2.2 Erwähnung.

4.2.1. Kommandozeilenparameter

Das Analyseprogramm muss die über die Kommandozeile angegebenen Parameter schon allein deshalb beachten und verarbeiten, um den Namen der zu analysierenden Datei zu erfahren. Indem weitere Parameter angegeben und erkannt werden können, wird die Funktionalität der Software erweitert. Alle gültigen Optionen für das Programm sind in Abbildung 4.1 dargestellt. Es spielt dabei keine Rolle, in welcher Reihenfolge sie angegeben werden. Wird ausschließlich der Name der zu untersuchenden Datei angegeben, erfolgt lediglich die Extraktion und Anzeige der Metadaten des ersten enthaltenen JPEG-Bildes. Dieses Verhalten macht sich die Annahme zunutze, dass alle Bilder einer MJPEG-Datei mit denselben Parametern komprimiert wurden und somit dieselben Metadaten besitzen. Statt einer Laufzeit von ungefähr 750 Millisekunden für die Überprüfung aller Bilder wird so eine deutlich kürzere Ausführungszeit von nur etwa 150 Millisekunden für ein Video mit 2800 Bildern benötigt.

```
usage: MJpegRtpCheck [options] file
  -c,--compliance  check for compliance with RFC 2435
  -f,--full-parse  parse all images of the file, not just one
  -h,--help        print this message
  -v,--version     print version and license info
```

Abbildung 4.1.: Kommandozeilenparameter des JPEG-Dateianalysators

Eine vollständige Überprüfung der Datei ist mit der Option `full-parse` möglich. Damit kann die interne Konsistenz der Daten überprüft werden. Diese erfolgt, indem die Metadaten aller Bilder extrahiert und verglichen werden. Als Referenz für den Vergleich dienen dabei immer die Metadaten des ersten JPEG-Bildes der Datei. Sollten bei der Überprüfung unterschiedliche Kompressionsparameter festgestellt werden, wird der Nutzer darüber informiert. Abschließend erfolgt in jedem Fall die Ausgabe der extrahierten Metadaten des ersten Bildes.

Der eigentliche Nutzen des Programmes liegt jedoch vor allem in der Überprüfung eines MJPEG-Videos auf die Vorgaben des RFC 2435 [3]. Diese wird durchgeführt, wenn

dem Programm die Option `compliance` als Parameter übergeben wird. Dass dieses Verhalten durch die Angabe eines Parameters erzielt wird und dieses nicht das Standardverhalten ist, wurde im Hinblick auf die Erweiterungsmöglichkeit des Programmes entschieden. Auf diese Weise können weitere Funktionalitäten leicht implementiert und über einen weiteren Kommandozeilenparameter zugänglich gemacht werden. Der Aufruf des Programmes mit der Option `compliance` bewirkt, dass die angegebene Datei auf die Eignung für eine JPEG-Übertragung per RTP überprüft wird. Für jede Anforderung erfolgt eine Ausgabe, ob diese erreicht wurde oder nicht. Abschließend stellt eine zusammenfassende Ausgabe fest, ob die überprüfte Datei für eine solche Übertragung geeignet ist. Beispiele für die Ausgaben des Programmes sind im Anhang A zu finden. Die Kombination der Parameter `full-parse` und `compliance` bewirkt das zu erwartende Verhalten: Zunächst werden die Metadaten aller Bilder auf Gleichheit überprüft. Anschließend wird verglichen, ob sie den Anforderungen der Übertragung gemäß RFC 2435 [3] entsprechen.

4.2.2. Interne Funktionsweise

Bevor Bilder aus der angegebenen Datei ausgelesen werden, erfolgt eine Prüfung der Dateinamensendung. Dabei wird diese mit einer Liste an bekannten Endungen verglichen. Jedoch weist die Endung nicht immer zweifelsfrei auf den Inhalt der Datei hin. Ein Beispiel dafür sind Containerformate, die viele verschiedene Mediendatenströme enthalten können. Im Fall des Analyseprogramms muss die Datei eindeutig als MJPEG gekennzeichnet oder ein unterstütztes Containerformat sein. Konkret ist diese Anforderung durch die Endungen `mjpeg`, `mjpg`, `avi` und `mov` erfüllt.

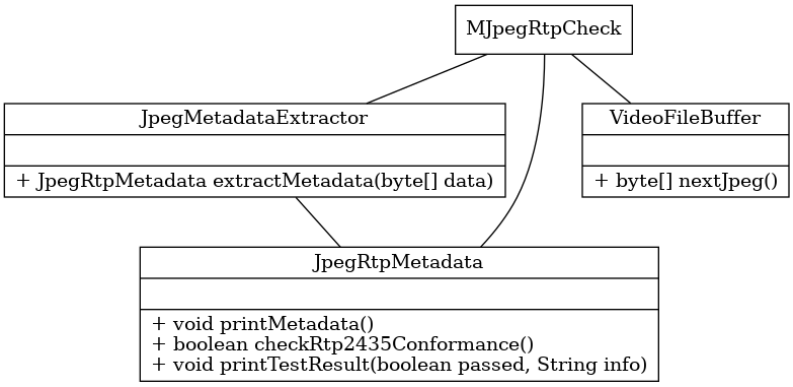


Abbildung 4.2.: Vereinfachte Klassenstruktur des Jpeg-Dateianalysators.

Im Anschluss an die Überprüfung der Dateiendung erfolgt das Auslesen der Bilddaten sowie die Verarbeitung derselben. Eine vereinfachte Klassenstruktur der daran beteiligten Komponenten wird in Abbildung 4.2 gezeigt. In dieser sind auch die Schnittstellen der Klassen dargestellt, welche Aufschluss über die Abfolge der Verarbeitung gibt.

Die Klasse `VideoFileBuffer` stellt mit der Methode `nextJpeg()` die aus der Datei ausgelesenen JPEG-Bilder bereit. Dabei implementiert sie einen Puffer, welcher das Einlesen der Daten effizienter gestaltet. Auf diese Weise werden bei jedem Lesezugriff auf die Datei mehrere Byte auf einmal ausgelesen. So können die Zugriffe auf die Datei möglichst gering und effizient gehalten werden. In den gelesenen Daten wird nach den SOI-

und EOI-Markern des JPEG-Bildes gesucht. Das auf diese Weise gefundene Bild kann an die aufrufende Umgebung zurückgegeben und anschließend weiterverarbeitet werden. Sind jedoch mehr Daten ausgelesen worden als benötigt, werden diese an den Anfang des Puffers verschoben und zwischengespeichert. Der so entstandene freie Speicher im Puffer kann vor der Suche nach dem nächsten Bild wieder gefüllt werden.

Aus dem ausgelesenen Bild werden anschließend die Metadaten extrahiert. Dazu wird der statische Aufruf `JpegMetadataExtractor.extractMetadata()` verwendet, welcher die Funktionalität der Java-Erweiterung `javax.imageio` kapselt. Diese extrahiert Metadaten aus einem bereitgestellten JPEG-Bild. Allerdings werden diese Metadaten in einer mit XML vergleichbaren Struktur dargestellt, welche die Weiterverarbeitung erschweren würde. Aus diesem Grund ist ein weiteres Parsen der Daten erforderlich. Dadurch können sie in die eigene Struktur `JpegRtpMetadata` überführt werden. Die so gegliederten Daten können dann ohne Probleme ausgegeben oder weiterverarbeitet werden.

Je nach geforderter Funktionalität können die extrahierten Metadaten weiter genutzt werden. Eine Ausgabe der Datenstruktur erfolgt mit dem Aufruf `printMetadata()`. Sollen die Metadaten jedoch auf die Anforderungen des RFC 2435 [3] überprüft werden, müssen sie weiter untersucht werden. Die entsprechende Verarbeitung erfolgt mit dem Aufruf der Methode `checkRtp2435Conformance()` der Klasse `JpegRtpMetadata`. Besagte Klasse implementiert die private Methode `rtp2435ConformantData()`, welche ein mit den entsprechenden Parametern initialisiertes Objekt derselben Klasse zurückgibt. Die Realisierung dieser Funktionalität als private Methode ermöglicht den internen Zugriff auf die benötigten Angaben, während sie vor anderen Klassen verborgen bleibt. Damit bleibt die Schnittstelle zur Funktionalität der Klasse `JpegRtpMetadata` klar und übersichtlich.

Für die Überprüfung der extrahierten Metadaten werden diese mit den vom RFC 2435 [3] geforderten verglichen. Dabei erfolgt für jedes Datenfeld eine farbige Ausgabe, ob die Videodaten den Anforderungen entsprechen. Ein Beispiel für einen fehlgeschlagenen Test eines Videos ist in Abbildung A.2 zu sehen. Im Anschluss an den Vergleich fasst eine Konsolenausgabe zusammen, ob das in der Datei enthaltene Video den Forderungen des RFC 2435 [3] entspricht. Damit ist auf einen Blick ersichtlich, ob die Tests bestanden wurden.

4.3. Erstellung von Motion JPEG-Daten

Für das Streaming von JPEG-Videos über RTP werden geeignete Dateien benötigt. Da das MJPEG-Format jedoch nicht sehr üblich ist, müssen die besagten Dateien erst erstellt werden. In den meisten Fällen ist die Konvertierung von einem anderen Videoformat die einfachste Lösung. Viele Programme bieten eine solche Konvertierung an, doch nicht jedes dieser Programme ist geeignet.

Ein anderer Aspekt neben dem Datenformat ist das genutzte Dateiformat. Wie bereits in Kapitel 3 beschrieben, können MJPEG-Daten in verschiedenen Dateiformaten gespeichert werden. Die kleinsten Dateien werden mit dem `mjpeg`-Dateiformat erreicht. Allerdings befinden sich darin keine Metadaten, welche unter anderem auch für das korrekte Abspielen des Videos erforderlich sind. Wird beispielsweise die fehlende Bildwiederholrate aus einem Standardwert abgeleitet, kann das ein etwas schnelleres oder etwas langsameres Abspielen zur Folge haben. In Containerformaten wie AVI oder QTFF kön-

	Video „SdI“	Video „LNdW“
Anzahl Bilder	2817	1800
Größe `mjpeg` [kB]	24968	18072
Größe `avi` [kB]	25036	18120
Größe `mov` [kB]	24976	18080

Tabelle 4.2.: Übersicht der Dateigrößen zweier Videos für verschiedene Formate, angegeben in Kilobyte. Bei den Videos handelt es sich um Bewegtbilder der HTWDD zum Studium der Informatik („SdI“) [28] und zur Langen Nacht der Wissenschaften 2019 („LNdW“) [14].

nen dagegen Metadaten gespeichert werden. Jedoch entstehen dadurch etwas größere Dateien. Der in 4.2 dargestellte Versuch ergab dabei, dass QTFF-Dateien unabhängig von der Länge des Videos nur wenige Kilobyte größer sind als MJPEG-Dateien. AVI-Dateien benötigen am meisten Speicherplatz. Dabei hängt der für AVI-Dateien zusätzlich benötigte Speicherplatz mit der Länge des Videos zusammen. Je mehr Bilder im Video enthalten sind, desto mehr zusätzlicher Speicherplatz wird benötigt. Zusammen mit einem grundsätzlichen Inkrement im Bereich von wenigen Kilobyte ist in AVI-Dateien pro 1000 Bildern im Video ein Anstieg von etwa 20 Kilobyte in der Dateigröße zu erwarten.

Der VLC Media Player ist ein weit verbreitetes Abspielprogramm für Audio- und Videodateien. Als zusätzliche Funktionalität implementiert er eine Möglichkeit zur Konvertierung von Dateien. Unter den unterstützten Dateiformaten befindet sich neben den bereits genannten Containerformaten auch ein MJPEG-Format. Dieses wurde in Abschnitt 3.1 näher beleuchtet. Auch die Skalierung des Videos ist möglich.

Um Videodateien mit der englischsprachigen grafischen Oberfläche des VLC Media Players zu konvertieren, ist die Option `Media -> Convert / Save ...` zu wählen. Es öffnet sich ein neues Fenster, in welchem im Reiter `File` über den Button `Add` die entsprechende Videodatei ausgewählt werden kann. Der Button `Convert / Save` am rechten unteren Rand des Fensters bestätigt die Auswahl und führt zu einem neuen Fenster, in welchem die Parameter für die Konvertierung festgelegt werden können. Neben der Angabe des Pfades der Ausgabedatei muss darin das Konvertierungsprofil ausgewählt werden. In diesem Profil werden Voreinstellungen für jeweils eine bestimmte Konvertierung gespeichert. Da kein vorgefertigtes Profil für die Konvertierung zu MJPEG besteht, muss ein neues angelegt werden. Zur Erstellung des Profiles öffnet sich erneut ein Fenster, in welchem die einzelnen Parameter festgelegt werden können. Im Reiter `Encapsulation` ist die Auswahl `MJPEG` zu treffen. Damit wird das Ausgabeformat der Datei als MJPEG definiert. Der Reiter `Video codec` lässt Angaben zu den Codierungsparametern machen. Für den Unterreiter `Encoding parameters` ist als Codec `M-JPEG` auszuwählen. Die Dimensionen des Videos können im Unterreiter `Resolution` mit der Angabe von `Width` und `Height` festgelegt werden. Der Button `Save` in der rechten unteren Ecke des Fensters speichert das erstellte Profil. Sind sowohl Ausgabedateipfad als auch das gewünschte Profil ausgewählt, kann die Konvertierung über den Button `Start` gestartet werden.

Mithilfe des entwickelten Analyseprogrammes konnte jedoch festgestellt werden, dass die vom VLC Media Player konvertieren MJPEG-Daten nicht den Anforderungen des RFC 2435 [3] entsprechen. Getestet wurde das mit der Version 3.0.9.2 des Programmes.

Einerseits werden für die Entropiecodierung nicht die geforderten Huffmantabellen verwendet. Die vom VLC Media Player verwendeten Tabellen entsprechen nicht denen aus Annex K.3 des JPEG-Standards [8]. Andererseits tritt bei der Skalierung von Videodaten ein Effekt auf, der die Daten für die Übertragung per RTP unbrauchbar macht. Wird bei der Skalierung das ursprüngliche Seitenverhältnis nicht beibehalten, werden die Dimensionen der Pixel verändert. Das Seitenverhältnis der Pixel entspricht dann nicht mehr 1 : 1, sondern wird zur Erhaltung des ursprünglichen Seitenverhältnisses angepasst. Damit ist der VLC Media Player als Konvertierungswerkzeug für diese Anforderungen nicht geeignet. Erwähnt werden soll jedoch noch, dass das Entfernen der Audiospuren kein Problem darstellte.

Ein anderes Programm mit vielfältiger Funktionalität für Audio- und Videodaten ist ffmpeg. Der Schwerpunkt dieses Programmes liegt in der Bearbeitung der erwähnten Multimediadaten. Das Programm wurde in der Version 4.2.4 getestet. Auch ffmpeg unterstützt ein MJPEG-Format. Dabei handelt es sich um die einfachste Form dieses Formates, die Konkatenation von JPEG-Bildern. Auch Containerformate wie AVI und QTFF werden unterstützt. Die Skalierung von Videos ist mit dem Programm problemlos. Wird bei dieser das Seitenverhältnis verändert, werden für das Resultat wieder quadratische Pixel berechnet. Ohne weitere Angaben konvertiert ffmpeg das Video in das gewünschte Format. Wird als Ausgabedatei jedoch ein Containerformat wie AVI oder QTFF angegeben, so wird auch die Audiospur übernommen. Die Option `-an` sorgt deshalb dafür, dass alle Audiospuren des Videos entfernt werden. Bei der Analyse der auf diese Weise konvertierten Dateien mit dem entwickelten Analyseprogramm wurde ein weiterer Aspekt festgestellt. Auch ffmpeg nutzt standardmäßig nicht die geforderten Huffmantabellen. Im Gegensatz zum VLC Media Player lässt sich das aber mit der Option `-huffman default` beheben. Der vollständige Befehl zur Konvertierung von Videodaten lautet demnach:

```
ffmpeg -i <input> -s <width>x<height> -c:v mjpeg
       -huffman default -an <output>
```

Die daraus entstehende Videodatei enthält MJPEG Daten mit den gewünschten Abmessungen, für deren Kompression die Huffman-Tabellen aus Annex K.3 des JPEG-Standards verwendet wurden. Handelt es sich bei dem Ausgabedateiformat um ein Containerformat, werden jegliche Audiospuren des Videos entfernt. Damit ist die durch die Konvertierung erhaltene Datei vollständig kompatibel zur in RFC 2435 [3] definierten Übertragung.

5. Optimierung einer Software zum Streaming mit RTP/RTSP

An der HTWDD gibt es bereits ein Projekt, welches das Streaming mit RTSP und RTP realisiert. Dieses Projekt „RTSP-Streaming“ wird im Modul Internettechnologien II verwendet. Bearbeitet wurde es unter anderem schon für die Implementierung des RTSP-Protokolls [19] und für die Erweiterung mit der Möglichkeit der Fehlerkorrektur unter Nutzung von Forward Error Correction (FEC) [34]. Das Projekt besteht aus einer Server-Client-Architektur. Der Server liest Videodaten aus einer Datei ein und schickt sie mit RTP-Paketen an den Client. Dieser verarbeitet die Pakete und zeigt das Video in einer grafischen Anwendung an.

In der praktischen Verwendung stellte sich allerdings heraus, dass die Ausführung des Projektes auf schwächerer Hardware zu hoher Auslastung des Prozessors führt. Als Grund dafür wurde vor allem das kostspielige Einlesen des Videos erkannt. Vor der eingehenderen Betrachtung der Software wurde ein Ziel für die Optimierungen definiert. Das aus Server und Client bestehende Projekt soll flüssig auf einem Raspberry Pi 3B laufen. Darunter ist vor allem das unterbrechungsfreie Abspielen des Videos in der grafischen Anwendung zu verstehen. Der Raspberry Pi ist ein Einplatinenrechner und einerseits weit verbreitet. Andererseits besitzen die aktuellen Modelle ausreichend Leistung, um komplexere Aufgaben zu lösen. In der Version 3B besitzt der Rechner einen Prozessor der ARMv8-Architektur und 1 Gigabyte Arbeitsspeicher. Aus Gründen der Kompatibilität läuft er allerdings als ARMv7 (vgl. [20]). Als Entwicklungsumgebung wird ein Laptop mit einem Prozessor des Modells Intel Core i5-4210M und 8 Gigabyte Arbeitsspeicher verwendet. Diese dient als erstes Testszenario für die Modifikationen und wird auch zur Bewertung der Effektivität der Änderungen herangezogen. Außerdem werden die Ausführungszeiten einzelner Methoden mit dem Profiler VisualVM in der Version 2.0.7 ermittelt. Auf diese Weise können die für verschiedene Optimierungsversuche ermittelten Zeiten mit den Referenzzeiten des Projektes vor den Optimierungen verglichen und so der Erfolg der Änderung bewertet werden. Die Referenzzeiten sind in Tabelle B.1 dargestellt.

Ein weiterer Grund für die Verbesserung des Projektes liegt in der Kompatibilität mit dem VLC Media Player. Bereits in ihrer Arbeit am „RTSP-Streaming“ hat Zschorlich ermöglicht, das Programm als Alternative für den Server in der bestehenden Projektstruktur einzusetzen (vgl. [34], S. 100 ff.). Dadurch konnte die Implementierung der Protokolle RTP und RTSP im Projekt einseitig validiert werden. Aufgrund der Weiterentwicklung des VLC Media Players war die Kompatibilität des Projektes mit diesem nicht mehr gesichert. Ziel der Optimierung war deshalb die Überprüfung der Kompatibilität mit dem VLC Media Player als Server sowie die Herstellung der Kompatibilität mit der externen Software als Client.

Im Folgenden werden die Maßnahmen erläutert, mit denen die Prozessorauslastung des Projektes reduziert werden soll. Außerdem wird jeweils reflektiert, wie erfolgreich die entsprechende Maßnahme zum Erreichen dieses Ziels war. Zunächst wird dazu die Überarbeitung des Einlesens der Videodaten beschrieben. Im Anschluss daran folgen Ausführungen zur Reduktion der Ausgaben auf der Kommandozeile. Auch die seltenere Aktualisierung der Angaben in der grafischen Oberfläche des Clients finden dabei Erwähnung. Ein weiterer Abschnitt befasst sich mit der Verringerung des Zeitintervalls für den Empfang der RTP-Pakete. Daraufhin wird beschrieben, welchen Einfluss die Ersetzung der Bildbereitstellung mittels `ImageIcon`s durch eine selbst entwickelte Zeichenfunktion auf die Reduktion der Prozessorauslastung des Clients hat. Abschließend wird betrachtet, welche Maßnahmen zur Herstellung der Kompatibilität zwischen dem „RTSP-Streaming“-Projekt und dem VLC Media Player erforderlich sind.

5.1. Reduktion der Lesezugriffe auf die Videodatei

Beim Einlesen des Videos aus der Datei müssen die einzelnen JPEG-Bilder erkannt und getrennt voneinander verarbeitet werden. Der Grund dafür ist, dass in der RTP-Übertragung jedes Bild einzeln in ein oder mehreren Paketen übertragen wird. Demzufolge müssen die aus der Datei gelesenen Daten auf SOI- und EOI-Marker durchsucht werden. Innerhalb dieser Marker befinden sich die Kompressionsdaten eines JPEG-Bildes.

Eine Möglichkeit, diese Suche zu realisieren, ist das byteweise Lesen aus der Datei. So können die Bytes sofort überprüft und JPEG-Marker schnell erkannt werden. Ein weiterer Vorteil dieses Verfahrens besteht darin, dass sich die Leseposition der Datei nach der Erkennung eines Bildes direkt am Ende desselben befindet. Es werden nicht zu viele Daten gelesen und die Suche nach dem nächsten Bild kann anschließend ohne Weiteres fortgesetzt werden.

Dieses byteweise Lesen verwendet auch das „RTSP-Streaming“-Projekt. Der größte Nachteil des Verfahrens ist aber der große rechentechnische Aufwand. Das feingranulare Auslesen der Datei erzeugt sehr viele Anfragen an das Dateisystem. Auch wenn diese mitunter durch das Betriebssystem gebündelt werden können, dauert die Bereitstellung der ausgelesenen Daten mehrere Prozessorzyklen. Während dieser Zeit wartet der Prozessor auf die Beantwortung seiner Anfrage und kann keine anderen Berechnungen durchführen. So wird Prozessorlast erzeugt, die keine Ergebnisse bringt.

