Technische Universität Braunschweig
Institut für Betriebssysteme und Rechnerverbund

Diplomarbeit

Design eines leicht-gewichtigen Multimedia-Gateways für mobile Clients

von
cand. inform. Jens Zechlin

Aufgabenstellung und Betreuung:

Prof. Dr. L. Wolf

Braunschweig, den 31. März 2003

Erklärung

Ich versichere, die vorliegende Arbeit selbstständig und nur unter Benutzung der angegebenen Hilfsmittel angefertigt zu haben.

Braunschweig, den 31. März 2003

Kurzfassung

Digitales Video erhält immer mehr Einzug in das Internet. Fortschritte in der Video- und Audiocodierung machen Multimediaübertragungen über einen großen Bereich verschiedener Netzwerke möglich. Durch den Trend zu immer größerer Mobilität wächst jedoch auch die Heterogenität des Internets. Neue Techniken für drahtlose Netzwerke und kompaktere Geräte lassen die Vision des überall verfügbaren Internets immer greifbarer werden. Bei der Multimediaübertragung stoßen diese mobilen Netze und Geräte jedoch schnell an ihre Grenzen. Geringe Bandbreiten und relativ geringe Hardwareausstattungen beeinträchtigen häufig den Multimediagenuss mit mobilen Geräten. Eine Lösung für dieses Problem liegt in der Anpassung der Daten an das Endgerät. Erreicht werden kann dies durch eine Umwandlung der Medienströme vor der Übertragung an den Client, was als Transcoding bezeichnet wird.
In dieser Arbeit wurde ein Multimedia-Gateway entwickelt, das mobilen Geräten ein Transcoding von Medienströmen zur Verfügung stellt. Als Grundlage diente eine freie Kommunikations-Umgebung zur Übertragung digitaler Medienströme. Diese Implementation wurde so erweitert, dass die Daten durch verschiedene Transcodingtechniken an die Bedürfnisse mobiler Geräte angepasst werden können. Ein Vorteil dieses Gateways liegt insbesondere in seiner leichten Erweiterbarkeit, die durch ein dynamisches Laden von Modulen zur Laufzeit des Gateways erreicht wurde.

Abstract

Digital video plays an increasingly important role on the Internet. Advances in video and audiocoding make multimedia streaming across a wide range of different networks possible. This trend to more and more mobility leads also to an incrasing heterogeneity of the Internet itself. New technologies for wireless networks and more compact devices let the vision of an unlimited availability of the Internet become more and more feasible. With regard to multimedia streaming, however, these mobile networks and devices quickly reach their limits. Low bandwidths and relatively slight hardware equipment frequently limit the pleasures of multimedia streaming with mobile devices. A solution to this problem is adapting the data itself to the requirements of the mobile device. This can be achieved by a media transcoding before streaming the data to the client.
In this thesis, a multimedia gateway was developed that makes transcoding of multimedia streams available for mobile clients. It is based on a free communication environment for streaming digital media. This existing implementation was extended in terms of allowing data transcoding with different transcoding methods depending on the needs of mobile devices. An advantage of this gateway lies particularly in its easy way to be extended, which was reached by dynamic loading of modules at runtime of the gateway.

TECHNISCHE UNIVERSITÄT BRAUNSCHWEIG
INSTITUT FÜR BETRIEBSSYSTEME UND RECHNERVERBUND
Prof. Dr. S. Fischer · Prof. Dr. L. Wolf

IBR • TU Braunschweig • Mühlenpfordtstr. 23 • D-38106 Braunschweig

Prof. Dr. Lars Wolf
(Geschäftsführender Leiter)

Telefon: +49 - 531-391-3288
Telefax: +49 - 531-391-5936
Email: wolf@ibr.cs.tu-bs.de
http://www.ibr.cs.tu-bs.de

Braunschweig, den 30. September 2002

Diplomarbeitsaufgabenstellung

Design eines leichtgewichtigen Multimedia-Gateways für mobile Clients

Bearbeiter: Herr Jens Zechlin, Matr.-Nr. 2509488, Email: j.zechlin@tu-bs.de

Bei verteilten multimedialen Anwendungen müssen mehrere Ströme audiovisueller Medien über Kommunikationsnetze übertragen werden. Aufgrund der hohen Datenraten solcher Audio- und Videoströme, ergeben sich hohe Anforderungen an die Netze und die in diesen eingesetzten Komponenten