Mittlerweile sind die meisten Computer auch in der Lage, größere Mengen an Daten auf einmal aus einer Datei zu lesen. Deshalb können die Lesezugriffe auf die Videodatei reduziert werden, um eine geringere Prozessorlast zu bewirken. Pro Anfrage an das Dateisystem können dann jeweils mehrere Bytes gelesen werden. Die gelesenen Daten können anschließend auf die JPEG-Marker untersucht werden. Besondere Beachtung gilt den Daten, die über den EOI-Marker hinaus gelesen wurden. Diese müssen in einem Puffer zwischengespeichert und für die Suche nach dem nächsten Bild bereitgehalten werden. Auf diese Weise müssen weder die Leseposition der Datei verschoben noch Daten mehrfach eingelesen werden.

Das Einlesen eines Datenblocks von mehreren Bytes und die Verwendung eines Pufferspeichers bewirkte eine deutlich verbesserte Leistung des Projektes auf dem Raspberry Pi. Statt einer stark verlangsamten und verzögerten Wiedergabe des Videos erfolgte dieselbe nun vergleichsweise flüssig. In Tabelle B.2 ist diese Verbesserung nicht eindeu-

tig erkennbar. Die für die ausgewählten Methoden benötigte Ausführungszeit auf dem Raspberry Pi hat sich insgesamt leicht verringert, was ein Anzeichen für den etwas weniger ausgelasteten Prozessor ist. Am deutlichsten ist die Reduktion der Paketverluste. Bei der Ausführung auf dem Laptop zeigten sich lediglich leichte Schwankungen, welche auf die Varianzen durch zeitgleich laufende Prozesse zurückzuführen sind. Der erste Ansatzpunkt der Optimierung stellte sich somit als richtig heraus und brachte eine wesentliche Verbesserung. Da die Videowiedergabe im Bezug auf die Originalgeschwindigkeit jedoch noch leicht verlangsamt lief, bestand weiterer Optimierungsbedarf.

5.2. Reduktion der Programmausgaben

Auch die Ausgaben eines Programmes haben einen wesentlichen Einfluss auf die Leistung desselben. Dazu zählen neben den Anzeigen in einer grafischen Oberfläche auch Textausgaben auf der Kommandozeile. Im Projekt „RTSP-Streaming“ finden die Anzeige des Videos, der statistischen Werte zu empfangenen Paketen, des Abspielfortschritts und des Pufferfüllstandes der Videoframes in der grafischen Oberfläche des Clients statt. Der Server besitzt eine solche Benutzeroberfläche für die Steuerung der Übertragungsparameter. Dagegen erfolgen Ausgaben zu gesendeten und empfangenen RTSP-Paketen, fehlenden Medienpaketen sowie der Status der Fehlerkorrektur mittels FEC auf der Kommandozeile. Beispielhafte Kommandozeilenausgaben von Server und Client sind in den Abbildungen 5.1 bzw. 5.2 dargestellt. Auf die gleiche Weise werden serverseitig Headerdaten von RTP-Pakten in Binärform ausgegeben.

```
Send frame: 59 media
Frame size: 14395
Send frame: 60 media
FEC-Encoder ready...
FEC-Header
00000000 00000000 00000000 00111011 00000000 00000000 00001100 01001000
    00000000 00100001
FEC-Level-Header
00000000 00000000 11000000 00000000
FEC-Payload
00000000 00000000 00000000
FEC packet: 13942 13942
Send frame: 60 fec
Frame size: 14569
```

Abbildung 5.1.: Beispiel der Konsolenausgaben des Servers für zwei RTP- und ein dazugehöriges FEC-Paket. Ausgegeben werden neben Paketnummer und -Länge auch die Binärdaten des FEC-Headers.

Bei den statistischen Werten handelt es sich um Angaben zu verlorengegangenen und korrigierten RTP-Paketen. Diese werden im Takt der Bildwiederholrate aktualisiert. Da diese Rate als Konstante codiert ist, werden die Daten bei jeder Ausführung 25 Mal pro Sekunde aktualisiert. Das ist deutlich häufiger, als einzelne Werte von Menschen erkannt werden können. Denn bei einer Wiederholfrequenz von 25 Hertz werden aufeinanderfolgende Einzelbilder als Animation wahrgenommen. Ähnliches gilt auch für sich ändernde

```
---------------- Receiver -----------------------
Got RTP packet with SeqNum # 60 TimeStamp: 180000 ms, of type 26 Size:
    13923
FEC: set list: 29 [59, 60]
---------------- Receiver -----------------------
Got RTP packet with SeqNum # 29 TimeStamp: 180000 ms, of type 127 Size:
     13968
----------------- Play timer --------------------
FEC: get RTP nu: 9
-> Get list of 1 RTPs with TS: 27000
Display TS: 27000 size: 18068
FEC: set media nr: 61
FEC: set sameTimestamps: 183000 [61]
```

Abbildung 5.2.: Beispiel der Konsolenausgaben des Clients für den Empfang eines RTP- und eines FEC-Paketes sowie das Abspielen eines Bildes. Dabei werden für jeden Schritt sehr detaillierte Informationen ausgegeben, zu denen unter anderem die Paketnummer und der Zeitstempel gehören.

Textangaben. Ein weiterer Effekt ist, dass die Aktualisierungen in der grafischen Oberfläche sehr aufwendig sind. Das spiegelt sich in der für die Änderung von Textanzeigen benötigten Ausführungszeit wider. In der Praxis reichen für Texte wie die statistischen Angaben eine Aktualisierung von fünf Mal pro Sekunde aus. Im konkreten Fall des Software-Projektes lässt sich das realisieren, indem die Statistik nur aller fünf aktualisierten Bilder erneuert wird.

Weiteres Optimierungspotential liegt in den Informationen, die auf der Kommandozeile ausgegeben werden. Auch wenn solche Ausgaben schneller bewerkstelligt werden als die in grafischen Oberflächen, benötigen sie doch deutlich mehr Ausführungszeit als normale Berechnungen. Im Projekt werden neben wichtigen und für didaktische Zwecke relevante Angaben wie die RTSP-Kommunikation auch Daten ausgegeben, die ausschließlich für das Debugging der Anwendungen benötigt werden. Dazu zählen unter anderem die Headerdaten von Paketen (siehe Abbildung 5.1) sowie Angaben zu empfangenen Pakten und abgespielten Bildern (siehe Abbildung 5.2). Eine Lösung für die Kategorisierung von Textausgaben ist die Verwendung eines Loggers. Dieser kann Nachrichten von verschiedener Priorität ausgeben. Ein Vorteil ist dabei, dass die gewünschte Priorität der Ausgaben konfigurierbar ist. Sollen also nur Warnungen und Fehler ausgegeben werden, werden alle Nachrichten mit niedrigerer Priorität übersprungen. Auf diese Weise können alle bisher bestehenden Ausgaben beibehalten werden. Bei der Überarbeitung des Ausgabemechanismus wird ihnen jeweils eine Priorität zugeordnet. Außerdem wird am Beginn der Anwendungen festgelegt, wie detailliert die Ausgaben erfolgen sollen. Standardmäßig ist so eine geringe Menge an Ausgaben konfiguriert, welche Angaben zum Verkehr der RTSP-Pakete, zu fehlenden Paketen und zur Korrigierbarkeit fehlender Pakete umfasst. Soll jedoch eine Anwendung debuggt werden, können auch alle weiteren Daten ausgegeben werden. Eine weitere Modifikation ist notwendig, um die Ausgaben in derselben Formatierung beizubehalten. Die voreingestellte Formatierung des Loggers schließt den Zeitpunkt und die Priorität der Ausgabe mit ein. Um das zu umgehen, muss der Logger in der Anwendung mit der entsprechenden Formatierungseinstellung konfiguriert werden.

Die Verwendung des Loggers inklusive der Prioritätseinstellung stellte sich als deutliche Verbesserung heraus. Auf diese Weise war eine flüssige Wiedergabe des über RTP empfangenen Videos möglich. Sichtbar wird das in Tabelle B.3, welche im Vergleich zu B.1 deutlich verringerte Ausführungszeiten zeigt. Die Reduktion der Kommandozeilenausgaben zeigt sich dabei auch darin, dass die für diese Ausgaben genutzte Methode `println()` nicht mehr verzeichnet wird. Ursächlich dafür ist die Verringerung dieser Ausgaben auf ein Minimum, wodurch die dafür benötigte Ausführungszeit vernachlässigbar wird. Die Erhöhung der Zeit für die Methode `receive()` lässt sich damit erklären, dass als Standardverhalten auf weitere Pakete gewartet wird, sollten in diesem Moment keine weiteren Berechnungen erforderlich sein. Auf dem Raspberry Pi zeigte sich allerdings auch eine Verzögerung beim Beginn der Wiedergabe, welche mit Paketverlusten im Wert von etwa zwei Sekunden Videodaten verbunden ist. Ein weiterer Paketverlust von etwa einer halben Sekunde an Videodaten war kurz nach der ersten Verzögerung festzustellen. Das restliche Video konnte allerdings ohne weitere Verluste abgespielt werden. Im Versuch zeigte sich, dass weder die anfängliche Verzögerung noch die Paketverluste auftraten, wenn in ein und derselben Ausführung des Projektes zuvor schon ein anderes Video abgespielt wurde. Diese Beobachtung legt nahe, dass es sich bei der Ursache der genannten Probleme um die Initialisierung von Strukturen handelt, welche am Beginn der ersten Übertragung stattfinden. Abschließend sei hier noch erwähnt, dass es in der Entwicklungsumgebung des Laptops keine Probleme nach dieser Optimierung gab.

5.3. Reduktion des Zeitintervalls für den Empfang der RTP-Pakete

Um den anfänglichen Paketverlusten entgegenzuwirken, wurde die Implementierung des RTSP-Projektes weiter analysiert. Dabei fiel eine Diskrepanz in den Sende- und Empfangsraten der RTP-Pakete bei Server und Client auf. Der Server sendet die RTP-Pakete in regelmäßigen Abständen, sodass ungefähr eine Rate von 25 Bildern in der Sekunde entsteht. Nach jedem auf diese Weise gesendeten Paket wird außerdem überprüft, ob ein FEC-Paket zum Senden bereit ist. Diese dienen der Korrektur von verloren gegangenen Paketen beim Empfänger. Ist ein solches Korrekturpaket vorhanden, wird dieses gleich im Anschluss an das vorhergehende Medienpaket verschickt. Damit ist es trotz der festen Intervalle beim Senden der RTP-Pakete möglich, dass im Client in kurzer Zeit zwei unmittelbar aufeinanderfolgende Pakete empfangen werden müssen. Für den Client entsteht daraus die Notwendigkeit, häufiger als die Senderate im Server auf ankommenden Pakete zu überprüfen. Ein weiterer Grund für die häufige Abfragen neuer Pakete beim Empfänger ist die Kompatibilität zu anderen RTSP-Servern, wie beispielsweise dem VLC Media Player. Diese senden im Gegensatz zum Server des „RTSP-Streaming“-Projektes mehrere Pakete pro Videoframe. Ein Frame des Videos wird von diesen Sendern demnach auf mehrere RTP-Pakete aufgeteilt, um die Größe der einzelnen Pakete zu reduzieren. Es entsteht damit für den Client die Erfordernis, entweder häufiger empfangene Datenpakete abzufragen oder bei unveränderter Abrufrate jeweils mehrere Pakete pro Aufruf zu verarbeiten. Im vorliegenden Projekt war die erste Möglichkeit der häufigeren Abfragen umgesetzt.

Dementsprechend ergab sich die Überlegung, ob der Paketverlust am Beginn der Übertragung durch eine andere Sende- und Empfangsstrategie verringert oder verhindert werden kann. Deshalb wurde die folgende Strategie entworfen und versuchsweise umgesetzt. Im Client sollte die Abrufrate auf das Doppelte der Bildwiederholrate des Videos verringert werden. Damit die FEC-Pakete des Servers auch weiterhin empfangen werden können, musste dessen Sendeverhalten angepasst werden. Das wurde durch eine Verdopplung der Senderate erreicht. So war das alternierende Verschicken von Medien- und Korrekturpaketen möglich. Ist für das Sendefenster eines FEC-Paketes kein solches bereit zum Versenden, wird der entsprechende Durchgang übersprungen. Für die Medienpakete wird damit trotz der Alternation eine effektive Senderate von 25 Paketen pro Sekunde erreicht.

In den praktischen Versuchen zeigte sich sowohl für den Laptop als auch für den Raspberry Pi eine Reduktion der Ausführungszeit des Paketabrufes im Client auf etwa ein Zehntel des Wertes vor dieser Anpassung. Erwartungsgemäß stieg die Prozessorauslastung des Servers leicht. Im Vergleich zur Reduktion derselben im Client und zur Gesamtleistung ist das allerdings vernachlässigbar. Weitaus wichtiger ist die Entwicklung im Bezug auf die Verzögerung des Abspielvorgangs und die Paketverluste. Auch wenn die Prozessorauslastung über die gesamte Ausführungszeit im Mittel gesunken ist, wurde für den Raspberry Pi festgestellt, dass sich sowohl die Verzögerung als auch die Paketverluste am Beginn der Übertragung verstärkt haben. Zu sehen ist das an der Anzahl verlorener Pakete in Tabelle B.4. Für den Client in Verbindung mit dem Server des Projektes konnten die Ausführungszeiten stark reduziert werden. Jedoch konnten Übertragungen vom VLC Media Player nicht mehr vollständig empfangen und angezeigt werden. Die experimentelle Änderung der Sende- und Empfangsstrategie der RTP-Pakete zeigte demnach keinen Erfolg. Genau wie bei der Analyse vor dieser Änderung wurde beobachtet, dass diese Probleme nicht bei einem wiederholten Abspielen eines Videos in derselben Ausführung auftreten. In einem weiteren Versuch wurde der Abspielvorgang im Bezug auf den Beginn der Übertragung der RTP-Pakete verzögert. Dabei war festzustellen, dass die Paketverluste direkt mit dem Beginn des Abspielens des Videos zusammenhängen. Als Ursache dafür wurde die Konvertierung der Bilddaten in eine der grafischen Oberfläche eigene Struktur erkannt. Diese trägt die Bezeichnung `ImageIcon` und wird für die Anzeige des Bildes in der grafischen Oberfläche genutzt.

5.4. Verwendung einer eigenen Zeichenfunktion

Alle grafischen Oberflächen des Projektes werden mit dem Framework Java Swing erzeugt. In diesem werden Bilder üblicherweise als `ImageIcon` angezeigt. Diese Struktur stellte sich in der vorigen Betrachtung als sehr kostspielig heraus. Besonders die Konvertierung eines Bildes in die Struktur der grafischen Oberfläche besaß im Vergleich zu den anderen Verarbeitungsschritten der Bilddaten eine hohe Ausführungszeit. Deshalb wurde die Visualisierung als `ImageIcon` durch eine eigene Funktion ersetzt.

Statt der Darstellung des konvertierten Bildes wird ein `JPanel` dargestellt. Dabei handelt es sich um ein abstraktes Objekt des Frameworks, welches vor allem zur Gliederung grafischer Oberflächen verwendet wird. Wie jedes Element einer solchen Oberfläche besitzt es eine `paint()`-Methode, welche das entsprechende Objekt an die gewünschte Position zeichnet. Diese Methode wird immer dann aufgerufen, wenn sich der Inhalt des

Objektes ändert und dieses deshalb neu gezeichnet werden muss. Für konkrete Instanzen eines solchen Objektes kann die Implementierung der genannten Methode überschrieben werden. In dem hier betrachteten Fall kann sie auf diese Weise genutzt werden, um das aktuelle Frame des Videos zu zeichnen. Dafür müssen beim Aufruf zum Anzeigen des Frames alle 40 Millisekunden nur die Bilddaten als Membervariable zwischengespeichert und das erneute Zeichnen des `JPanels` aufgerufen werden.

Bei der Untersuchung dieser Modifikation zeigte sich eine deutlich reduzierte Prozessorlast in der Entwicklungsumgebung des Laptops, dargestellt in Tabelle B.5. Statt der Initialisierung eines `ImageIcons` ist darin die eigens implementierte `paint()`-Methode verzeichnet. Dagegen hat sich die Leistung auf dem Raspberry Pi verschlechtert. Die Anzeige des Videos war deutlich verzögert und verlangsamt, vergleichbar mit dem Stand vor den hier beschriebenen Änderungen zur Optimierung der Software. Außerdem war eine deutlich erhöhte Zahl an Paketverlusten zu verzeichnen. Zurückzuführen ist das auf die architektonischen Unterschiede der beiden Testrechner. Während im Laptop Hardware zur Unterstützung individueller Zeichenfunktionen eingebaut ist, müssen diese Berechnungen auf dem Raspberry Pi in Software geschehen. Demzufolge ist die Verwendung einer eigenen Zeichenfunktion nicht dafür geeignet, die Leistung des Projektes auf dem Raspberry Pi zu verbessern.

5.5. Herstellung der Kompatibilität mit dem VLC Media Player

Bei der im Projekt implementierten Softwarearchitektur für das RTSP-gesteuerte Streaming von Videos gibt es die beiden Rollen Server und Client. Während der Server Videos einliest und als Pakete versendet, empfängt der Client die Pakete und verarbeitet diese. Dementsprechend gibt es zwei Szenarien, wie die Implementierung des Projektes mithilfe des VLC Media Players validiert werden kann. Zum einen kann das besagte Programm als Server fungieren, dessen Datenübertragung vom Client des „RTSP-Streaming“-Projektes empfangen wird. Zum anderen kann diese Zuständigkeit umgekehrt werden, wobei der Server des Projektes Pakete an den VLC Media Player als Client sendet.

Im Rahmen der Überprüfung der Kompatibilität konnte festgestellt werden, dass der Client keine Implementierungsfehler enthält, die sich auf den Empfang und die Darstellung von RTP- und RTSP-Paketen auswirken. Diese Erkenntnis basiert auf dem Versuch, bei welchem der VLC Media Player als Server eingesetzt und der Client des „RTSP-Streaming“-Projektes verwendet wurde. Bei diesem Versuchsaufbau wurde das Video fehlerfrei übertragen.

Zu einem anderen Ergebnis kam es bei dem Einsatz des VLC Media Players als Client, dem der Server des Projektes Pakete zusendete. Zunächst wurde die RTSP-Verbindung vom Empfänger nicht angenommen. Um die Ursache dafür zu ermitteln, wurde die RTSP-Verbindung zwischen zwei Instanzen des VLC Media Players als Server-Client-Architektur mit den Rückmeldungen des Servers des „RTSP-Streaming“-Projektes verglichen. Dazu wurde die Netzwerkübertragung zweier Instanzen des VLC Media Players mit dem Programm Wireshark aufgezeichnet und analysiert. Dabei stellte sich heraus, dass der VLC Media Player in der Funktion als Client auf die RTSP-Anfrage `SETUP` eine Antwort mit bestimmten Parametern erwartet. Während der Server im

„RTSP-Streaming“ bereits die meisten Parameter in seine Antwort integriert, informiert er nicht über die Ports, welche er für die RTP-Verbindung verwendet. Diese Angabe ist Teil des Feldes `Transport:` und wird mit dem Schlüsselbegriff `server_port` angegeben (vgl. [24], S. 61). Wird diese Information in die Antwort des Servers integriert, akzeptiert der VLC Media Player die RTSP-Verbindung. Es soll hier noch einmal betont werden, dass es sich bei dem verwendeten VLC Media Player um die Version 3.0.9.2 handelt. Das ist insbesondere von Bedeutung, da dieser das RTSP-Protokoll in der ersten Version verwendet. Mittlerweile existiert seit einigen Jahren die Version 2.0 des Protokolles (vgl. [25]). In dieser werden die Schlüsselbegriffe `client_port` und `server_port` durch `src_addr` und `dest_addr` ersetzt. Bei einer zukünftigen Änderung des VLC Media Players auf die neue Version des RTSP ist deshalb eine Überarbeitung der betrachteten Angaben notwendig.

Auch im Anschluss an die Ergänzung der Angaben zum RTP-Port am Server konnte der VLC Media Player die Videoübertragung nicht abspielen. Da in der Logausgabe des Programmes keine Fehler verzeichnet waren, jedoch gemäß der Aufzeichnungen von Wireshark Pakete verschickt wurden, wurde ein weiteres Programm zum Erproben der Übertragung hinzugezogen. Dabei handelte es sich um den LIVE555 Media Server in der Version 0.99, dessen Bibliotheken auch intern vom VLC Media Player für RTSP-Übertragungen verwendet werden. In dessen Debugausgaben wurde die Fehlermeldung generiert, dass die empfangenen Medienpakete zum Zeitpunkt des Empfangs zu alt waren. Als Konsequenz daraus war der Fehler in der Implementierung bei der Zeitangabe der Pakete zu suchen. Eine erneute Betrachtung des RFC 2435 [3] ergab, dass die Zeitangabe von JPEG-Paketen in der Einheit 90000Hz angegeben sein müssen. Außerdem muss in jedem letzten Paket eines Frames das Markerbit im RTP-Header gesetzt werden. Beides war im Server des „RTSP-Streaming“-Projektes nicht der Fall. Die Behebung beider Fehler ermöglichten schließlich die Verbindung und das Abspielen dem Server des Projektes und dem VLC Media Player als Client.

6. Vorbereitung der Verschlüsselung von Mediendaten

Neben der Codierung und echtzeitfähigen Übertragung von Mediendaten spielt auch die Verschlüsselung derselben eine wichtige Rolle. Während eine solche Sicherung bei der Anwendung auf dem eigenen Computer oder in einem lokalen Netzwerk nicht sehr wichtig scheint, ist die Verschlüsselung bei der Übertragung zu Computern außerhalb des eigenen Netzwerkes von umso größerer Bedeutung. Nicht nur werden damit die Daten vor dem Zugriff Dritter geschützt. Auch der Inhalt der Übertragung bzw. das im Video Gezeigte bleibt so vertraulich. Für kommerzielle Dienste ist die Verschlüsselung ebenfalls eine wichtige Technologie. Beispielsweise können Streamingdienste auf diese Weise ausschließlich registrierten Nutzern Zugriff auf ihre meist kostenpflichtigen Dienste geben.

Das bereits in Kapitel 5 vorgestellte Projekt „RTSP-Streaming“ implementiert neben der Steuerung einer RTP-Übertragung mit dem namensgebenden Protokoll auch ein Verfahren zur Fehlerkorrektur verlorengegangener Pakete mittels FEC. Eine Verschlüsselung der übertragenen Daten wurde jedoch noch nicht realisiert. Im Falle einer Übertragung eines Videostreams auf einen entfernten Computer geschieht diese demnach im Klartext, sodass jeder am Transport der Daten beteiligte Rechner das Video ohne Weiteres dekodieren und abspielen kann.

Die Implementierung einer Verschlüsselung im Projekt erhöht dessen Sicherheit bei der Übertragung. Außerdem kann die Verschlüsselung so in dem bestehenden Projekt praktisch erprobt und angewendet werden. Zuletzt besteht dadurch auch die Möglichkeit, die Verschlüsselungen im Zusammenspiel mit der bereits implementierten Fehlerkorrektur zu erproben. Die Nutzung der Verschlüsselung im Projekt soll allerdings wählbar bleiben. So können Nutzer individuell entscheiden, ob unverschlüsselt oder durch ein Verschlüsselungsverfahren geschützt übertragen werden soll.

Nachfolgend sollen zunächst die Anforderungen an eine solche Verschlüsselung unter Echtzeitbedingungen analysiert werden. Als Szenario wird dabei angenommen, dass der Inhalt des übertragenen Videos nicht von Dritten angesehen werden kann. Im Anschluss daran wird das bestehende Projekt analysiert, um eine geeignete Struktur für die Implementierung von Verschlüsselungsverfahren zu entwerfen. Die daraus folgende Restrukturierung wird zum Ende dieses Kapitels erläutert.

6.1. Anforderungen an die Verschlüsselung

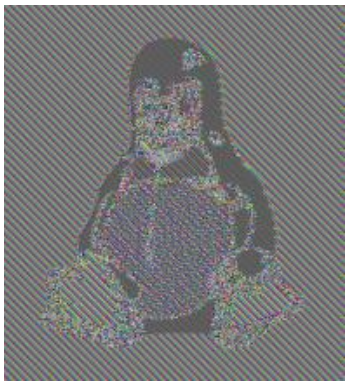
Je nach Anwendungsszenario ergeben sich unterschiedliche Anforderungen an die Verschlüsselung der Daten. Das Ziel ist dabei vor allem der geeignete Einsatz eines Verschlüsselungsverfahrens. Deshalb werden häufig Kryptosysteme genutzt, die schon vielfach getestet wurden und noch nicht gebrochen werden konnten. Die Erstellung eines

eigenen Verschlüsselungsverfahrens ist bei der Anwendung von Verschlüsselung in einer Software nicht das Ziel.

Allen Szenarien, in denen Daten verschlüsselt werden sollen, ist das Erfordernis der kryptografischen Sicherheit gemein. Darunter ist zu verstehen, dass das verwendete Verschlüsselungsverfahren zum Zeitpunkt der Verwendung nicht gebrochen ist und als sicher gilt. Beispielsweise ist der Data Encryption Standard (DES) ein gebrochenes Verfahren, welches im Jahr 2005 zurückgezogen wurde (vgl. [26]) und deshalb nicht mehr verwendet werden sollte.

Eine weitere Anforderung, besonders im Bereich von Medien, ist die Sicherheit im Bezug auf die Wahrnehmung (engl. *perceptive security*). Gemeint ist damit, dass der Inhalt von Bildern oder Videos nach der Verschlüsselung nicht mehr erkennbar sein soll. Ein anschauliches Negativbeispiel dafür ist der Betriebsmodus Electronic Code Book (ECB) für Blockchiffren. Werden unkomprimierte Bilddaten auf diese Weise verschlüsselt, lassen sich immer noch die Konturen derselben erkennen. Ein Beispiel dafür zeigt Abbildung 6.1. Hervorgerufen wird dieser Effekt durch die Tatsache, dass inhaltlich gleiche Blöcke der Originaldaten auf identische Blöcke an verschlüsselten Daten abgebildet werden. Beispielsweise werden alle Bildblöcke des weißen Hintergrundes beim Pinguin Tux in den verschlüsselten Daten durch eine schräge Schraffierung dargestellt. Dieser Effekt trifft in äquivalenter Weise, wenngleich weniger anschaulich, auch auf komprimierte Bild-, Video- oder allgemein andere Daten zu.

(a) Originalbild vom Pinguin Tux, dem Linux-Maskottchen



(b) Mit ECB-Modus verschlüsseltes Originalbild

Abbildung 6.1.: Vergleich des Originalbildes mit dessen ECB-verschlüsselter Variante. (Originalbild von Larry Ewing lewing@isc.tamu.edu mit GIMP erstellt; verschlüsseltes Bild von Wikipedia-Nutzer Lunkwill erstellt auf Basis des Originalbildes, Adresse: https://commons.wikimedia.org/wiki/File:Tux_ecb.jpg (besucht am 02.07.2021)

Im Kontext der Anwendung der Verschlüsselung bei der Übertragung eines Videos per RTSP und RTP erhält die Echtzeitfähigkeit des Verfahrens eine besondere Bedeutung. Um die Ausführungszeit der Verschlüsselung gering zu halten bzw. bei Bedarf zu reduzieren, gibt es zwei Ansatzpunkte (vgl. [16] S. 15). Zum einen kann die Menge der zu verschlüsselnden Daten reduziert werden. Erreicht werden kann das, indem nur wichti-

ge Teile der Daten verschlüsselt werden. Im Fall von Bilddaten können das beispielsweise die zur Decodierung benötigten Tabellen, aber auch die Kompressionswerte der mittleren Helligkeit und der Grundstrukturen sein. Damit können die Mediendaten nicht von Dritten abgespielt oder angezeigt werden und die Verschlüsselung ist schneller fertiggestellt. Die andere Variante besteht in der Verwendung eines im Bezug auf die Rechenkomplexität leichtgewichtigen Verschlüsselungsverfahrens. Eine geringere Komplexität der Berechnungen resultiert in einer schnelleren Ausführungszeit. Das ist der Grund dafür, dass für die Anwendung von Verschlüsselung in solchen Szenarien hauptsächlich symmetrische Verfahren genutzt werden. Denn die meisten symmetrischen Verfahren „können einige hundert- bis tausendmal schneller verschlüsseln als asymmetrische Algorithmen“ ([18], S. 179), da letztgenannte besonders viele Berechnungen benötigen. Während die Vorteile asymmetrischer Verschlüsselungsverfahren beispielsweise beim Schlüsselaustausch liegen, sind sie für die Verschlüsselung von großen Datenmengen in begrenzter Zeit weniger geeignet.

Auch die Beibehaltung des Datenformates ist eine Anforderung mancher Anwendungen. In einigen Fällen ist es erforderlich, dass die Formatinformationen der zu verschlüsselnden Daten beibehalten werden. Das dient vor allem dazu, die korrekte Verarbeitung der Daten zu gewährleisten. Konkret bedeutet das beispielsweise, dass eine auf Paketebene operierende Verschlüsselung die erforderliche Paketstruktur beibehält. Ohne die Einhaltung dieses Formates können Empfänger den Typ des Paketes nicht richtig erkennen und dementsprechend die enthaltenen Daten nicht richtig verarbeiten. Übertragen auf eine auf der Bildebene operierende Verschlüsselung müssen die Formatinformationen des Bildes beibehalten werden, um dieser Anforderung zu entsprechen. Für den Anwendungsfall von JPEG-Bildern sind die Marker zu berücksichtigen. Umgekehrt bedeutet das im Kontext der Decodierung bzw. Entschlüsselung aber auch, dass durch die Verschlüsselung keine Daten erzeugt werden, die fälschlicherweise für Formatinformationen gehalten werden können. Wird das beim Entwurf der Verschlüsselung nicht beachtet, resultiert daraus ein Fehlschlagen der Decodierung. Das kann soweit führen, dass ein verschlüsseltes Medium nicht wieder entschlüsselt werden kann. In einem solchen Fall führen durch die Verschlüsselung erzeugten Formatinformationen zu Fehlentscheidungen im Entschlüsselungsalgorithmus.

Neben den genannten Anforderungen für den Einsatz von Verschlüsselung existierten noch weitere (vgl. [16]). Diese hängen ebenfalls stark vom konkreten Anwendungsszenario ab. Ein Beispiel dafür ist die Eignung des Verfahrens für die Kompression, wenn die Verschlüsselung vor oder während der Kompression stattfinden soll. Auf diese soll hier jedoch nicht näher eingegangen werden.

6.2. Analyse des bestehenden Projektes

Das Projekt „RTSP-Streaming“ ist in seiner Implementierung in zwei Teile getrennt. Zum einen ist ein Server implementiert, welcher die Videodaten einliest und diese überträgt. Zum anderen empfängt ein Client die Datenpakete und verarbeitet sie, um diese in einer grafischen Anwendung anzeigen zu können. Trotz der beiden Teile ist ein Großteil der Funktionalitäten vereint. So übernimmt beispielsweise der `FecHandler` die Verarbeitung aller RTP-Pakete. Dazu zählt sowohl die Bereitstellung von FEC-Paketen im Server als auch die Korrektur fehlender Pakete im Client. Ein anderes Beispiel dafür ist die Klasse

`JpegFrame`, welche sowohl JPEG-Bilder in RTP-Pakete transformiert als auch die umgekehrte Operation bereitstellt.

Bei detaillierterer Betrachtung fällt zunächst auf, dass beide Hauptklassen `Server` und `Client` Programmlogik enthalten, während sie gleichzeitig die grafische Oberfläche definieren und deren Funktionalität implementieren. Dieses Prinzip ist für kleinere Projekte praktikabel. Je umfangreicher allerdings ein Projekt wird, desto wichtiger ist eine klare Trennung der Zuständigkeiten. Diese ist vor allem notwendig, um die Übersichtlichkeit zu wahren. Ein gängiges Vorgehen für größere Projekte ist deshalb, die grafische Darstellung von der Programmlogik zu trennen. Zur Darstellung gehören dabei neben grafischen Oberflächen auch Kommandozeilenoberflächen. Durch die Trennung werden die beiden angesprochenen Komponenten einer Anwendung an verschiedenen Stellen implementiert, was vor allem die einzelnen Dateien übersichtlicher macht. Außerdem entsteht dabei zwangsläufig eine Schnittstelle der Anzeige zur Programmlogik. Wird diese bewusst entworfen und strukturiert, sorgen einzelne Funktionsaufrufe der Logikkomponente für einen durchschaubaren und leicht nachvollziehbaren Programmablauf.

Eine weitere Feststellung lässt sich in Betracht der Klassenstruktur treffen, die in Abbildung 6.2 visualisiert ist. Die beiden Klassen `Server` und `Client` greifen auf die Funktionalität vieler anderer Klassen zu, die jeweils spezielle Aufgaben erfüllen. Diese auffällig flache Hierarchie weist darauf hin, dass ein Großteil der Programmlogik in den beiden Hauptklassen zu finden ist. Um diese Logik gewährleisten zu können, müssen sie auf viele andere Klassen zugreifen. Diese Struktur ist nicht unüblich und in vielen Softwareprojekten zu finden. In dem hier betrachteten Projekt verdeutlicht sie jedoch die mangelnde Struktur in der Implementierung. Sowohl die RTSP-Kommunikation als auch die RTP-Übertragung laufen in den beiden Hauptklassen ab. Damit werden die Implementierungen des RTSP und des RTP vermischt. So existiert für die beiden Protokolle keine Stelle, an der sie eindeutig und getrennt von anderer Funktionalität implementiert sind. Dazu kommt die im vorigen Absatz angesprochene Realisierung der grafischen Oberfläche in den besagten Klassen. Zusammengefasst verringert das die Übersichtlichkeit des Quellcodes in den angesprochenen Klassen deutlich.

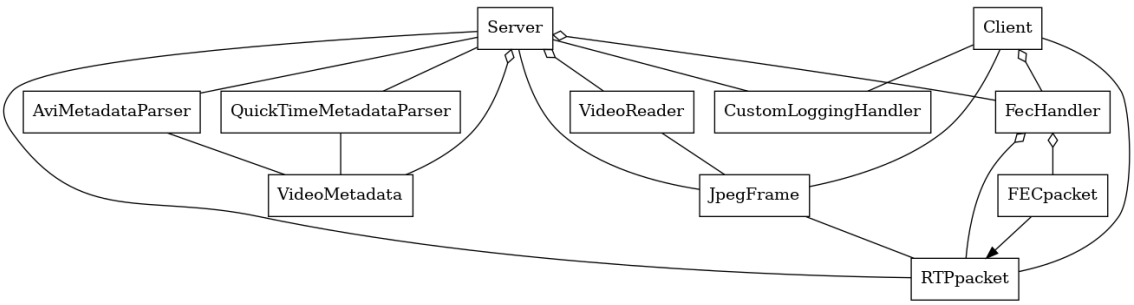


Abbildung 6.2.: Bestehende Klassenstruktur des Projektes „RTSP-Streaming“

Eine weitere strukturelle Schwachstelle findet sich in der Klasse `FecHandler`, welche die Fehlerkorrektur implementiert. Wie bereits erwähnt wird in dieser Klasse die Funktionalität für die Bereitstellung von FEC-Paketen und die Fehlerkorrektur implementiert. Der Server prüft so selbstständig auf das Vorhandensein eines Korrekturpaketes und versendet dieses bei Bedarf. Dagegen wird der funktionelle Umfang der Klasse auf der Empfängerseite erweitert. Statt einer Bereitstellung von Paketen, die verloren gegangen

sind oder bei der Übertragung beschädigt wurden, werden alle RTP-Pakete durch diese Klasse verwaltet. Unabhängig vom Fehlerstatus eines Paketes erfragt der Client so alle Medienpakete vom `FecHandler`. Zurückzuführen ist diese Entwicklungsentscheidung vermutlich auf die schrittweise Erweiterung des Projektes. Für die Implementierung der Fehlerkorrektur per FEC im Projekt erwies sich die Verwaltung der RTP-Pakete auf Empfängerseite als einfach zu realisieren. Der Einfluss dieser Struktur auf die Erweiterbarkeit des Projektes wurde dabei nicht in die Entscheidung einbezogen.

Für die Realisierung einer Verschlüsselung für die Übertragung erweist sich diese Struktur als hinderlich. Auf Paketebene agierende Verschlüsselungen, die als Schnittstelle zwischen Anwendung und Übertragung operieren, verarbeiten alle zu übermittelnden oder empfangenen RTP-Pakete. Eine klare Zuständigkeit für diese Funktionalität und damit verbunden auch eine eindeutige Stelle ihrer Implementierung existiert im bestehenden Projekt nicht. Die Erfüllung dieser Anforderung ist somit im Kontext der bestehenden Struktur schwierig. Ähnlich verhält es sich mit Verschlüsselungen, die auf der Bildebene arbeiten und so zwischen Bildbereitstellung und Paketverarbeitung bzw. für den Empfänger zwischen Paketverarbeitung und Bildanzeige stattfinden müssen. Auch für diese Verarbeitung existiert keine Klasse im Projekt, in deren direkte Zuständigkeit sie fällt. Eine Erweiterung des Projektes durch die Option der Verschlüsselung ohne eine Umstrukturierung im Bereich der Paketverarbeitung würde damit die Übersichtlichkeit des Quellcodes weiter reduzieren. Außerdem würde die Anwendung der Verschlüsselung wiederum in den Klassen `Server` und `Client` stattfinden. Statt einer klaren Schnittstelle zwischen den Anwendungen und der Implementierung des RTP-Protokolls entstünden damit zwei noch deutlicher mit Funktionalität überladene Hauptklassen. Eine Restrukturierung ist deshalb, zumindest für die Verarbeitung von RTP-Paketen, unumgänglich.

6.3. Entwurf einer Neustrukturierung des Projektes

Für die Neustrukturierung des Projektes bedarf es eines ausführlichen Entwurfes. Dabei ist auf eine klare Trennung der Definition grafischer Oberflächen und der Programmlogik zu achten. Außerdem sollten Schnittstellen entworfen werden, welche die Zuständigkeiten der Klassen für bestimmte Aufgabenbereich klar regeln. Nachfolgend soll der Entwurf der besagten Neustrukturierung grob skizziert werden.

Zunächst kann die Implementierung der grafischen Oberflächen von der Funktionalität der Programme getrennt werden. Dies ist mit der Erstellung der Klassen `ServerGui` und `ClientGui` möglich. Beide Klassen stellen ausschließlich die für die Darstellung der Oberflächen benötigten Berechnungen an. Benötigte Informationen, wie beispielsweise das anzuzeigende Bild oder die Empfangsstatistik, sind bei den Logikklassen `Server` und `Client` zu erfragen. Ähnlich verhält es sich mit Nutzereingaben. Diese werden ohne weitere Verarbeitung an die besagten Klassen weitergegeben. Auf diese Weise bleiben `ServerGui` und `ClientGui` ausschließlich für die Darstellung der grafischen Oberflächen verantwortlich.

Die bereits im Projekt bestehenden Klassen `Server` und `Client` bleiben erhalten und stellen die Schnittstelle der grafischen Oberfläche zur Programmfunktionalität dar. Grafisch dargestellt ist das in Abbildung 6.3. Allerdings wird die Funktionalität beider Klassen auf die Ausführung der übergeordneten Logik reduziert. Dazu gehören das Senden

und Empfangen von Netzwerkpaketen und die Delegation der Abarbeitung und Beantwortung von Nachrichten an untergeordnete Klassen. Außerdem bleibt der Server weiterhin dafür verantwortlich, das Auslesen von Videodaten für das Versenden aufzurufen.

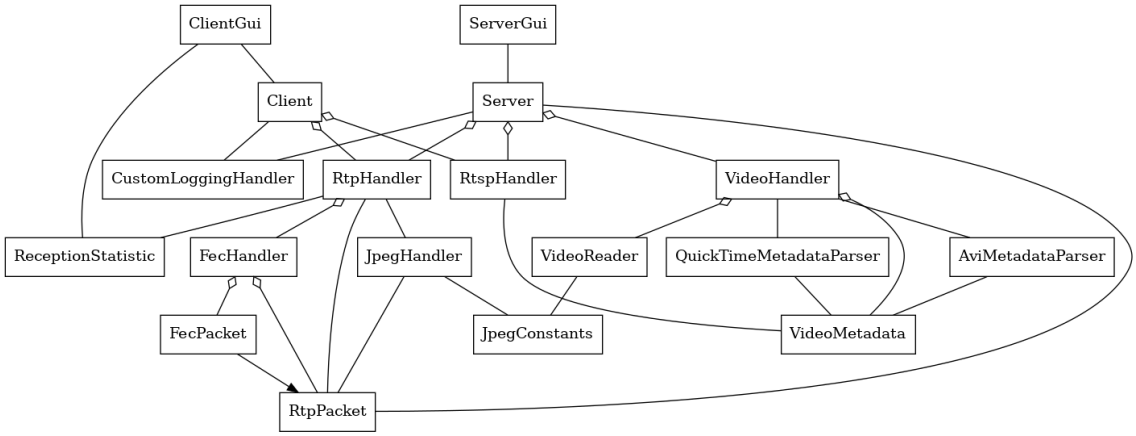


Abbildung 6.3.: Klassenstruktur des Projektes nach einer Neustrukturierung. Der Aufruf der Programme geschieht über die Klassen der grafischen Oberflächen. Getrennt davon sind Server- und Clientfunktionalität, welche wiederum an Unterklassen delegieren.

Für das Auslesen von Bildern aus der geforderten Videodatei sowie das Extrahieren der entsprechenden Metadaten wird die Klasse `VideoHandler` verantwortlich. Diese kapselt jeglichen Zugriff auf Videodateien für den Server. Die Extraktion von Metadaten geschieht dabei über die bereits bestehenden Klassen `AviMetadataParser` und `QuickTimeMetadataParser`. Das Auslesen von Bildern übernimmt weiterhin der `VideoReader`. Dieser benötigt für das Parsen von JPEG-Bildern Informationen über deren Aufbau. In der aktuellen Implementierung sind diese in der Klasse `JpegFrame` enthalten. Diese realisiert jedoch außerdem die Verarbeitung von JPEGs zu RTP-Paketen, worauf der `VideoReader` keinen Zugriff benötigt. Deshalb werden alle JPEG-spezifischen Konstanten in eine neue Klasse `JpegConstants` ausgelagert, die in ihrer Funktionalität den Headerdateien der Programmiersprachen C und C++ ähnlich sind. Die Verarbeitung der JPEG-Bilder übernimmt weiterhin die Klasse `JpegFrame`, die jedoch aufgrund der passenderen Bezeichnung zu `JpegHandler` umbenannt wird.

Da die Klassen `Server` und `Client` zwar Netzwerkpakete empfangen, diese aber nicht verarbeiten sollen, werden die Implementierungen der Protokolle RTP und RTSP ausgelagert. Für die RTSP-Funktionalität entsteht dabei die Klasse `RtspHandler`. Ihre Aufgaben umfassen die Bewerkstelligung des gesamten RTSP-Verkehrs. Für den Client entspricht das dem Versenden von Anfragen sowie dem Parsen der erhaltenen Antworten. Umgekehrt implementiert die Klasse für den Server das Parsen von Anfragen und deren Beantwortung. Auch die entsprechende Behandlung von Nachrichten sowie das Weitergeben der erhaltenen Informationen liegt in der Verantwortung der Klasse `RtspHandler`. Die Auslagerung der RTP-Funktionalität geschieht auf ähnliche Weise. Sie wird im nächsten Abschnitt genauer besprochen.

6.4. Restrukturierung der Verarbeitung von RTP-Paketen

Die Umsetzung der Neustrukturierung des gesamten Projektes entsprechend des Entwurfs des vorigen Abschnittes ist aus verschiedenen Gründen nicht möglich. Zum einen stellt sie selbst eine komplexe und umfangreiche Aufgabe dar. Neben der Implementierung der geplanten Änderungen gehört auch eine detailliertere Ausarbeitung des Entwurfes dazu. Zum anderen dient die Umstrukturierung in diesem Kontext ausschließlich dazu, die Implementierung von Verschlüsselungen für die Übertragung von Mediendaten zu ermöglichen und zu vereinfachen. Deshalb wird an dieser Stelle ausschließlich die strukturelle Verbesserung für die Verarbeitung von RTP-Paketen vorgenommen.

Startpunkt einer verbesserten Klassenstruktur bildet die Auslagerung aller Funktionalitäten des RTP in eine neue Klasse, die als `RtpHandler` bezeichnet wird. Die Eingliederung dieser Klasse im Projekt sowie die von ihr bereitgestellten Schnittstellen sind in Abbildung 6.4 visualisiert. Wie im vorigen Abschnitt bereits festgestellt wurde, findet serverseitig die meiste Programmlogik in Verbindung mit dem RTP in der Klasse `Server` statt. Das Auslesen neuer Videodaten wird dabei vom Server selbst gesteuert. Dasselbe trifft auf die Verarbeitung der einzelnen Bilder zu RTP-Paketen zu, wofür die Klasse `JpegFrame` benötigt wird. Der `FecHandler` stellt dabei ausschließlich die Korrekturpakete bereit, welche nach ihrer Fertigstellung versendet werden.

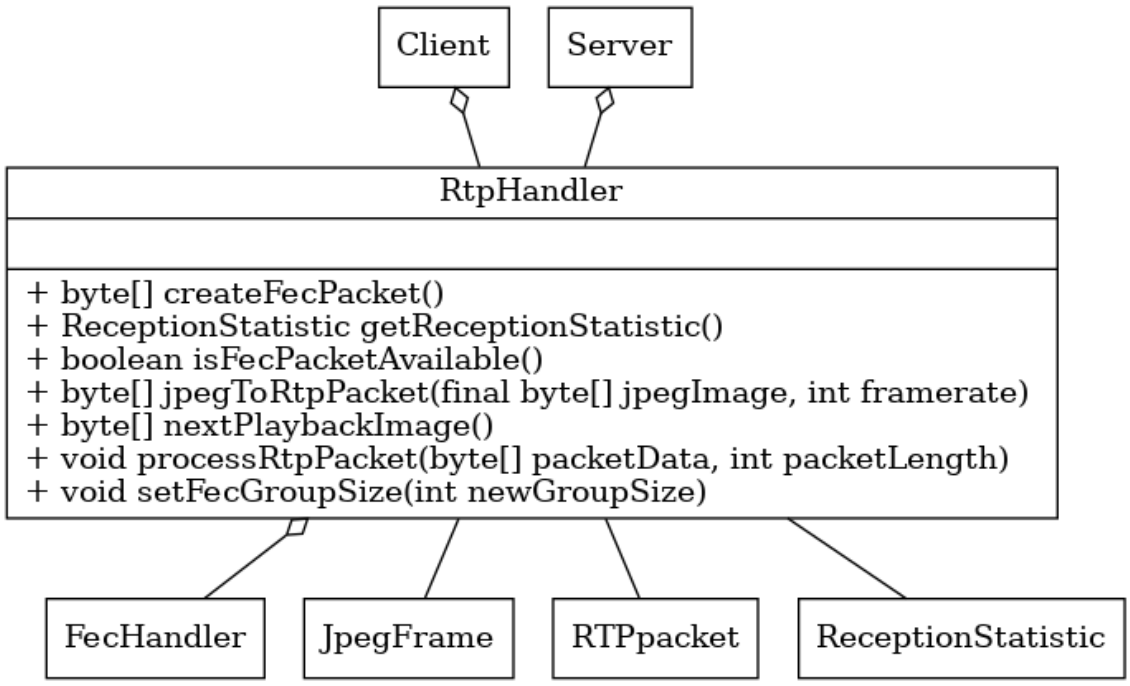


Abbildung 6.4.: Vereinfachte Darstellung der neuen Klassenstruktur nach der Verschiebung der RTP-Funktionalität in eine separate Klasse. Für diese Klasse `RtpHandler` sind außerdem die Methoden dargestellt, welche die Schnittstelle für Server und Client definieren. Neben der klaren Schnittstelle zur Implementierung des RTP-Protokolls ist auch ein vereinfachter Mechanismus zum Abruf der Empfangsstatistik realisiert.

Da ausschließlich die RTP-Verarbeitung umstrukturiert werden soll, ruft der Server auch nach der Einführung des RtpHandlers weiterhin selbst neue Videodaten vom VideoReader ab. Für die weitere Verarbeitung der Bilder werden diese jedoch direkt per jpegToRtpPacket() an den RtpHandler übergeben. Dieser sorgt für die Umformatierung der Daten zu RTP-Paketen. Außerdem wird das erzeugte Paket bei aktivierter FEC-Funktion an den FecHandler weitergegeben, ohne dass der Server davon Kenntnis nimmt. Nach dem Erhalt des angeforderten Paketes und dem Versenden desselben kann der Server mit isFecAvailable() beim RtpHandler ein FEC-Paket erfragen. Gibt der Aufruf den Wert true zurück, holt sich der Server per createFecPacket() das entsprechende Paket und versendet dieses. Alternativ dazu ist auch der direkte Aufruf der FEC-Paket-erstellenden Methode möglich. Diese gibt den Wert null zurück, sollte kein solches Paket verfügbar sein.

Mit dieser neuen Struktur reduziert sich die Zuständigkeit des Servers auf den alleinigen Abruf von RTP- und FEC-Paketen beim neu erstellen Handler. Jegliche Vorverarbeitung der Pakete findet damit direkt im RtpHandler statt. Weitere wichtige Funktionalitäten bleiben jedoch im Server erhalten bzw. werden erst nach Eingaben in der grafischen Oberfläche ausgelöst. Dabei handelt es sich einerseits um die Veränderung der Gruppengröße des FEC-Fehlerschutzes, welche dem Nutzer als Slider in der Oberfläche des Servers zugänglich gemacht ist. Eine Änderung des damit assoziierten Wertes hat den Aufruf der Methode setFecGroupSize() beim RtpHandler zufolge, welche dieser wiederum an den FecHandler weiterleitet. Andererseits wird die Häufigkeit, mit der Videodaten gelesen und Bilder verschickt werden, weiterhin vom Server festgelegt. Dieser erhält die Information entweder aus der Videodatei oder aus einem gesetzten Standardwert und initialisiert damit den in der grafischen Oberfläche implementierten Timer. Ausschließlich die periodisch erzeugten Timerereignisse sorgen für die Verarbeitung von Daten im RtpHandler. Somit stellt die letztgenannte Klasse zwar Funktionalität bereit, deren Aufruf erfolgt aber nur über den steuernden Server.

Die im Bezug auf die Strukturierung der Funktionalität größere Veränderung ist auf der Clientseite zu finden. Dabei gibt es mit der Verarbeitung von RTP-Paketen und dem Abruf von JPEG-Bildern aus dem Speicher zur Anzeige des Videos zwei Schwerpunkte, in denen sich die Zuständigkeiten ändern. Auch mit der Auslagerung von Funktionalität in den RtpHandler steuert der Client den Empfang von Datenpaketen selbst. Allerdings werden die Pakete nun der neu eingeführten Klasse per processRtpPacket() übergeben. Dieser Aufruf gilt auch für FEC-Pakete, da diese der Grundstruktur von RTP-Paketen folgen. Daraufhin ist es die Aufgabe des RtpHandlers, empfangene Pakete anhand ihres Inhaltes zu unterscheiden und entsprechend zu verarbeiten. Der Ablauf dieser Verarbeitung hat sich im Gegensatz zu ihrer Zuständigkeit durch die Restrukturierung nicht geändert. Handelt es sich bei dem empfangenen Paket um ein RTP-Paket mit JPEG-Inhalt, wird dieses in einer Datenstruktur gespeichert. Außerdem wird in Listen festgehalten, welche Medienpakete zusammen ein JPEG-Bild bilden. Für die Verarbeitung von FEC-Paketen ist auch weiterhin der FecHandler zuständig.

Der Abruf von Bildern zur Anzeige des Videos hat sich durch die Einführung der Klasse RtpHandler vereinfacht. Statt eine Liste von RTP-Paketen vom FecHandler zu erfragen, um diese selbst weiterzuverarbeiten, muss der Client ausschließlich die Methode nextPlaybackImage() aufrufen. Mit diesem Aufruf stellt der RtpHandler die benötigten RTP-Pakete als Liste zusammen. Fehlt eines dieser Pakete, wird bei aktivierter

FEC-Funktion der `FecHandler` mit der Korrektur der entsprechenden Pakete beauftragt. Scheitert eine Korrektur, kann kein Bild zurückgegeben werden. Ist die Liste der Medienpakete jedoch vollständig, erstellt der `RtpHandler` daraus mithilfe der Klasse `JpegFrame` ein entsprechendes Bild. Dieses kann dann an den Client zurückgegeben und daraufhin in der grafischen Oberfläche angezeigt werden. Ebenso wie das Senden von Paketen im Server wird der Abruf und die Anzeige von Bildern im Client von einem Timer gesteuert. Dieser ist an die grafische Oberfläche gebunden und wird vom Client initialisiert. Damit stellt die Klasse `RtpHandler` auch clientseitig Funktionalität bereit, deren Aufruf ausschließlich von außen erfolgt.

Auch die Bereitstellung der im Client angezeigten Empfangsstatistik erhält eine bessere und einfachere Schnittstelle. Statt alle Werte einzeln zu erfragen, können diese mit der Methode `getReceptionStatistic()` von der Klasse `RtpHandler` erhalten werden. Dafür wird eine neue Klasse `ReceptionStatistic` bereitgestellt, welche als Datenstruktur dient und keine weitere Funktionalität enthält. Auf diese Weise kann der Abruf der Empfangsstatistik ohne mehrfache Methodenaufrufe erfolgen. Für den Fall der Erweiterung der erfassten Werte ist außerdem keine neue Methode, sondern lediglich die Anpassung der entsprechenden Klasse sowie deren Verarbeitung notwendig.

7. Das Protokoll SRTP

Ein generisches Protokoll für die Verschlüsselung und Authentifikation von Medienpaketen ist das SRTP. Definiert wird es im RFC 3711 [2] als Erweiterung des „RTP Audio/Video Profils“ aus RFC 3551 [22]. Als solche besitzt SRTP auch die eigene Profilbezeichnung „RTP/SAVP“, welche unter anderem per Session Description Protocol (SDP) kommuniziert wird. Damit gibt die Serverantwort auf eine „DESCRIBE“-Anfrage des Empfängers im RTSP ein Indiz für die Verwendung dieses Protokolls. Ein weiterer Effekt der Erweiterung des besagten Medienprofils ist, dass auch bereits definierte Datenformate unterstützt werden. Beispielsweise bleibt der „Payload-Type“ 26 weiter als JPEG identifizierbar. Neben der RTP-Erweiterung im RFC 3711 [2] wird im selben Dokument auch eine äquivalente Erweiterung für das RTP control protocol (RTCP) vorgestellt. Damit wird die Verschlüsselung der Kontrollnachrichten für eine RTP-Übertragung optional verfügbar, während die Authentifikation der besagten Nachrichten zwingend erforderlich wird.

Anwendung findet SRTP beispielsweise in der Technologie Web Real-Time Communication (WebRTC). Diese ist ein weitverbreitetes browserbasiertes Protokollstack für die echtzeitfähige Übertragung von Video-, Audio und anderen Daten. Wie Zschorlich bei der Analyse verschiedener Video-Streaming-Verfahren feststellte, wird die angesprochene Technologie immer häufiger für interaktive Webanwendungen verwendet (vgl. [34], S. 49). Beispiele dafür sind die Verwendung von WebRTC in Videokonferenz-Anwendungen wie „BigBlueButton“ (vgl. [4]) und „Jitsi Meet“ (vgl. [13]).

SRTP wird zwischen der Transportschicht und der RTP nutzenden Anwendung eingesetzt. Alle RTP-Pakete werden in ein äquivalentes SRTP-Paket umgeformt und anschließend transportiert (vgl. [2], S. 5). Das Verhalten auf der Empfängerseite ist dementsprechend umgekehrt. SRTP entspricht dabei einer auf Paketebene agierenden Transportverschlüsselung und Authentifikation. Der Schutz der Mediendaten erfolgt nur indirekt während der Übertragung. Nach derselben sind die in den Paketen enthaltenen Daten nicht weiter geschützt. Dieses Verhalten bedingt jedoch die Interoperabilität mit anderen Erweiterungen für RTP, beispielsweise FEC-Paketen für den Fehlerschutz oder JPEG-Pakten. SRTP behandelt diese unabhängig von ihrem eigentlichen Inhalt und generiert ein entsprechendes verschlüsseltes Paket, welches übertragen wird. Der darüberliegenden Anwendung ist es somit möglich, mit speziellen Pakettypen zu arbeiten, ohne auf eine sichere Übertragung verzichten zu müssen.

Der Kern von SRTP besteht aus der Bereitstellung eines kryptografischen Kontexts, welcher verschiedene Funktionen für die Sicherung von RTP-Paketen bereitstellt. Dieser enthält unter anderem Informationen wie den Verschlüsselungsverfahren mit den dazugehörigen Parametern, den Algorithmus für die Authentifikation sowie Schlüssel für die verwendeten Verfahren. Mithilfe des genannten Kontextes lassen sich Pakete verschlüsseln und authentifizieren. Der organisatorische Aufwand der Schlüsselverwaltung kann dabei durch die Verwendung von „Master-Key“ und „Master-Salt“ verringert werden. Der „Master-Key"dient dabei als Hauptschlüssel, aus welchem alle Sitzungsschlüssel ab-

geleitet werden. Die Sicherheit wird zusätzlich noch vom „Master-Salt“ erhöht, welches in jede dieser Ableitungen einfließt und somit bestimmte Angriffe gegen das Verfahren erschwert. Ein Beispiel dieser Angriffe ist die Offline-Berechnung von Sitzungsschlüsseln und Paketdaten für bestimmte „Master-Keys“. Abgefangene verschlüsselte Pakete könnten so mit den im Voraus berechneten Daten verglichen werden, woraus sich der verwendete Hauptschlüssel ableiten und somit die gesamte Kommunikation mitlesen lassen würde. Das für jede Sitzung neu erzeugte „Master-Salt“ verhindert solche Angriffe, da die Menge an möglichen Kombinationen, die im Voraus berechnet werden müssten, sich so um mehrere Potenzen erhöht.

Mit diesem Kernkonzept wird außerdem deutlich, dass SRTP nicht die Aufgabe der Aushandlung von Sitzungsparametern wie Verschlüsselungsverfahren oder Schlüsseln übernimmt. Diese Aushandlung muss bei der Sitzungsinitialisierung von anderen Protokollen durchgeführt werden. Beispiele dafür sind „Multimedia Internet KEYing“ (RFC 3830) oder die Erweiterungen für SDP und RTSP im RFC 4567 bzw. ausschließlich für SDP in RFC 4568.

Nachfolgend soll zunächst auf den Aufbau von SRTP-Paketen eingegangen werden. Im Anschluss daran werden die drei Hauptbestandteile von SRTP erläutert. Dabei handelt es sich um die Verschlüsselung, die Authentifikation und Integritätssicherung von Nachrichten sowie Algorithmen zur Ableitung von Unterschlüsseln aus dem „Master-Key“. Für jeden dieser Funktionsbestandteile definiert SRTP Standards, die in jedem Fall implementiert werden müssen. Auf diese soll in den Ausführungen besonders eingegangen werden. Außerdem besteht im Protokoll die Möglichkeit, weitere Verfahren aufzunehmen und zu nutzen. Diese ist jedoch nicht Gegenstand der weiteren Betrachtungen.

7.1. Aufbau von SRTP-Paketen

Da das SRTP auf dem Protokoll RTP aufbaut, besitzen die so erzeugten Pakete auch nahezu dasselbe Format. Dieses in in Abbildung 7.1 dargestellt. Die Headerdaten der Pakete sind identisch zu denen in RTP-Paketen und bleiben unberührt. Dazu gehören neben Informationen zur Protokollversion, zum Typ der enthaltenen Daten, zur Paketnummer und zum Zeitstempel unter anderem auch der Indentifikator für die Quelle der Pakete.

An den RTP-Header schließen sich die Paketdaten an. Diese unterliegen vollständig der im Vorfeld definierten Verschlüsselungsoperation. Besonderer Fokus liegt dabei auf der Datenmenge. Alle im SRTP definierten Verschlüsselungen sind so entworfen, dass die Längen der verschlüsselten Daten und der Daten im Klartext identisch sind. Damit ist die Verschlüsselungsoperation auch im Sinne der Paketgröße transparent.

Vergrößert wird ein durch SRTP verarbeitetes RTP-Paket ausschließlich durch das Anhängen des Identifikators für den „Master-Key“ oder die Authentifikationsdaten. Die Übermittlung des Erstgenannten in jedem Paket ist optional. Er dient entweder zur Erneuerung der Schlüssel in einer laufenden Sitzung oder zur Identifikation eines „Master-Keys“, sollten mehrere zur Auswahl stehen. Existiert nur einer dieser Hauptschlüssel und wird dieser zu Beginn der Sitzung ausgehandelt, ist die Übermittlung dieses Identifikators nicht nötig. Anders steht es mit den Authentifikationsdaten. Diese werden mithilfe eines erzeugten Sitzungsschlüssels für das gesamte Paket, mit Ausnahme der angehängt Daten, ermittelt. Die Verwendung einer solchen Authentifikation wird vom SRTP-Protokoll empfohlen. Damit kann ein empfangenes Paket verifiziert und Replay-Angriffen

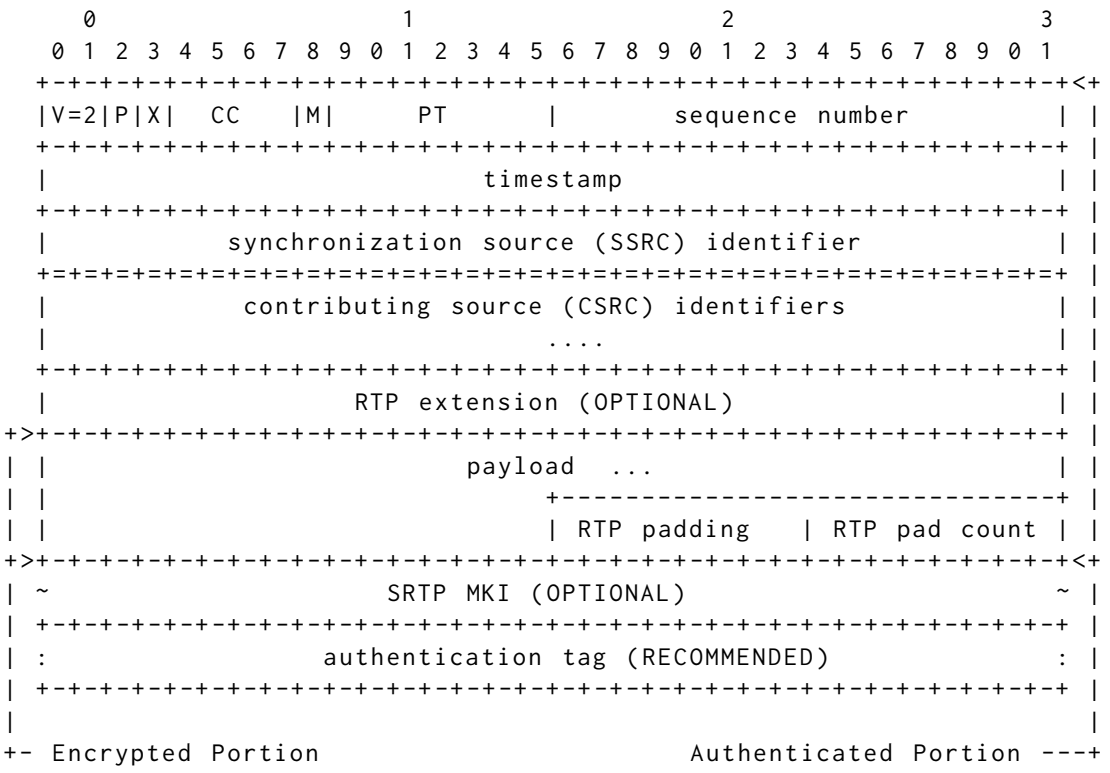


Abbildung 7.1.: Format von SRTP-Paketen. (aus [2], S. 6)

vorgebeugt werden. Als Replay-Angriff ist das Aufzeichnen und später wiederholte Abspielen von Paketen zu verstehen. Diese Art von Angriffen zielt hauptsächlich darauf ab, dass ein Dritter die Identität des Senders einnimmt. Auf diese Weise entstehen verschiedene Angriffsszenarien, wozu auch das Offenlegen von Sitzungsschlüsseln gegenüber Dritten gehört. Replay-Angriffe bei SRTP-Übertragungen zielen im Allgemeinen weniger auf das Kompromittieren von Schlüsseln ab, sondern eher auf das wiederholte Senden bestimmter Daten. Ein Beispiel dafür ist das Mitschneiden der Aufnahmen von Überwachungskameras, welche zu einem späteren Zeitpunkt wieder übertragen werden (vgl. [2], S. 41 f.). Für den Zeitraum der Übertragung dieser aufgezeichneten Daten ist die Überwachung des gewünschten Objektes ausgesetzt.

7.2. Verschlüsselung im SRTP

Die vom SRTP vorgegebenen und zu implementierenden Verschlüsselungsverfahren sind Counter-Modus des AES (AES-CM) und NULL. Dabei bezeichnet die NULL-Chiffre die Verarbeitung eines Paketes ohne die Daten desselben zu verschlüsseln. Sie kann vor allem zu Testzwecken eingesetzt werden. Die Chiffre AES-CM steht für den Counter-Modus des Advanced Encryption Standard (AES) mit einem Schlüssel von 128 Bit. Bei der Verwendung dieses Verfahrens müssen sowohl der Modus als auch das Verfahren selbst von einem anderen Protokoll ausgehandelt werden. Der Grund dafür ist, dass die Aushandlung außerhalb des Rahmens von SRTP ist und das besagte Protokoll deshalb nicht die erforderliche Funktionalität implementiert.

Der AES ist „die heutzutage am meisten genutzte symmetrische Chiffre überhaupt“ ([18], S. 103) und gilt zur Zeit als sicher. Zu den bekanntesten Standards, die AES verwenden, gehören unter anderem das Protokoll Transport Layer Security (TLS) und das Secure-Shell-Protokoll SSH (vgl. [18], S. 103). Als symmetrisches Verfahren ist diese Chiffre, gemessen an asymmetrischen Verfahren, vergleichsweise schnell. Der Counter-Modus bezeichnet einen Betriebsmodus für Blockchiffren, bei welchem für die Verschlüsselung jedes Blockes ein Initialisierungsvektor und ein Zählerwert genutzt werden (vgl. [18], S. 154). Für die Nutzung im SRTP ist der Initialisierungsvektor `IV` wie folgt definiert:

```
IV = (k_s * 2^16) XOR (SSRC * 2^64) XOR (index * 2^16)
```

Dabei bezeichnet `k_s` den für das „Salting“ erzeugte Sitzungsschlüssel. Dieser sorgt dafür, dass keine Kombinationen für Initialisierungsvektoren und verschlüsselte Daten vorberechnet werden können. Mit `SSRC` und `index` gehen außerdem die Identifikation des Senders und der Nummer des aktuellen Paketes mit in die Berechnung ein. Für den Zähler sind die 16 Least Significant Bits (LSBs) des Initialisierungsvektors vorbehalten. Der Wert des Zählers wird bei der Verschlüsselung für jeden Klartextblock inkrementiert. Insbesondere der Zähler verhindert, dass gleiche Eingangsdaten auf dieselben verschlüsselten Daten abgebildet werden. Das Erkennen von Inhalten in den verschlüsselten Daten wie beim ECB-Modus kann auf diese Weise verhindert werden.

Weitere Vorteile des Counter-Modus sind, dass er gut parallelisierbar ist und als Stromchiffre eingesetzt werden kann. Die Nutzung als Stromchiffre kommt besonders dann zur Geltung, wenn ein Klartextblock kleiner ist als die Blockgröße des Verschlüsselungsverfahrens. Bei einer reinen Blockchiffre müsste der Klartextblock durch Padding auf die Blockgröße des Verfahrens erweitert werden. Für die Verschlüsselung mit einer Stromchiffre ist diese Erweiterung nicht erforderlich. Das resultiert in einem Chiffrat mit derselben Größe wie der Klartext, was besonders bei der Übertragung von Daten in Netzwerken von Vorteil ist. Das Chiffrat einer Blockchiffre wäre in einem solchen Fall größer als der ursprüngliche Klartext.

7.3. Der Authentifikationsmechanismus des SRTP

Neben der Verschlüsselung ist SRTP auch in der Lage, Pakete zu authentifizieren. Die Authentifikation erfolgt standardmäßig mit dem Verfahren „HMAC-SHA1“, welches in RFC 2104 [12] definiert wird. Dieses Verfahren muss außerdem durch jede Implementierung des SRTP unterstützt werden. Bei „HMAC-SHA1“ handelt es sich um einen Message Authentication Code (MAC), welcher auf dem Hashverfahren SHA1 basiert. Das allgemeine Funktionsprinzip von MACs ist, dass aus einem Schlüssel und den zu authentifizierenden Daten eine Prüfsumme berechnet wird. Diese Prüfsumme wird zusammen mit den Daten übertragen. Der Empfänger kann anschließend dieselben Berechnungen wie der Sender durchführen. Stimmt die vom Empfänger berechnete Prüfsumme mit der übertragenen überein, ist das Datenpaket authentifiziert.

Die Sicherheit des MAC-Verfahrens basiert vor allem auf der Geheimhaltung des Schlüssels sowie einem sicheren Berechnungsalgorithmus. Ist der Schlüssel nur dem Sender und dem Empfänger bekannt, kann ein authentifiziertes Paket nur vom Sender stammen. Dasselbe gilt für Pakete, die in umgekehrter Richtung versendet werden. Der Berechnungsalgorithmus für die Prüfsumme ist nur dann sicher, wenn dieselbe Prüfsumme

nicht aus einer anderen Schlüssel-Daten-Kombination erzeugt werden kann. Hashalgorithmen bringen solche Eigenschaften mit und werden deshalb häufig für MACs genutzt, woraus die Bezeichnung „HMAC“ entsteht. Zur Zeit gilt unter anderem SHA1 als sicheres Hashverfahren. Mit diesen Voraussetzungen definiert RFC 2104 [12] die Berechnung der Prüfsumme `c` für das „HMAC-SHA1“-Verfahren wie folgt:

```
c = SHA1(key XOR opad, SHA1(key XOR ipad, packet))
```

Zunächst wird der Schlüssel mit einem ersten Datenfeld `ipad` per XOR verknüpft und mit den Daten des Paketes konkateniert. Das Datenfeld `ipad` besteht dabei aus einer Aneinanderreihung des Bytes `0x36` von der Länge der Blockgröße des Hashverfahrens. Anschließend wird eine erste Prüfsumme aus den konkatenierten Daten berechnet. Diese wird an die Verknüpfung des Schlüssels mit einem zweiten Datenfeld `opad` angehängt. Das zweite Datenfeld `opad` besitzt dieselbe Länge wie `ipad`, besteht aber aus der Aneinanderreihung des Bytes `0x5C`. Aus den so gewonnenen Daten wird abschließend die Prüfsumme berechnet, die als MAC verwendet wird.

Neben der Authentizität eines Pakets wird durch MACs auch die Integrität desselben gesichert. Würde der Inhalt eines Paketes bei der Übertragung verändert werden, könnte der Empfänger die Prüfsumme des Paketes nicht selbst berechnen. Anhand der mangelnden Übereinstimmung der übertragenen und der berechneten Prüfsumme würde er feststellen, dass der Inhalt des Paketes verändert wurde.

7.4. Die Ableitung von Schlüsseln aus dem „Master-Key“

Das SRTP ermöglicht die Nutzung eines „Master-Keys“, sodass nur dieser in der Sitzungsinitialisierung ausgetauscht werden muss. Dieser Hauptschlüssel dient ausschließlich zur Berechnung weiterer Schlüssel, welche in den verschiedenen Funktionsbestandteilen zum Einsatz kommen. Auf diese Weise werden die für die Verschlüsselung, für die Authentifikation und für das „Salting“ benötigten Schlüssel abgeleitet. Bei der Verwendung eines „Master-Keys“ muss diese Ableitung mindestens einmal zur Initialisierung aller Schlüssel stattfinden. Weitere Neuberechnungen der Schlüssel sind von der im kryptografischen Kontext gesetzten `key_derivation_rate` abhängig (vgl. [2], S. 26).

Auch für die Ableitung von Schlüsseln definiert SRTP ein Standardverfahren. Dieses basiert ebenso wie die Verschlüsselung auf dem Counter-Modus des AES. Das hat den Vorteil, dass nur wenige kryptografische Algorithmen implementiert werden müssen. Außerdem werden die Schlüssel standardmäßig nur einmal pro Sitzung berechnet (vgl. [2], S. 27), was den Berechnungsaufwand derselben minimiert. Weiterhin wird für jede Operation ein Label in der Größe eines Bytes vergeben, zu sehen in Tabelle 7.1. So kann sichergestellt werden, dass sich die erzeugten Schlüssel für Verschlüsselung und Authentifikation unterscheiden. Um die Sicherheit der Schlüssel weiter zu erhöhen, kann das „Master-Salt“ verwendet werden. Damit werden die bereits genannten Angriffe der Offline-Berechnung von Schlüssel-Daten-Paaren erschwert. Der RFC 3711 legt fest, dass das „Master-Salt“ zufällig sein muss (vgl. [2], S. 9).

Aus „Master-Key“, Label und „Master-Salt“ können anschließend die Schlüssel berechnet werden. Das geschieht mit der folgenden Vorschrift:

```
key = AES-CM(master-key, ((label « 48) XOR master_salt) « 16)
```

Je nach benötigter Schlüssellänge wird der berechnete `key` nach der entsprechenden Menge an Bytes abgeschnitten, um den zu verwendenden Schlüssel zu erhalten. Die Standardwerte die benötigten Variablen sind in Tabelle 7.1 zu finden.

Verwendung Schlüssel	**Label**	**Benötigte Schlüssellänge [Bit]**
Verschlüsselung	`0x00`	128
Authentifikation	`0x01`	160
„Salting“	`0x02`	112

Tabelle 7.1.: Standardwerte für die Ableitung von Schlüsseln

7.5. Implementierung des SRTP-Protokolls zur Verschlüsselung

Mit der in Abschnitt 6.4 besprochenen Erstellung der Klasse `RtpHandler` und der damit verbundenen Restrukturierung ist es möglich, das SRTP-Protokoll auf einfache Weise zu implementieren. Jegliche zu dem Protokoll gehörende Funktionalität wird in der neuen Klasse `SrtpHandler` implementiert. Gleichzeitig stellt die Klasse Methoden als Schnittstelle für den `RtpHandler` bereit, welche die Verarbeitung von SRTP-Paketen ermöglichen. Diese werden in Abbildung 7.2 gezeigt. Da der Fokus dieser Arbeit allerdings nur auf der Verschlüsselung von Datenpaketen liegt, wird auf die Implementierung der Authentifikation im Rahmen des SRTP verzichtet.

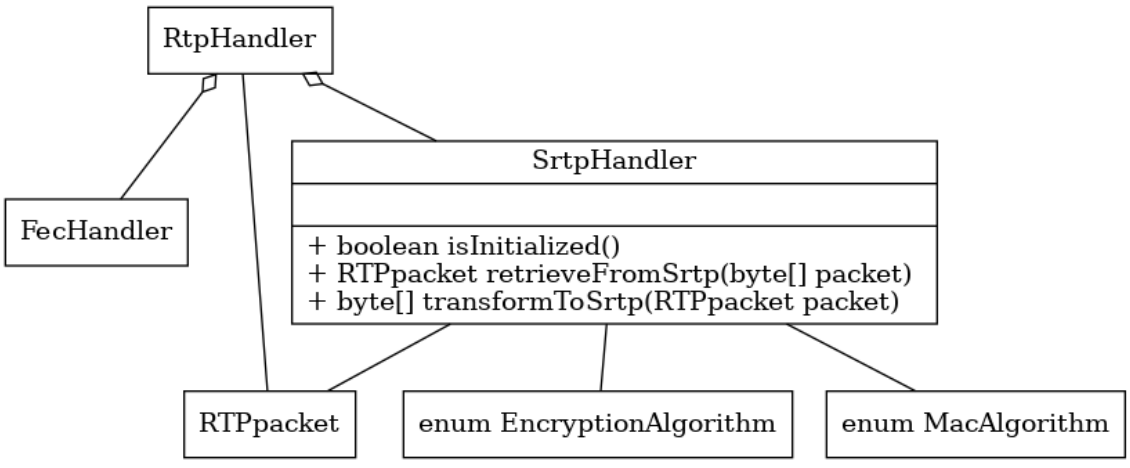


Abbildung 7.2.: Vereinfachte Klassenstruktur der Einbindung des `SrtpHandlers` in das Projekt. Dargestellt sind ebenfalls die Methoden, welche die Schnittstellen zum `RtpHandler` bilden.

Wie bereits ausgeführt, umfasst das Protokoll SRTP ausschließlich die Verschlüsselung und Authentifikation von Paketen. Der Schlüsselaustausch zum Beginn einer Übertragung wird deshalb nicht implementiert. Stattdessen wird ein „Master-Key“ eingebaut, der als bereits vor der Sitzung zwischen Server und Client ausgetauscht gilt. Dieselbe Begründung trifft auch auf die Bereitstellung der Verschlüsselung in den grafischen Oberflächen zu. Um die Implementierung der Absprache der zu verwendenden Algorithmen zu umgehen und damit den Fokus auf SRTP zu belassen, sollen die Nutzer dieselbe simulieren. Die Realisierung dieser Simulation geschieht in Form von Radiobuttons, bei welchen jeweils nur eine der möglichen Optionen gewählt sein kann. Ein weiterer Grund für die Gestaltungsentscheidung der Verschlüsselungsoptionen ist die bessere Möglichkeit für das Testen verschiedener Szenarien. Beispielsweise kann damit das Verhalten eines Client simuliert werden, welcher das Protokoll SRTP nicht implementiert, aber einen solchen Datenstrom erhält.

Der einprogrammierte „Master-Key“ ist für Nutzer nicht editierbar. Zwar könnte diese Funktion implementiert werden, sie dient aber ausschließlich der Simulation des Schlüsselaustausches. Die Unterstützung desselben ist jedoch nicht das Ziel der Implementierung. So ist die Bereitstellung einer Möglichkeit zur Änderung des Schlüssels nicht erforderlich, da diese im Falle der Realisierung des Schlüsselaustausches obsolet würde.

Die Implementierung der Klasse `SrtpHandler` lässt sich in verschiedene Phasen teilen. Diese entsprechen jeweils der Realisierung der AES-Verschlüsselung von Daten, der Ableitung von Sitzungsschlüsseln aus dem „Master-Key“ und der Verarbeitung von RTP-Paketen. Dabei wurde auf die Java-Erweiterung `javax.crypto` zurückgegriffen, die unter anderem auch das Verschlüsselungsverfahren AES bereitstellt. Demnach hat der `SrtpHandler` bei der Verschlüsselung von Daten lediglich die benötigten Parameter bereitzustellen und vorzubereiten sowie das besagte AES-Verfahren anzuwenden. Die Bereitstellung beinhaltet vor allem die Konvertierung der Parameter in einen gemeinsamen Datentyp. Da die Verarbeitung von Daten, wie hier die Verschlüsselung, meist bytebasiert ist, wurden für die Repräsentation der Angaben Bytearrays genutzt. Ein weiterer Schritt in der Vorbereitung der Verschlüsselung ist die Definition des Initialisierungsvektors. Dieser besteht aus der XOR-Verknüpfung des „Salting Keys“, der `SSRC`-Kennzeichnung der Paketquelle sowie dem Paketindex. Der „Salting Key“ ist dabei ein aus dem „Master Key“ abgeleiteter Sitzungsschlüssel, welcher unter anderem von der Ableitungsrate der Schlüssel und dem Paketindex abhängig ist. Ist die Ableitungsrate gleich „0“, wird dieser Schlüssel nur einmal zu Beginn der Sitzung berechnet. Ansonsten gilt er ausschließlich für eine begrenzte Anzahl an Paketen. Besagter Schlüssel dient zum Schutz vor semantischen Angriffen wie beispielsweise bei dem im Abschnitt 6.1 angesprochenen ECB-Modus. Die `SSRC`-Kennzeichnung bei der Berechnung des Initialisierungsvektors ist ein im RTP-Protokoll definierter Wert zur Identifikation verschiedener Quellen für Videoströme. Eine Besonderheit des AES-Verfahrens ist, dass Verschlüsselung und Entschlüsselung denselben Mechanismus verwenden. Somit ist die Verschlüsselung eines Ciphertextes gleich der Entschlüsselung desselben. Jedoch bietet die Klasse `Cipher` des Paketes `javax.crypto` eine generische Schnittstelle für verschiedene Verschlüsselungsverfahren. Deshalb muss bei der Initialisierung dieser Klasse auf die Angabe des korrekten Modus, Verschlüsselung oder Entschlüsselung, geachtet werden. Abgesehen davon sind beide Operationen der Implementierung identisch.

Der zuvor angesprochene „Salting Key"muss, wie auch der entsprechende Schlüssel zur Verschlüsselung, mindestens einmal zu Beginn der Sitzung aus dem „Master Key“ berechnet werden. Dazu wird die Funktion zur Ableitung von Schlüsseln genutzt. Diese basiert, wie bereits im Abschnitt 7.4 erwähnt, genauso wie die Verschlüsselung auf dem Counter-Modus des AES-Verfahrens. Als bislang sicheres Verschlüsselungsverfahren hat es die Eigenschaft, stark von den Eingabeparametern abhängig zu sein und als Ausgabe zufällig aussehende Daten zu generieren. Diese Eigenschaft wird bei der Schlüsselableitung im SRTP genutzt. Ebenso wie bei der Verschlüsselung der Daten besteht der größte Aufwand bei der Implementierung dieser Funktionalität in der Vorbereitung der benötigten Parameter. Besonders sind dabei die Eingangsdaten, welche verschlüsselt werden. Dabei handelt es sich um ein Datenfeld aus Null-Bytes, welches mindestens so lang wie die gewünschte Schlüssellänge sein muss. Der Grund für die Verwendung von Null-Bytes liegt in der Funktionsweise des Counter-Modus. Bei diesem wird aus dem Initialisierungsvektor und dem Schlüssel ein verschlüsselter Datenblock berechnet. Dieser wird anschließend per XOR-Operation mit einem zu verschlüsselnden Datenblock verknüpft. Auf diese Weise wird nicht der Klartext-Datenblock selbst im Verschlüsselungsverfahren verarbeitet. Die XOR-Verknüpfung sorgt jedoch effektiv für eine Verschlüsselung desselben. Für die Schlüsselgenerierung im Rahmen von SRTP ist ausschließlich der vom Verfahren erzeugte verschlüsselte Datenblock ohne die Verknüpfung mit den zu schützenden Daten von Interesse. Deshalb werden Null-Bytes verwendet. Bei der XOR-Verknüpfung von Daten mit Null-Daten werden die Daten nicht verändert. So ist es möglich, den gewünschten Schlüsselstrom aus der generischen Verschlüsselungsfunktion der Klasse `Cipher` zu generieren.

Für die Überprüfung der Schlüsselableitung stellt der RFC 3711 [2] in Anhang B.3 Testdaten bereit. Es wird dabei von festen Werten für den „Master Key“ und das „Master Salt“ ausgegangen. Anschließend werden für einen gegebenen Paketindex die Schlüssel für Verschlüsselung, Authentifikation und „Salting“ angegeben. Auf diese Weise kann überprüft werden, ob die Implementierung wie vorgesehen arbeitet. Deshalb implementiert die Klasse `SrtpHandler` eine Testfunktion für die Schlüsselableitung. In dieser werden alle benötigten Werte manuell gesetzt und anschließend die Schlüssel berechnet. Die Ausführung dieser Methode ergab, dass die erarbeitete Implementierung korrekt funktioniert.

Um die bereits implementierten Verfahren in das Projekt „RTSP-Streaming“ integrieren zu können, müssen auch Methoden zur Verarbeitung von RTP-Paketen bereitgestellt werden. Diese dienen zeitgleich als Schnittstelle zur Klasse `SrtpHandler`. Maßgeblich für den Entwurf dieser Methoden war auch deren Anwendung im `RtpHandler`. Da das Protokoll SRTP zwischen Transportebene und Anwendung agiert, müssen ausgehende Pakete verschlüsselt und empfangene Pakete entschlüsselt werden. Diese Operationen finden direkt vor dem Versenden bzw. direkt nach dem Empfang statt, sodass die weitere Verarbeitung der RTP-Pakete nicht beeinflusst wird. Deshalb nimmt die Methode `transformToSrtp()` ein Objekt der Klasse `RTPpacket` entgegen und gibt ein Bytearray zurück, welches sofort versendet werden kann. Ihre Funktionalität umfasst die Ableitung des Paketindexes, die Verschlüsselung des Paketinhaltes, die Authentifikation des Paketes und die Konkatenation des ursprünglichen RTP-Headers mit den verschlüsselten Paketdaten. Dabei wird auf die zuvor implementierte Funktionalität zur Verschlüsselung der Daten mittels AES zurückgegriffen. Die Authentifikation von Paketen wurde

bei der Entwicklung vorgesehen, jedoch nicht implementiert. So besteht bereits bei der Konstruktion des `SrtpHandlers` die Möglichkeit, eine Authentifizierung zu aktivieren. Die Implementierung dieser Funktionalität kann als Fortsetzung der vorliegenden Arbeit realisiert werden.

Die Entschlüsselung von Paketen wird mit der Methode `retrieveFromSrtp()` ermöglicht. Sie nimmt die empfangenen Daten als Parameter entgegen und gibt das entschlüsselte RTP-Paket zurück. Dabei gleicht sie in ihrem Ablauf und ihrer Funktionalität der Methode `transformToSrtp()`. Notwendig ist dabei die umgekehrte Reihenfolge von Entschlüsselung und Authentifikation. Da die Daten zur Authentifikation an die verschlüsselten Paketdaten angehängt wurden, müssen diese zunächst wieder entfernt und zugleich authentifiziert werden. Erst danach kann die Entschlüsselung stattfinden.

Damit die Paketverarbeitung vor Client und Server gekapselt wird, stellt der `RtpHandler` Methoden als Schnittstellen bereit. Demzufolge findet auch der Aufruf der Verarbeitung mit SRTP in dieser Klasse statt. Zunächst muss sich der `RtpHandler` merken, welcher Art der Verschlüsselung gewünscht ist. Neben der Option, keine Verschlüsselung anzuwenden, soll auch das Protokoll SRTP gewählt werden können. Eine Verarbeitung hinsichtlich der Verschlüsselung erfolgt somit erst nach einer Überprüfung des Verschlüsselungsmechanismus. Für die Bereitstellung von Paketen müssen die Methoden `jpegToRtpPacket()` und `createFecPacket()` auf die Verwendung der Verschlüsselung prüfen. Soll mit SRTP verschlüsselt werden, wird die entsprechende Methode beim `SrtpHandler` aufgerufen. Dabei muss der Inhalt des Paketes nicht berücksichtigt werden.

Auf der Seite des Clients ist gegebenenfalls die Entschlüsselung der empfangenen Pakete erforderlich. Ebenso wie im Server wird dabei auf das gewählte Verschlüsselungsverfahren geprüft. Die Verarbeitung erfolgt dabei ausschließlich beim Empfang von Paketen in der Methode `processRtpPacket()`. Im Anschluss daran kann das Paket wie zuvor weiterverarbeitet werden. Deshalb ist auch keine Berücksichtigung der Verschlüsselung in der Methode `nextPlaybackImage()` notwendig. Das macht den Charakter der Verschlüsselung mit SRTP als Transportverschlüsselung deutlich. Die Verarbeitung laut SRTP-Protokoll erfolgt ausschließlich in direkter Nähe zur Übertragung, sodass alle anderen Verarbeitungsschritte keiner Veränderung bedürfen.

7.6. Test der Verschlüsselung

Die Funktionalität der SRTP-Verschlüsselung kann bei der Ausführung des Projektes getestet werden, indem die Verschlüsselungsoption in den grafischen Oberflächen des Clients und des Servers auf `SRTP` gestellt wird. Dadurch initialisiert der für die Paketverarbeitung zuständige `RtpHandler` den `SrtpHandler`. In der Folge werden alle Pakete vor dem Versenden verschlüsselt bzw. direkt nach dem Empfang entschlüsselt.

Verglichen mit der Übertragung ohne Verschlüsselung lassen sich bei der Übertragung mit SRTP-Paketen subjektiv keine Unterschiede feststellen. Die Verarbeitung der Daten verläuft sowohl auf Senderseite als auch auf der Seite des Empfängers ohne Probleme. Der Client kann die Bilder korrekt und flüssig anzeigen. Insofern erfüllt die Verschlüsselung die Anforderung der Echtzeitfähigkeit. Bei der Simulation von Paketverlusten zeigt das Projekt ebenfalls keine Veränderung des Verhaltens. Niedrige Verlustraten werden per FEC ausgeglichen, während höhere Verluste nicht zu kompensieren sind.

Durch die Analyse der gesendeten Pakete mit dem Programm Wireshark lässt sich die erfolgreiche Verschlüsselung feststellen. Beweis dafür sind unter anderem die im JPEG-Header enthaltenen Bilddimensionen, welche sich im Paketstrom von Bild zu Bild ändern. Tatsächlich wird jedoch eine Video mit konstanten Bildmaßen übertragen. Die sich ändernden Bilddimensionen sind dementsprechend ein Anzeichen für die erfolgte Verschlüsselung.

Die Betrachtung der gesendeten Pakete zeigt auch, dass diese nach wie vor als JPEG-Pakete erkannt werden. Das liegt vor allem am RTP-Header, welcher nicht verschlüsselt wurde und demnach noch dasselbe Format besitzt. Diese Tatsache zeigt, dass die übertragenen Pakete nach außen hin bei oberflächlicher Betrachtung als unverschlüsselt erscheinen. Empfänger, welche die Verschlüsselung per SRTP nicht unterstützen oder nicht über die verschlüsselte Übertragung informiert sind, können demnach die Pakete nicht korrekt entschlüsseln. Dieses Problem ist jedoch eher theoretischer Natur. Die fehlerhafte Entschlüsselung wird im vorliegenden Projekt nur durch eine falsche Einstellung der Verschlüsselung in den Oberflächen von Server und Client ermöglicht. Wird zusätzlich ein Protokoll für den Schlüsselaustausch und die Sitzungsinitialisierung implementiert, ist ein Fehlschlagen der Entschlüsselung nur bei einer fehlerhaften Initialisierung möglich.

Zum selben Ergebnis kommt auch der Versuch, die SRTP-Verschlüsselung im Projekt nur einseitig zu aktivieren. Dabei entschlüsselt der Client entweder unverschlüsselte Pakete oder er versucht, verschlüsselte Daten anzuzeigen. Beide Fälle führen dazu, dass verschlüsselte und somit nicht JPEG-konforme Daten weiterverarbeitet werden. Das führt zum Fehlschlagen der Decodierung des Bildes. Die Erkenntnis daraus ist, dass sowohl Sender als auch Empfänger die Verschlüsselung unterstützen müssen. Außerdem müssen alle benötigten Parameter, wie beispielsweise das Verschlüsselungsverfahren und der Schlüssel, im Vorfeld ausgetauscht und beiden Parteien bekannt sein.

Um die Implementierung der SRTP-Verschlüsselung zu verifizieren, sollten auch Übertragungen mit dem VLC Media Player getestet werden. Allerdings zeigte sich schon bei der Erprobung der SRTP-Übertragung zwischen zwei Instanzen dieses Programmes, dass das Zusammenspiel von RTSP und SRTP nicht unterstützt wird. In einem weiteren Schritt wurden weitere Programme ermittelt, welche die erforderlichen Protokolle implementieren. Dabei ergab sich, dass viele Programme zum Abspielen von Mediendateien und -Strömen die Bibliothek LIVE555 verwenden. Diese implementiert die Funktionalität verschiedener Protokolle für die Echtzeitübertragung von Daten, wozu auch RTP und RTSP gehören. Zur Funktionalität der Bibliothek gehört auch eine Implementierung des SRTP-Protokolls. Jedoch ist diese sehr eingeschränkt und unterstützt zur Zeit ausschließlich die Übertragung bestimmter Netzwerkkameras, welche über eine TLS-Verbindung senden (vgl. [5], Eintrag vom 11.02.2020). Eine solche Verbindung ist im Projekt „RTSP-Streaming“ nicht implementiert. Da keine anderen freien Programme das SRTP-Protokoll implementieren, konnte die Implementierung im besagten Projekt nicht weiter getestet werden.

8. JPEG-spezifische Verschlüsselung

Eine generische Verschlüsselung wie beispielsweise das angesprochene SRTP-Protokoll erfüllt nicht immer alle Anforderungen (vgl. [15], S. 1). Beispielsweise schützt SRTP die Daten zwar während der Übertragung, nach der Entfernung der Übertragungsverschlüsselung können diese jedoch ungehindert weiterverarbeitet werden. Zur Verschlüsselung von JPEG-Dateien wurde deshalb schon viel geforscht. Das Ergebnis dieser Forschung ist eine Vielzahl spezieller Verfahren, deren Eignung für die gewünschten Anwendungsszenarien jeweils überprüft werden muss.

Wie bei anderen Verschlüsselungsverfahren auch spielt die Aushandlung des Verfahrens und des Schlüssels zwischen den Kommunikationsteilnehmern eine wesentliche Rolle. Das JFIF-Format bietet dabei keine standardisierte Möglichkeit, ein bei der Verschlüsselung verwendetes Verfahren anzugeben. Auch aufgrund der Vielfalt unterschiedlicher Verfahren ist die Kommunikation des Verfahrens schwierig. Deshalb muss diese zwangsläufig über einen anderen Kanal als das JPEG-Bild selbst stattfinden.

Nachfolgend soll zunächst genauer auf die Gründe für die Anwendung einer JPEG-spezifischen Verschlüsslung eingegangen werden. Dem folgt ein Überblick über die verschiedenen Ansätze einer solchen Verschlüsselung. Da im „RTSP-Streaming“ bereits komprimierte Bilder verschlüsselt werden sollen, liegt ein besonderer Fokus auf der Verarbeitung der Bilddaten im JFIF-Format. Ein weiterer Abschnitt beschäftigt sich mit dem Entwurf einer JPEG-Verschlüsselung. Deren Implementierung und Unsicherheit wird im sich daran anschließenden Abschnitt besprochen. Abschließend wird ein besseres, doch deutlich aufwendigeres Verschlüsselungsverfahren für JPEG-Bilder vorgestellt.

8.1. Gründe für die Anwendung JPEG-spezifischer Verschlüsselung

Der deutlichste Unterschied zwischen generischen Übertragungsverschlüsselungen und JPEG-spezifischer Verschlüsselung ist die logische Ebene, auf der die Verfahren angewendet werden. Bei Übertragungsverschlüsselungen, zu denen auch SRTP gehört, werden meist Netzwerkpakete verarbeitet. Ihr Schutz währt nur für die Dauer der Übertragung vom Sender zum Empfänger. Nachdem die Pakete vom Empfänger entschlüsselt wurden, sind die Originaldaten wieder hergestellt und können weiterverarbeitet werden. JPEG-spezifische Verschlüsselungen werden für jedes Bild einzeln angewendet. Bei der Übertragung eines auf diese Weise verschlüsselten Bildes wäre das Datenformat als JPEG identifizierbar. Jedoch könnte der originale Bildinhalt nicht ohne die korrekte Entschlüsselung angezeigt werden. Der Vorteil einer auf Bildebene agierenden Verschlüsselung ist dabei, dass die Daten auch nach dem Empfang noch vor Weiterverarbeitung geschützt sind. Darunter sind sowohl die Bearbeitung der Bildinhalte selbst als auch das Kopieren und Weitergeben des Bildes zu verstehen.

Ein weiterer Vorteil spezifischer Verschlüsselungen liegt in der Menge der zu verschlüsselnden Daten. Generische Verfahren verschlüsseln meist alle zu schützenden Daten ohne Rücksicht auf deren Struktur. Das erleichtert die Verschlüsselung vieler verschiedener Datenstrukturen. Schwierigkeiten können dabei jedoch entstehen, wenn große Mengen an Daten in Echtzeit zu verarbeiten sind. Auch wenn moderne Rechner ohne Probleme größere Datenmengen verarbeiten können, liegen die Grenzen der Verarbeitbarkeit entweder in noch größeren Datenmengen oder schwächerer Hardware, wie beispielsweise „Internet of Things"-Geräten. Bei JPEG-spezifischen Verfahren ist es möglich, nur einen Teil der Daten zu verschlüsseln und dabei dieselbe Sicherheit zu erlangen. Der Grund dafür ist, dass bestimmte Datenfelder für die Decodierung der Bilder besonders wichtig sind. Beispielsweise kann die Decodierung nicht das Originalbild wiederherstellen, wenn die Huffman- oder Quantisierungstabellen verschlüsselt sind. Gleiches trifft auch auf die Verschlüsselung der entropiecodierten Daten zu. In der Entwicklung der Verschlüsselung multimedialer Inhalte wurde partielle Verschlüsselung deshalb dazu genutzt, um die Echtzeitfähigkeit der Verfahren herzustellen (vgl. [16], S. 3).

Schlussendlich können auch Anforderungen an das Datenformat Gründe für die Verwendung von JPEG-Verschlüsselung sein. Für generische Verfahren gibt es häufig keine Notwendigkeit eines bestimmten Datenformates. Bei SRTP ist zwar der Header der RTP-Pakete gefordert, die enthaltenen Daten müssen allerdings kein bestimmtes Format erfüllen. Im Gegensatz dazu ist das Ergebnis JPEG-spezifischer Verfahren ein verschlüsseltes Bild im JPEG-Format. Durch die Beachtung des Datenformates bei der Verschlüsselung werden wiederum gültige JPEG-Bilder erzeugt. Dabei muss vor allem auf die Marker geachtet werden. Zum einen dürfen diese nicht verschlüsselt werden, weil sonst Formatinformationen verloren gingen. Zum anderen müssen die Verschlüsselungsverfahren darauf achten, keine Daten zu erzeugen, die fälschlicherweise als Marker erkannt werden könnten. Diese würden sonst zu einer Kompromittierung des Datenformates führen. Im schlimmsten Fall könnte ein solches verschlüsseltes JPEG-Bild auch nicht wieder korrekt entschlüsselt werden. Neben der Notwendigkeit für die erfolgreiche Entschlüsselung hat die Beibehaltung des JPEG-Formates außerdem den Effekt, dass verschlüsselte Bilder von Decodern verarbeitet werden können, welche das Verschlüsselungsverfahren nicht implementieren.

8.2. Gliederung von JPEG-basierten Verschlüsselungsverfahren

Abhängig von den Anforderungen an die Verschlüsselung von JPEG-Bildern erweisen sich Verfahren mit unterschiedlichen Ansätzen als geeignet. Diese Verfahren lassen sich entsprechend ihrer Abfolge im Bezug auf die JPEG-Kompression gliedern. Dabei ist die Verschlüsselung von Bilddaten vor, während oder nach der Kompression möglich. Eine Übersicht über exemplarische Verfahren der verschiedenen Ansätze haben Li und Lo in ihrer Untersuchung aus dem Jahr 2020 erstellt (vgl. [15]).

Die Verschlüsselung von Bilddaten vor der Kompression ist im Allgemeinen einfach zu implementieren, da sie die unkomprimierten Daten verarbeitet. Sie wird häufig als Encryption-Then-Compression (EtC) bezeichnet. Da für die anschließende Kompression keine Änderungen stattfinden müssen, ist keine angepasste Software für die Codierung

und Decodierung erforderlich. Das Ergebnis der Kompression eines verschlüsselten Bildes entspricht außerdem dem geforderten Format der Daten. Der deutlichste Nachteil solcher Verfahren ist ihr Einfluss auf die Kompressionsqualität. Durch die Verschlüsselung der Eingangsdaten der Kompression wird die Korrelation der Pixel im Originalbild zerstört (vgl. [15], S. 478). Auf dieser beruht allerdings das JPEG-Kompressionsverfahren. Verschlüsselungen, welche diese Tatsache nicht berücksichtigen, verringern die Datenreduktion durch die Kompression deutlich. Ein EtC-Verfahren sollte Bilder deshalb auf solche Weise verschlüsseln, dass die Korrelation der Pixel bestmöglich beibehalten wird.

Bei der Verschlüsselung von Bildern während der Kompression gibt es deutliche Unterschiede in den Abläufen und den Eigenschaften der Verfahren. Diese sind vor allem davon abhängig, an welcher Stelle der Kompression das jeweilige Verfahren ansetzt. Derartige Verschlüsselungen haben den Vorteil, dass sie auf Zwischenergebnisse der Datenreduktion zugreifen und diese modifizieren können. So agieren diese Verfahren beispielsweise auf der Basis der quantisierten Koeffizienten oder durch Veränderungen in der Entropiecodierung. Viele der während der Kompression ablaufenden Verschlüsselungen können jedoch einen negativen Einfluss auf das Gesamtergebnis haben. Die Schwierigkeit dieser Verfahren besteht dabei darin, die Eigenschaften der Originaldaten bei der Verschlüsselung beizubehalten. Durch die Modifikation von Werten, wie beispielsweise Huffmantabellen, können die Daten gegebenenfalls nicht mehr so effizient reduziert werden. Für die verbreitete Nutzung solcher Verfahren ist außerdem hinderlich, dass angepasste Codierer und Decodierer verwendet werden müssen. Standardsoftware für die Anzeige von JPEG-Bildern könnten die so erzeugten Bilder zwar verarbeiten. Da jedoch die entsprechenden Verschlüsselungsverfahren im Allgemeinen nicht unterstützt werden, ist die Entschlüsselung und Anzeige des Originalbildes nicht möglich.

JPEG-Verschlüsselungsverfahren, die nach der Kompression verschlüsseln, werden „bitstream-based“ genannt. Da sie den Kompressionsvorgang nicht beeinflussen, sind sie mit Standardsoftware für die Verarbeitung von JPEG-Bildern kompatibel. Außerdem ist es solchen Verfahren aufgrund ihrer Unabhängigkeit von der Kompression möglich, verschlüsselte Bilder mit ähnlicher Größe wie die Originalbilder zu erzeugen. Allerdings muss besondere Vorsicht angewandt werden, um das Datenformat beizubehalten. Generische Verschlüsselungsalgorithmen wie beispielsweise AES würden bei der Anwendung auf große Datenblöcke das Format der Datei zerstören. Deshalb ist eine umsichtige Auswahl der zu verschlüsselnden Datenabschnitte nötig. Für das JPEG-Format muss darauf geachtet werden, dass keine Marker verschlüsselt werden. Andererseits dürfen bei der Kompression keine Daten entstehen, die als Marker fehlinterpretiert werden könnten. Ähnliches gilt für die Verschlüsselung von entropiecodierten Daten im SOS-Segment. Um ein Bild auch nach der Verschlüsselung noch decodieren zu können, dürfen weder Huffmancodes verschlüsselt noch Daten erzeugt werden, die fälschlicherweise als solche Codes interpretiert werden können.

8.3. JPEG-Verschlüsselung auf Basis der Quantisierungstabellen

Für den Entwurf einer auf der JPEG-Struktur basierenden Verschlüsselung bieten die Huffman- und Quantisierungstabellen einen guten Ansatzpunkt. Beide Arten von Ta-

bellen sind zwingend für die korrekte Decodierung der Bilder erforderlich. Eine Verschlüsselung dieser Daten hätte demnach zur Folge, dass ein Bild nur mit dem korrekten Schlüssel betrachtet werden kann. Weiterhin handelt es sich bei den Tabellen um einige Blöcke an Daten, die als Gesamtes verarbeitet werden können. Das ist besonders für die Verschlüsselungsalgorithmen von Vorteil. Nicht zuletzt ist durch die Verschlüsselung der Tabellen nur ein Bruchteil des ganzen Bildes zu verarbeiten. Das macht das Verfahren schnell und in Echtzeit ausführbar.

Die weiteren Anforderungen an die Verschlüsselungen werden im nächsten Abschnitt diskutiert. Außerdem wird darin ein Entwurf des Verfahrens erstellt. Daran schließt sich ein Abschnitt zur Betrachtung der Implementierung der Verschlüsselung an. Zuletzt werden die Schwachstellen des entwickelten Verfahrens analysiert und ein Angriffsversuch auf dasselbe unternommen.

8.3.1. Entwurf des Verfahrens

Für den Entwurf einer JPEG-basierten Verschlüsselung ist zunächst die Analyse der Anforderungen erforderlich. Diese entsprechen größtenteils den in Abschnitt 6.1 ausgeführten. Für den Anwendungsfall der Übertragung per RTP sollen diese genauer beschrieben werden. Auf diese Weise ist ein konkreter Entwurf der Verschlüsselung möglich. Eine Authentifikation der Bilder ist dabei nicht integriert. Sie ist zwar grundsätzlich möglich, erfordert jedoch eigene Verfahren. Diese können beispielsweise den Urheber des entsprechenden Bildes vermerken. Aufgrund des Umfangs der Authentifikationsmechanismen und deren Implementierung spielen sie im weiteren Verlauf der Arbeit allerdings keine Rolle.

Angewendet werden soll die Verschlüsselung bei der RTP-Übertragung von Videodateien im MJPEG-Format. Als Grundlage der Implementierung wird das Projekt „RTSP-Streaming“ verwendet. Da die Bilder im besagten Projekt gemäß RFC 2435 [3] zu RTP-Paketen umgeformt werden, muss das JPEG-Datenformat eingehalten werden. Dementsprechend findet die geplante Verschlüsselung zwischen dem Auslesen der Bilddaten und deren Verarbeitung zu Paketen statt. Die Verschlüsselung operiert damit auf den komprimierten Bilddaten. Daraus folgen die in Abschnitt 8.2 erläuterten Schwierigkeiten für die Verschlüsselung komprimierter Daten. Diese erweisen sich jedoch für das Grundkonzept der zu entwerfenden Verschlüsselung als nicht besonders kompliziert.

Der Kerngedanke des Entwurfes ist die Verschlüsselung von für die Decodierung benötigten Tabellen. Dies sind entweder die bei der Entropiecodierung verwendeten Huffmantabellen oder die Quantisierungstabellen, welche die für die diskrete Kosinustransformation-Koeffizienten benötigte Datenmenge reduzieren. Allerdings sind für die Übertragung von JPEG-Bildern per RTP bestimmte Huffmantabellen vorgegeben (siehe Abschnitt 4.1). Eine Verschlüsselung der Huffmantabellen wäre deshalb nutzlos, da diese ohnehin nicht in die Pakete eingebracht und somit nicht übertragen werden. Die Quantisierungstabellen können dagegen in einem eigenen Abschnitt der RTP-Pakete für JPEG transportiert werden.

Im JPEG-Datenformat sind Quantisierungstabellen in Form von DQT-Segmenten gespeichert. Diese enthalten mindestens eine der genannten Tabellen. Sind für die Decodierung eines JPEG-Bildes mehrere Quantisierungstabellen erforderlich, können diese dementsprechend in einem oder auch in mehreren DQT-Segmenten definiert werden.

Solche Segmente besitzen ein vorgegebenes Format, was die Verarbeitung der enthaltenen Daten vereinfacht. Die Tabellen liegen dabei als Datenblock mit fester Länge vor. Solange sie bei der Verschlüsselung auf ein Chiffrat derselben Länge abgebildet werden, vergrößern sich die zu übermittelnden Daten nicht. Auch eine Fehlinterpretation der verschlüsselten Daten wird ausgeschlossen, wenn die für die Speicherung der Quantisierungstabellen benötigten Datenmenge gleich bleibt. Da auf diese Weise die vorgegebene Struktur nicht verändert wird, besitzt auch das verschlüsselte Bild ein valides JPEG-Format. Weil mit den Quantisierungstabellen außerdem nur Teile eines Bildes verschlüsselt werden, ist die Verarbeitung schneller als bei einer Verschlüsselung des gesamten Bildes. Diese Tatsache trägt dazu bei, dass die Verschlüsselung in Echtzeit stattfinden kann.

Um die Datenmenge bei der Verschlüsselung nicht zu vergrößern, muss ein geeignetes Verschlüsselungsverfahren gewählt werden. Als schnelle Verfahren kommen dabei Blockchiffren und Stromchiffren infrage. Allerdings sind nur Stromchiffren in der Lage, alle möglichen Eingangsdaten auf ein Chiffrat derselben Länge abzubilden. Da Blockchiffren jedoch deutlich weiter verbreitet sind, liegt die Lösung in der Wahl eines geeigneten Betriebsmodus. Ein Beispiel dafür ist der AES-CM, welcher auch im SRTP verwendet wird und für die zu entwickelnde Verschlüsselung genutzt werden soll. Der Counter-Modus erlaubt die Verwendung des AES als Stromchiffre, womit die für die Speicherung benötigte Datenmenge durch die Verschlüsselung gleich bleibt. Wie im Abschnitt 7.2 bereits erläutert, handelt es sich beim AES um ein sicheres Verschlüsselungsverfahren. Die Anforderung der kryptografischen Sicherheit für den Verschlüsselungsentwurf ist damit Genüge getan. Weiterhin gilt ebenfalls, dass die schnelle Abarbeitung der AES-Chiffre die Echtzeitfähigkeit der Verschlüsselung begünstigt.

8.3.2. Implementierung und Test der Verschlüsselung

Für die Implementierung der entworfenen Verschlüsselung im „RTSP-Streaming“-Projekt wurde die neue Klasse `JpegEncryptionHandler` angelegt. Diese realisiert die vorgestellte Funktionalität und stellt sie dem `RtpHandler` zur Verwendung bereit. In Abbildung 8.1 ist die Anbindung des `JpegEncryptionHandlers` an das Projekt dargestellt. Als Schnittstellen fungieren dabei die beiden Methoden `encrypt()` und `decrypt()`, welche jeweils die Verschlüsselung bzw. Entschlüsselung von JPEG-Bildern vornehmen. Da die Verschlüsselung im Gegensatz zu SRTP jedoch auf Bildebene agiert, muss sie an anderen Stellen in der Verarbeitung der Bilder zu Paketen aufgerufen werden. Direkt nach dem Auslesen eines Bildes aus der Videodatei wird das Bild verschlüsselt. Damit können alle nachfolgenden Schritte ohne Änderung durchgeführt werden. Die Entschlüsselung eines Bildes erfolgt erst kurz vor der Anzeige desselben in der grafischen Oberfläche. Jegliche Verarbeitung davor dient der Extraktion des Bildes aus dem RTP-Paket. Erst kurz vor der Darstellung ist das Bild im JPEG-Format vorhanden, welches zur Entschlüsselung erforderlich ist.

Die Abläufe von Ver- und Entschlüsselung sind im entworfenen Verfahren nahezu identisch. Zunächst wird das zu verarbeitende Bild nach DQT-Markern durchsucht. Wird ein solcher Marker gefunden, beginnt die weitere Verarbeitung. Dabei werden die im durch den Marker gekennzeichneten Segment enthaltenen Quantisierungstabellen einzeln ver- bzw. entschlüsselt. Der dafür benötigte Schlüssel und der „Salt“-Parameter werden dabei als bereits ausgetauscht angenommen. Auf diese Weise ist die Implementie-

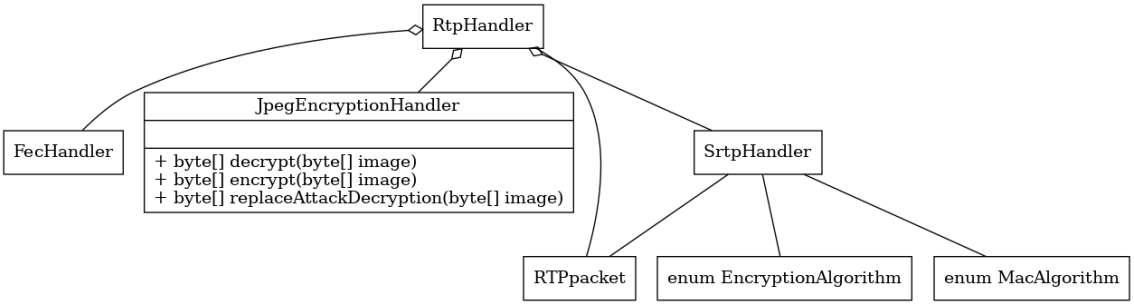
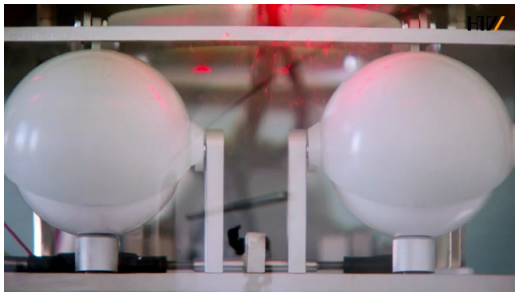


Abbildung 8.1.: Anbindung des `JpegEncryptionHandlers` an das Projekt mit vereinfachter Klassenstruktur. Die Schnittstelle zur Funktionalität besteht aus Methoden zur Verschlüsselung und Entschlüsselung von -Bildern und einer Methode zum Angriff auf die Verschlüsselung.

rung des Schlüsselaustausch nicht erforderlich. Im Anschluss an die Verschlüsselung der Tabellen des Segments wird nach weiteren DQT-Markern gesucht. Sind im Bild weitere solche Segmente vorhanden, werden diese äquivalent verarbeitet. Alle bei dieser Verarbeitung nicht verschlüsselten Abschnitte der Datei werden ohne Änderung übernommen.



(a) Originalbild „SdI“

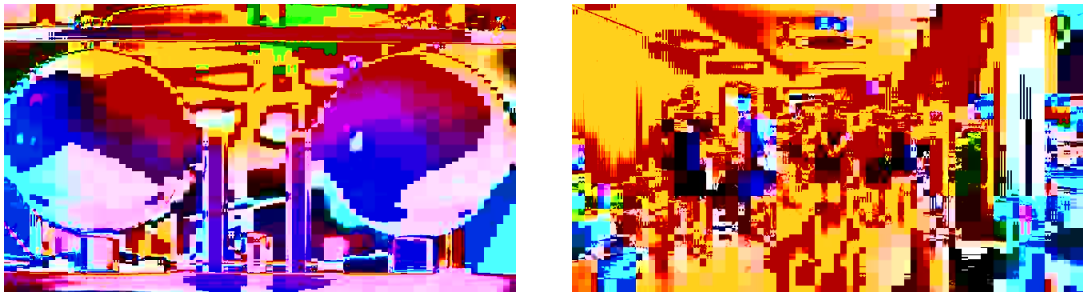
(b) Originalbild „LNdW“

Abbildung 8.2.: Originalbilder aus dem Video zum Studium der Informatik [28] und dem Video zur Langen Nacht der Wissenschaften 2019 [14]. Bei den Bildern handelt es sich jeweils um das erste Frame des entsprechenden Videos.

Um die Korrektheit der Verschlüsselung zu überprüfen, wurde ein einfacher Test implementiert. Dieser liest ein Bild aus der als Parameter angegebenen Datei ein und verarbeitet es. Für jeden Schritt in der Verarbeitung wird ein separates Bild in das Arbeitsverzeichnis geschrieben, welches nach der Ausführung angesehen und inspiziert werden kann. Als Testbilder wurden jeweils das erste Frame aus zwei Videos genutzt. Dabei handelt es sich um das Video zum Studium der Informatik an der HTWDD [28] sowie das Video zur Langen Nacht der Wissenschaften an der HTWDD im Jahr 2019 [14]. Die für das Testen der Verschlüsselung verwendeten Bilder sind in Abbildung 8.2 dargestellt. Bei dem Test werden die Bilder zunächst verschlüsselt und anschließend wieder entschlüsselt.

Die Durchführung des Tests ergab, dass die Verschlüsselung korrekt implementiert wurde und umkehrbar ist. Das heißt, bei der Entschlüsselung des verschlüsselten Bildes entsteht wieder das Originalbild. Eine Schwachstelle zeigte sich bei der Betrachtung der

verschlüsselten Bilder, welche in Abbildung 8.3 zu sehen sind. Während die Farbwerte und die blockinternen Strukturen zwar deutlich unkenntlich gemacht sind, können einige Konturen des Originalbildes immer noch erkannt werden. In der Verschlüsselung des Bildes „SdI“ bezieht sich das vor allem auf die Umrisse der Augen des Roboters. Dagegen können im verschlüsselten Bild „LNdW“ hauptsächlich die Konturen von Farbflächen erkannt werden. Aber auch detailliertere Strukturen, wie beispielsweise die Umrisse der Panele auf der linken Seite oder zwei an der Decke hängende Luftballons, können erahnt werden. Dass solche Konturen auch noch in den verschlüsselten Bildern erkannt werden können, weist auf eine geringe wahrnehmungsbezogene Sicherheit hin. Konkret bedeutet das, dass trotz angewendeter Verschlüsselung noch Inhalte des Originalmediums erkennbar sind. Da nur ein Teil der in den Bildern enthaltenen Daten verschlüsselt wurden, ergibt sich daraus, dass die Auswahl der zu verschlüsselnden Daten beim Entwurf der Verschlüsselung für diese nicht geeignet ist.



(a) Verschlüsseltes Bild „SdI“ (b) Verschlüsseltes Bild „LNdW“

Abbildung 8.3.: Bilder mit verschlüsselten Quantisierungstabellen.

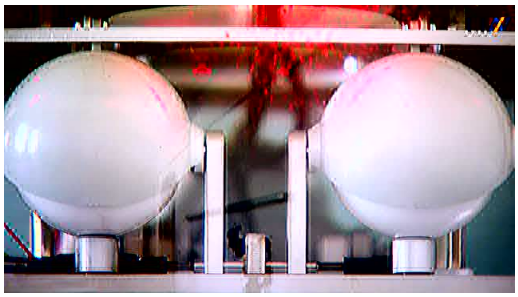
8.3.3. Angriff auf die Verschlüsselung der Quantisierungstabellen

Die entworfene Verschlüsselung besitzt einen nachvollziehbaren und sinnvollen Grundgedanken. Mit der Verschlüsselung der Quantisierungstabellen werden die Farbwerte der Bildblöcke unkenntlich gemacht und das Originalbild kann nicht mehr angezeigt werden. Durch die Verwendung des AES mit Counter-Modus können die Daten nicht ohne das Wissen über den verwendeten Schlüssel und das „Salt“ wiederhergestellt werden. Weil im Verhältnis zur Größe des gesamten Bildes wenig Daten verschlüsselt werden und ein schnelles symmetrisches Verschlüsselungschiffre verwendet wird, kann die Verschlüsselung in Echtzeit stattfinden.

Trotz der größtenteils guten Entscheidungen beim Entwurf der Verschlüsselung zeigte sich bereits im vorigen Abschnitt, dass das entworfene Verfahren eine Schwachstelle besitzt. Mit der Verschlüsselung der Quantisierungstabellen wurde darauf geachtet, dass ein für die Decodierung notwendiger Teil der Daten geschützt wird. Allerdings verändert dieser Schutz nicht die Korrelation zwischen benachbarten Bildblöcken. Da die entropiecodierten Daten in das verschlüsselte Bild übernommen werden, bleibt die Prädiktion der DC-Koeffizienten bestehen. Somit ist die ähnliche Helligkeit benachbarter Bildblöcke auch im verschlüsselten Bild zu sehen. Die auf diese Weise erkennbaren Konturen verletzen die wahrnehmungsbezogene Sicherheit. Darum ist die entworfene Verschlüsselung unsicher.

Die Tatsache, dass die Quantisierungstabellen verschlüsselt wurden, lässt sich noch weiter ausnutzen. Wie schon festgestellt wurde, können die verschlüsselten Daten nicht ohne Weiteres von Dritten entschlüsselt werden. Allerdings lässt sich die Verschlüsselung durch eine Ersetzung angreifen. Deren Prinzip beruht darauf, dass Quantisierungstabellen immer eine feste Größe besitzen und der JPEG-Standard Beispieltabellen angibt. Diese Tabellen des Standards wurden empirisch ermittelt und erzielen gute Ergebnisse (vgl. [8], S. 143). Insofern weichen sie aller Wahrscheinlichkeit nicht sehr von den in den Beispielbildern verwendeten Quantisierungstabellen ab.

Anstatt der Entschlüsselung empfangener Bilder können dementsprechend die Speicherorte der verschlüsselten Quantisierungstabellen gesucht und diese durch die Tabellen des JPEG-Standards ersetzt werden. Im „RTSP-Streaming“-Projekt wurde diese Vorgehensweise implementiert und im Client als Option verfügbar gemacht. Außerdem wurde ein Testfall für diesen Angriff erstellt. Die Ergebnisse für die Ersetzung der Tabellen in den Beispielbildern sind in Abbildung 8.4 dargestellt.



(a) Ergebnis des Ersetzungsangriffs auf das verschlüsselte Bild „SdI“

(b) Ergebnis des Ersetzungsangriffs auf verschlüsselte Bild „LNdW“

Abbildung 8.4.: Ersetzungsangriff auf die verschlüsselten Bilder.

Wie in der Darstellung zu sehen, erweist sich der Angriff auf die verschlüsselten Bilder als erfolgreich. Die Farbwerte wurden weitestgehend wieder hergestellt und nur einige Blockartefakte verbleiben. Im Vergleich zu den Originalbildern scheinen die Ergebnisse der Angriffe einen etwas erhöhten Kontrast zu besitzen. Das ist auf die Unterschiede der Quantisierungstabellen aus dem JPEG-Standard zu den tatsächlich bei der Kompression verwendeten Tabellen zurückzuführen.

Dieser Angriff zeigt außerdem, dass der Entwurf einer Verschlüsselung für Mediendaten viel Aufwand erfordert. Neben der Erfüllung der anwendungsspezifischen Anforderungen ist jederzeit die Sicherheit des Verfahrens zu beurteilen. Auch bei ansonsten gut gewählten Verschlüsselungsparametern reicht eine Schwachstelle aus, um das Verfahren zu kompromittieren. Bei der hier entworfenen Verschlüsselung liegt der Schwachpunkt in der Auswahl der zu verschlüsselnden Daten. Da es Beispieldaten für Quantisierungstabellen gibt und sich diese generell aufgrund ihrer Struktur und Bedeutung bei der Kompression ähnlich sehen, eignen sie sich nicht als Ziel der Verschlüsselung. Außerdem hat die Verschlüsselung dieser Tabellen keinen Einfluss auf die Korrelation der Bildblöcke, weshalb diese auch nach der Verarbeitung bestehen bleibt. Stattdessen könnte eine Verschlüsselung der entropiecodierten Daten zu einem sichereren Verfahren führen.

8.4. Bitstream-basierte Verschlüsselung mit Beibehaltung der Dateigröße

Wie sich in den vorigen Abschnitten gezeigt hat, sind bei der Gestaltung JPEG-basierter Verschlüsselungen viele verschiedene Aspekte zu beachten. Der in 8.3.1 gewählte Ansatz berücksichtigt die meisten dieser Anforderungen. Seine Schwachstelle liegt jedoch, wie bereits ermittelt, bei der Auswahl der zu verschlüsselnden Daten. Ein weiteres Verfahren, welches komprimierte Bilddaten verschlüsselt, stellen Kobayashi und Kiya in ihrem Artikel [11] vor. Statt der Verschlüsselung der Quantisierungstabellen stellen sie ein Verfahren vor, welches die entropiecodierten Daten schützt.

Bei der Huffmancodierung in der JPEG-Kompression werden die quantisierten Koeffizienten auf einen Huffmancode abgebildet. Dieser leitet sich aus der Größenordnung des Wertes ab. Maßgeblich ist dafür das höchste gesetzte Bit im ein Byte großen Wert. Der Koeffizient wird anschließend in Form des Huffmancodes codiert. Um den ursprünglichen Wert vollständig wiederherstellen zu können, wird eine Repräsentation desselben im Anschluss an den Huffmancode gespeichert. Da diese eine je nach Huffmancode unterschiedliche Größe besitzt, werden die Daten der Repräsentation auch als zusätzliche Bits bezeichnet. Das Grundprinzip des von Kobayashi und Kiya vorgestellten Verfahrens basiert auf der Verschlüsselung dieser Bits. Da diese die ursprünglich quantisierten Werte codieren, macht eine Verschlüsselung derselben die Darstellung des Originalbildes unmöglich. Dadurch ist die Sicherheit des Verfahrens im Bezug auf die Wahrnehmung gewährleistet. Die Schwierigkeit des Verfahrens liegt dabei in der Ermittlung der zu verschlüsselnden Daten. Für die Extraktion der zusätzlichen Bits muss das vollständige Segment der entropiecodierten Daten geparst werden. Erforderlich sind dafür die vollständigen Huffmantabellen, wie sie zur Decodierung von Bildern gebraucht werden. Damit bedarf die Implementierung der Verschlüsselung die vollständige Funktionalität der Huffmandecodierung aus der JPEG-Kompression. Außerdem werden die entropiecodierten Daten bei der Verschlüsselung byteweise verarbeitet. Wie in Abbildung 8.5 zu sehen ist, überschreiten Huffmancodes und zusätzliche Bits aber teilweise die Bytegrenzen. So ist eine reine byteweise Verarbeitung zur Verschlüsselung der Daten nicht möglich. Trotz des einfachen Grundgedankens des Verschlüsselungsverfahrens stellt die Realisierung desselben einen sehr hohen Aufwand dar.

Für die Verschlüsselung nach Kobayashi und Kiya wird zunächst byteweise über die entropiecodierten Daten iteriert. Dabei werden die zuvor beschriebenen zusätzlichen Bits extrahiert, wenn zwei Bedingungen erfüllt sind. Diese Bedingungen gewährleisten, dass die Decodierbarkeit des JPEG-Bildes beibehalten wird. Zum einen müssen in dem entsprechenden Byte sowohl Teile von Huffmancodes als auch Teile von zusätzlichen Bits enthalten sein. Zum anderen muss der enthaltene Huffmancode mindestens ein Null-Bit haben. Der Grund dafür ist, keine neuen `0xFF`-Bytes zu erzeugen, da diese im JPEG-Format als Kennzeichnung eines Markers gelten. Um ein auf diese Weise erzeugtes Byte als Datum zu kennzeichnen, müsste ein zusätzliches Null-Byte eingefügt werden. Bei der Entschlüsselung könnten die dabei entstehenden Bytes `0xFF 0x00` allerdings nicht eindeutig entschieden werden, ob sie durch die Verschlüsselung oder durch die Kompression entstanden sind. Eine forcierte Entscheidung könnte dabei zum Fehlschlagen der Entschlüsselung und anschließenden Decodierung führen. Mit dem Ausschluss solcher Bytes von der Verschlüsselung wird einer Vergrößerung des Bildes vorgebeugt.

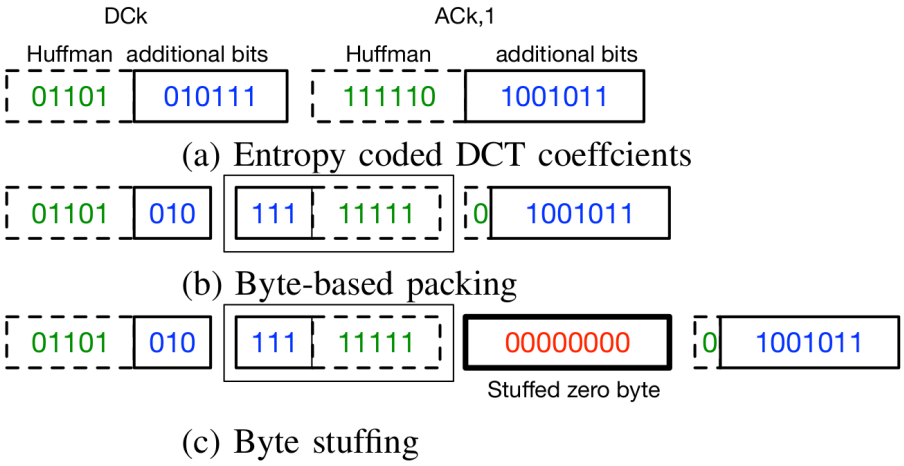


Abbildung 8.5.: Beispiel für das Ergänzen eines Null-Bytes in den entropiecodierten Daten. Huffmancodes und zusätzliche Bits überschreiten dabei häufig die Bytegrenzen. (aus [11], S. 385).

Im Anschluss an die Extraktion wird unter Verwendung eines geheimen Schlüssels eine zufällige Bitsequenz generiert. Das kann mithilfe eines Verschlüsselungsverfahrens bewerkstelligt werden. Dabei ist darauf zu achten, dass ein sicheres Verfahren verwendet wird. Ein Beispiel dafür ist AES im Counter-Modus, wie es auch bei SRTP eingesetzt wird. Die extrahierten Bits werden nachfolgend per XOR-Operation mit der Bitsequenz verknüpft. Diese Verarbeitung entspricht der Verschlüsselung mit einer Stromchiffre. Anstelle einer solchen Chiffre könnte alternativ auch eine Blockchiffre mit einem Betriebsmodus verwendet werden, der die Nutzung als Stromchiffre möglich macht. Zuletzt wird das verschlüsselte Bild erzeugt, indem die nicht verschlüsselten Daten mit dem verschlüsselten Bitstrom kombiniert werden.

Das vorgestellte Verschlüsselungsverfahren macht es auch möglich, nur Teile der zusätzlichen Bits zu verarbeiten. Beispielsweise können nur die DC-Koeffizienten verschlüsselt werden. Dadurch würde die zu verschlüsselnde Datenmenge reduziert und die Ausführungszeit verringert werden. Auf diese Weise kann die Echtzeitfähigkeit hergestellt oder begünstigt werden.

9. Zusammenfassung und Ausblick

In der vorliegenden Arbeit sollten ein Analyseprogramm, welches die Eignung von MJPEG-Dateien prüft, entworfen und implementiert, Optimierungen im Projekt „RTSP-Streaming“ vorgenommen sowie verschiedene Verschlüsselungskonzepte für diese Übertragung untersucht und praktisch erprobt werden. Den Beginn dieser Untersuchungen stellte die Betrachtung der JPEG- und MJPEG-Formate dar. Es stellte sich heraus, dass im Gegensatz zu JPEG kein standardisiertes MJPEG-Format existiert. Die meisten dieser Formate speichern die Daten jedoch mit einem ähnlichen Aufbau. Neben dem vom VLC Media Player generierten MJPEG-Format gibt es auch Containerformate wie QTFF und AVI, welche Metadaten des Videos speichern können.

Bei der Analyse von JPEG-Übertragungen per RTP zeigte sich, dass verschiedene Anforderungen an die Übertragung der Daten gestellt werden. Dazu gehören zum Beispiel Maximalwerte für die Dimension von Bildern, Vorgaben für Huffmantabellen und die Verwendung quadratischer Pixel. Die Untersuchung verschiedener Dateien ergab, dass diese Kriterien von manchen Konvertierungsprogrammen nicht eingehalten werden. Durch die Skalierung von Videos bei der Konvertierung mit dem VLC Media Player werden die Daten so verändert, dass die Pixel des erzeugten Videos nicht mehr quadratisch sind. Außerdem werden nicht die geforderten Huffmantabellen zur Entropiecodierung verwendet. Parameter zur Abstellung dieser Fehler konnten bei der Konvertierung nicht eingestellt werden. Die andere untersuchte Konvertierungssoftware ffmpeg erzeugte zunächst ebenfalls für die Übertragung ungeeignete Dateien. Grund dafür war auch die Verwendung anderer als der vorgegebenen Huffmantabellen. Im Gegensatz zum VLC Media Player konnte dieses Verhalten jedoch über einen Parameter angepasst werden. In der Folge wurden geeignete MJPEG-Dateien erzeugt.

Im Anschluss an diese Analysen wurde die Optimierung der Software „RTSP-Streaming“ der HTWDD vorgenommen. Dabei sollte die Laufzeit bestimmter Methoden reduziert werden, um eine flüssige Wiedergabe von Videos auf schwächerer Hardware zu ermöglichen. Dazu gehörte unter anderem die Verbesserung des Einlesesystems für Videodaten. Die Lösung dafür war, die Größe der einzulesenden Datenblöcke zu vergrößern und die überschüssigen gelesenen Daten zwischenzuspeichern. Allerdings war auch mit dieser Verbesserung kein flüssiges Abspielen auf der Referenzplattform Raspberry Pi 3B möglich. Somit waren weitere Optimierungsversuche nötig, die eine Reduktion der Programmausgaben, die Reduktion der Zeitintervalle bei der RTP-Verarbeitung und die Verwendung einer eigenen Zeichenfunktion umfassten. Jedoch zeigte lediglich die Verringerung der Programmausgaben auf der Konsole Erfolg. Die beiden anderen Versuche führten zwar auf dem Laptop zu einer Verbesserung. Auf dem Raspberry Pi wurde allerdings eine Verschlechterung der Gesamtleistung der Software festgestellt. Bei der Herstellung der Kompatibilität des Programmes mit dem VLC Media Player zeigte sich, dass die Implementierung des RTSP-Protokolles im Projekt einige Fehler beinhaltete. Dazu gehörten neben unvollständigen Antworten auf Clientanfragen auch fehlerhafte

Angaben im Headerfeld der RTP-Pakete. Mit der Behebung dieser Mängel konnte die beidseitige Verwendung des Projektes mit dem VLC Media Player als Server bzw. Client ermöglicht werden.

Um die Untersuchungen zu Verschlüsselungskonzepten bei der Übertragung von JPEG-Bildern per RTP vorzubereiten, wurden sowohl die Anforderungen an solche Konzepte sowie die Struktur des bestehenden Projektes analysiert. Dabei zeigte sich, dass die Software einen Großteil ihrer Funktionalität in wenigen Klassen vereint. Dadurch war der Quellcode nicht sehr übersichtlich und auch klare Schnittstellen zwischen den einzelnen Funktionsbestandteilen waren nicht vorhanden. Als Folge daraus wurde ein Konzept für eine vollständige Neustrukturierung des Projektes entworfen. Aus Gründen der benötigten Zeit und des Umfangs dieser Aufgabe wurde der Entwurf auf die Restrukturierung der RTP-Funktionalität reduziert und diese umgesetzt.

Ein generisches Verschlüsselungsverfahren wurde mit dem Protokoll SRTP vorgestellt. Als kryptografisches Framework für die Verschlüsselung und Authentifikation von RTP-Pakten stellt es ein umfassendes Konzept zur Sicherung solcher Pakete bereit. Ein Verfahren zum initialen Schlüsselaustausch stellt SRTP nicht bereit. Die Untersuchung und Implementierung eines solchen ist Ansatzpunkt für zukünftige Arbeiten. Im Anschluss an die Umsetzung von SRTP im Projekt wurde die implementierte Funktionalität getestet. Dabei war in den Programmen Server und Client die Verschlüsselung mit SRTP manuell zu wählen. Die Ursache dafür ist die nicht vorhandene Sitzungsinitialisierung. Das Testen mit einem externen Programm wie dem VLC Media Player war nicht möglich, da dieses die gefragte Funktionalität nicht unterstützt.

Als Alternative zur generischen Verschlüsselung wurde zuletzt auch die JPEG-spezifische Verschlüsselung untersucht. Es zeigte sich, dass dabei entsprechend der Anordnung bezüglich der Kompression verschiedene Ansätze existieren. Die Grundlage eines eigenen Entwurfes bildete dabei der Ansatz, die Verschlüsselung nach Vollenden der Kompression durchzuführen. Auf diese Weise konnte mit bereits komprimierten Videodateien gearbeitet werden. Der eigene Verschlüsselungsentwurf beruhte dabei auf der Verschlüsselung der Quantisierungstabellen. Ein Verfahren für die Authentifizierung der Daten war dabei nicht vorgesehen. Dafür würde wiederum ein eigenes Verfahren benötigt. Zusammen mit der Implementierung der Authentifikation des SRTP im Projekt stellt die Untersuchung solcher Verfahren eine weitere Möglichkeit dar, die in dieser Arbeit vorgestellten Ergebnisse zu ergänzen und zu erweitern. Die Analyse des eigenen Verschlüsselungsverfahrens zeigte im Anschluss die Unsicherheit desselben auf. Trotz der Einhaltung der meisten Anforderungen an solche Verschlüsselungen führte die Nichtbeachtung der wahrnehmungsbezogenen Sicherheit dazu, dass das entworfene Verschlüsselungsverfahren zu brechen ist. Praktisch zeigte sich das auch in einem erfolgreich durchgeführten Angriff auf das Verfahren. Abschließend wurde das von Kobayashi und Kiya erarbeitete Verfahren [11] vorgestellt, welches dem in dieser Arbeit entworfenen ähnlich ist. Statt den Quantisierungstabellen werden jedoch Teile der entropiecodierten Daten verschlüsselt. Demzufolge ist auch mit der Implementierung des Verfahrens von Kobayashi und Kiya oder mit dem Entwurf und der Implementierung einer sichereren JPEG-spezifischen Verschlüsselung die Fortsetzung der hier vorgestellten Arbeit möglich.

Literaturverzeichnis

[1] Apple Developer, „QuickTime File Format Specification," Apple Computer, Inc., Specification, Sep. 2016. Adresse: https://developer.apple.com/standards/classic-quicktime/ (besucht am 14. 05. 2021).

[2] M. Baugher, D. McGrew, M. Naslund, E. Carrara und K. Norrman, „The Secure Real-time Transport Protocol (SRTP)," RFC Editor, RFC 3711, März 2004. Adresse: https://rfc-editor.org/rfc/rfc3711.

[3] L. Berc, W. Fenner, R. Frederick, S. McCanne und P. Stewart, „RTP Payload Format for JPEG-compressed Video," RFC Editor, RFC 2435, Okt. 1998. Adresse: https://rfc-editor.org/rfc/rfc2435.txt.

[4] BigBlueButton Inc., *BigBlueButton: BigBlueButton*, 2021. Adresse: https://docs.bigbluebutton.org/ (besucht am 06. 07. 2021).

[5] R. Finlayson, *LIVE555 Streaming Media Changelog*, Aug. 2021. Adresse: http://www.live555.com/liveMedia/public/changelog.txt (besucht am 09. 08. 2021).

[6] N. Freed und N. Borenstein, „Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies," RFC Editor, RFC 2045, Nov. 1996. Adresse: https://rfc-editor.org/rfc/rfc2045.

[7] ——, „Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types," RFC Editor, RFC 2046, Nov. 1996. Adresse: https://rfc-editor.org/rfc/rfc2046.

[8] ISO/IEC 10918-1:1994, „Information technology – Digital compression and coding of continuous-tone still images: Requirements and guidelines," International Organization for Standardization, Genf, CH, Standard, Feb. 1994. Adresse: https://www.iso.org/standard/18902.html.

[9] ISO/IEC 10918-5:2013, „Information technology – Digital compression and coding of continuous-tone still images: JPEG File Interchange Format (JFIF)," International Organization for Standardization, Genf, CH, Standard, Mai 2013. Adresse: https://www.iso.org/standard/54989.html.

[10] ITU-R BT.601-7, „Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios," International Telecommunication Union, Genf, CH, Recommendation, März 2011. Adresse: https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.601-7-201103-I!!PDF-E.pdf.

[11] H. Kobayashi und H. Kiya, „Bitstream-Based JPEG Image Encryption with File-Size Preserving," in *2018 IEEE 7th Global Conference on Consumer Electronics (GCCE)*, 2018, S. 384–387. DOI: 10.1109/GCCE.2018.8574605.

[12] H. Krawczyk, M. Bellare und R. Canetti, „HMAC: Keyed-Hashing for Message Authentication," RFC Editor, RFC 2104, Feb. 1997. Adresse: https://rfc-editor.org/rfc/rfc2104.

[13] K. Kropf, toby63, S. I. Corretgé, P. Tiedtke und D. Dascalescu, *Architecture – Jitsi Meet Handbook*, Juni 2021. Adresse: https://jitsi.github.io/handbook/docs/architecture (besucht am 06. 07. 2021).

[14] *Lange Nacht der Wissenschaften @HTW Dresden*, Aug. 2019. Adresse: https://www.youtube.com/watch?v=7_zdIClIhJw (besucht am 10. 08. 2021).

[15] P. Li und K.-T. Lo, „Survey on JPEG compatible joint image compression and encryption algorithms," *IET Signal Processing*, Jg. 14, Nr. 8, S. 475–488, 2020. DOI: https://doi.org/10.1049/iet-spr.2019.0276.

[16] S. Lian, *Multimedia Content Encryption: Techniques and Applications*. Auerbach Publications, 2009.

[17] OpenDML AVI M-JPEG File Format Subcommittee, „OpenDML AVI File Format Extensions," Matrox Electronic Systems Ltd., Techn. Ber., Sep. 1997. Adresse: http://www.jmcgowan.com/odmlff2.pdf.

[18] C. Paar und J. Pelzl, *Kryptografie verständlich - Ein Lehrbuch für Studierende und Anwender*. Springer Vieweg, 2016.

[19] K. Pohl, „Demonstration des RTSP-Videostreamings mittels VLC-Player und einer eigenen Implementierung," Diplomarbeit, Hochschule für Technik und Wirtschaft Dresden, Apr. 2015.

[20] Raspberry Pi Foandation, *Operating system images*, Mai 2021. Adresse: https://www.raspberrypi.org/software/operating-systems/ (besucht am 09. 06. 2021).

[21] D. Salomon und G. Motta, *Handbook of Data Compression*, 5. Aufl. Springer-Verlag London Limited, 2010.

[22] H. Schulzrinne und S. Casner, „RTP Profile for Audio and Video Conferences with Minimal Control," RFC Editor, RFC 3551, Juli 2003. Adresse: https://rfc-editor.org/rfc/rfc3551.

[23] H. Schulzrinne, S. Casner, R. Frederick und V. Jacobson, „RTP: A Transport Protocol for Real-Time Applications," RFC Editor, RFC 3550, Juli 2003. Adresse: https://rfc-editor.org/rfc/rfc3550.

[24] H. Schulzrinne, A. Rao und R. Lanphier, „Real Time Streaming Protocol (RTSP)," RFC Editor, RFC 2326, Apr. 1998. Adresse: https://rfc-editor.org/rfc/rfc2326.

[25] H. Schulzrinne, A. Rao, R.Lanphier, M. Westerlund und M. Stiemerling, „Real-Time Streaming Protocol Version 2.0," RFC Editor, RFC 7826, Dez. 2016. Adresse: https://rfc-editor.org/rfc/rfc7826.

[26] H. G. Semerjian, „Announcing Approval of the Withdrawal of Federal Information Processing Standard (FIPS) 46-3, Data Encryption Standard (DES); FIPS 74, Guidelines for Implementing and Using the NBS Data Encryption Standard; and FIPS 81, DES Modes of Operation," *Federal Register*, Jg. 70, Nr. 96, S. 28 907–28 908, Mai 2005. Adresse: https://www.federalregister.gov/documents/2005/05/19/05-9945/announcing-approval-of-the-withdrawal-of-federal-information-processing-standard-fips-46-3-data (besucht am 02. 07. 2021).

[27] T. Strutz, *Bilddatenkompression, Grandlagen, Codierung, Wavelets, JPEG, MPEG, H.264*, 4. Aufl. Wiesbaden: Vieweg+Teubner | GWV Fachverlage GmbH, 2009.

[28] *Studium der #Informatik @HTW Dresden*, Juni 2017. Adresse: https://www.youtube.com/watch?v=_U261jTthJE (besucht am 10. 08. 2021).

[29] VideoLAN, *Official download of VLC media player, the best Open Source player*. Adresse: https://www.videolan.org/vlc/ (besucht am 01. 06. 2021).

[30] Y.-K. Wang, R. Even, T. Kristensen und R. Jesup, „RTP Payload Format for H.264 Video," RFC Editor, RFC 6184, Mai 2011. Adresse: https://rfc-editor.org/rfc/rfc6184.

[31] Y.-K. Wang, Y. Sanchez, T. Schierl, S. Wenger und M. M. Hannuksela, „RTP Payload Format for High Efficiency Video Coding (HEVC)," RFC Editor, RFC 7798, März 2016. Adresse: https://rfc-editor.org/rfc/rfc7798.

[32] S. White, K. Sharkey, D. Coulter, D. Batchelor, M. Jacobs und M. Satran, *AVI RIFF File Reference*, Mai 2018. Adresse: https://docs.microsoft.com/en-us/windows/win32/directshow/avi-riff-file-reference (besucht am 04. 05. 2021).

[33] Wikipedia Contributors, *Bit numbering — Wikipedia, The Free Encyclopedia*, 2021. Adresse: https://en.wikipedia.org/w/index.php?title=Bit_numbering&oldid=1036783503 (besucht am 13. 08. 2021).

[34] E. Zschorlich, „Vergleich von Video-Streaming-Verfahren unter besonderer Berücksichtigung des Fehlerschutzes und Implementierung eines ausgewählten Verfahrens," Diplomarbeit, Hochschule für Technik und Wirtschaft Dresden, Aug. 2017.

A. Ausgaben des JPEG-Analyseprogramms

```
$ java -cp bin/ MJpegRtpCheck data/lndw_mjpeg_640x360.mov
Baseline DCT sequential: true
Huffman entropy coding: true
Number of SOS segments: 1
interleaved: true
width: 640 px
height: 360 px
sampe precision: 8 bit
pixel aspect ratio: 1.0
number of components in frame: 3
number of quantization tables: 1
number of huffman tables: 4
subsampling: 4:2:0
```

Abbildung A.1.: Ausgabe der Metadaten einer Videodatei mit dem JPEG-Dateianalysator

```
$ java -cp bin/ MJpegRtpCheck -c data/lndw_vlc_640x360.mjpeg
[ PASSED ] Baseline DCT sequential
[ PASSED ] Huffman entropy coding
[ PASSED ] single-scan, interleaved
[ PASSED ] width <= 2040
[ PASSED ] height <= 2040
[ PASSED ] sample precision 8 bit
[ FAILED ] pixel aspect ratio = 1.0
[ PASSED ] number of components in frame = 3
[ PASSED ] number of quantization tables = [1, 2]
[ PASSED ] number of huffman tables = [1, 4]
[ FAILED ] Huffman tables are conform with Annex K.3
[ PASSED ] subsampling 4:2:2 or 4:2:0
====================================
JPEG is not conformant to RFC 2435
```

Abbildung A.2.: Ausgabe eines fehlgeschlagenen Tests der Überprüfung auf die Anforderungen von RFC 2435 [3] einer Videodatei

```
$ java -cp bin/ MJpegRtpCheck -f -c data/lndw_mjpeg_640x360.mov
====================================
[ PASSED ] All frames share metadata

[ PASSED ] Baseline DCT sequential
[ PASSED ] Huffman entropy coding
[ PASSED ] single-scan, interleaved
[ PASSED ] width <= 2040
[ PASSED ] height <= 2040
[ PASSED ] sample precision 8 bit
[ PASSED ] pixel aspect ratio = 1.0
[ PASSED ] number of components in frame = 3
[ PASSED ] number of quantization tables = [1, 2]
[ PASSED ] number of huffman tables = [1, 4]
[ PASSED ] Huffman tables are conform with Annex K.3
[ PASSED ] subsampling 4:2:2 or 4:2:0
====================================
JPEG is conformant to RFC 2435
```

Abbildung A.3.: Ausgabe eines erfolgreichen vollständigen Tests der Überprüfung auf die Anforderungen von RFC 2435 [3] einer Videodatei

B. Ausführungszeiten einzelner Verarbeitungsschritte bei Optimierungsmaßnahmen

Um den Erfolg der jeweiligen Optimierungsmaßnahmen zu beurteilen, wurden neben der Abspielrate des Videos im Client und der Anzahl verlorener Pakete auch die Abarbeitungszeiten einzelner Ausführungsschritte betrachtet. Für die Erhebung dieser Zeiten wurde das Programm VisualVM in der Version 2.0.7 verwendet. Es stellt eine grafische Schnittstelle für die im Java Development Kit (JDK) mitgelieferten Analysewerkzeuge mit. Dazu gehören auch „Sampling Profiler“. Diese Profiler zeichnen die Ausführungszeiten der Methoden eines Javaprogrammes auf. So kann beispielsweise festgestellt werden, welche Funktionalitäten einer Software besonders lange für ihre Ausführung benötigen. Auf diese Weise können die ermittelten Methoden genauer analysiert und anschließend optimiert werden, was zu einer Verbesserung der Gesamtleistung des Programmes führt.

Im „RTSP-Streaming“-Projekt wurden jeweils die Ausführungszeiten der Methoden des Clients für die Übertragung eines Videos betrachtet. Dabei war die Fehlerkorrektur mittels FEC eingeschaltet. Der Server erstellte FEC-Pakete für eine Gruppengröße von zwei Medienpaketen und simulierte eine Kanalverlustrate von 0 Prozent. Als Testvideos dienten dabei zwei Videos der HTWDD. Dabei handelt es sich um die Videos zum Studium der Informatik („SdI“) [28] und zur Langen Nacht der Wissenschaften 2019 („LNdW“) [14]. Für jeden Optimierungsversuch wurden beide Videos pro Hardwareplattform jeweils einmal vollständig übertragen und abgespielt. Die verwendeten Plattformen sind ein Raspberry Pi 3B (RPi 3B) und ein handelsüblicher Laptop und wurden bereits in Kapitel 5 erwähnt. Ein Nachteil in der Verwendung der bereits genannten Profilersoftware ist, dass diese selbst eine gewisse Prozessorauslastung erzeugt. Auf schwächerer Hardware kann das zu einer Verstärkung der beobachteten Effekte führen. Außerdem gibt es für jeden einzelnen Übertragungsvorgang Varianzen, die durch andere zeitgleich laufende Prozesse und die Priorisierung der Abarbeitung derselben entstehen. Deshalb müssen diese unbedingt mit in die Analyse der aufgezeichneten Daten einbezogen werden.

Damit ein Vergleich zur Leistung der Software vor den jeweiligen Optimierungsversuchen gezogen werden kann, wurden die in Tabelle B.1 dargestellten Daten aufgezeichnet. Bereits darin lässt sich ein deutlicher Unterschied zwischen den Abarbeitungszeiten auf dem Raspberry Pi und dem Laptop erkennen. Auf dem Laptop können die erforderlichen Berechnungen deutlich schneller durchgeführt werden. Das deckt sich jedoch mit der Beobachtung, dass Probleme mit der Ausführung des Projektes nur auf schwächerer Hardware auftreten.

Video (Gesamtzahl Pakete)	**„SdI“ (2804)**		**„LNdW“ (1793)**	
	RPi 3B	Laptop	RPi 3B	Laptop
`ImageIcon()` [ms]	68673	7582	68157	4806
`setStatistics()` [ms]	77640	3056	62991	2623
`combineToOneImage()` [ms]	2931	783	2875	197
`getNextRtpList()` [ms]	6244	202	4062	0
`println()` [ms]	11074	297	7773	100
`receive()` [ms]	16398	49922	9972	31911
`rcvRtpPacket()` [ms]	5129	404	3724	289
verlorene Pakete [ms]	1297	0	757	0

Tabelle B.1.: Ausführungszeiten ausgewählter Methoden vor den Optimierungsmaßnahmen

Video (Gesamtzahl Pakete)	**„SdI“ (2804)**		**„LNdW“ (1793)**	
	RPi 3B	Laptop	RPi 3B	Laptop
`ImageIcon()` [ms]	61420	7379	42598	5308
`setStatistics()` [ms]	48194	2933	33054	2443
`combineToOneImage()` [ms]	2722	394	1951	184
`getNextRtpList()` [ms]	1532	195	1995	195
`println()` [ms]	5235	490	4270	294
`receive()` [ms]	13425	50342	10365	32717
`rcvRtpPacket()` [ms]	3024	303	2772	197
verlorene Pakete [ms]	538	0	465	1

Tabelle B.2.: Ausführungszeiten ausgewählter Methoden nach der Verbesserung des Einlesens von Videodaten

Die Daten erster Optimierungsversuche sind in den Tabellen B.2 und B.3 zu sehen. Während sich die Leistung auf dem Raspberry Pi deutlich verbessert, wie an den kürzeren Ausführungszeiten zu erkennen ist, muss darauf geachtet werden, dass sich die Leistung auf dem Laptop ebenfalls verbessert oder zumindest nicht deutlich verschlechtert.

Die beiden letzten Optimierungsmaßnahmen, deren Profilerdaten in B.4 bzw. B.5 zu sehen sind, brachten keinen eindeutigen Erfolg. Während sich die Ausführungszeiten der Methoden auf dem Laptop deutlich verkürzten, stiegen diese auf dem Raspberry Pi deutlich an. Für die Verbesserung der Leistung des Projektes auf schwächerer Hardware waren diese deshalb nicht erfolgreich.

Video (Gesamtzahl Pakete)	**„SdI“ (2804)**		**„LNdW“ (1793)**	
	RPi 3B	Laptop	RPi 3B	Laptop
`ImageIcon()` [ms]	49908	9885	29213	7452
`setStatistics()` [ms]	6995	969	7358	1267
`combineToOneImage()` [ms]	2849	574	2316	774
`getNextRtpList()` [ms]	537	191	596	0
`receive()` [ms]	20969	53728	15870	33401
`rcvRtpPacket()` [ms]	663	297	820	192
verlorene Pakete [ms]	40	0	46	0

Tabelle B.3.: Ausführungszeiten ausgewählter Methoden nach der Einführung eines Loggers zur Reduzierung der Kommandozeilenausgaben

Video (Gesamtzahl Pakete)	**„SdI“ (2804)**		**„LNdW“ (1793)**	
	RPi 3B	Laptop	RPi 3B	Laptop
`ImageIcon()` [ms]	46323	10066	30540	7184
`setStatistics()` [ms]	7068	1267	6321	1387
`combineToOneImage()` [ms]	2928	1264	1824	290
`getNextRtpList()` [ms]	997	102	874	0
`receive()` [ms]	4389	4183	3253	3094
`rcvRtpPacket()` [ms]	927	197	602	0
verlorene Pakete [ms]	60	0	55	0

Tabelle B.4.: Ausführungszeiten ausgewählter Methoden nach der Anpassung der Übertragungszeiten für RTP-Pakete

Video (Gesamtzahl Pakete)	**„SdI“ (2804)**		**„LNdW“ (1793)**	
	RPi 3B	Laptop	RPi 3B	Laptop
`paint()`	180558	9570	104393	4498
`setStatistics()`	12581	1290	9937	1402
`getNextRtpList()`	2807	100	1215	0
`combineToOneImage()`	1.531	512	1553	509
`receive()`	7412	4647	4347	4049
`rcvRtpPacket()`	370	190	518	387
verlorene Pakete	1142	0	656	0

Tabelle B.5.: Ausführungszeiten ausgewählter Methoden nach der Einführung einer eigenen Zeichenfunktion

Selbstständigkeitserklärung

Ich versichere, dass ich die Diplomarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Dresden, den 30.08.2021

Emanuel Günther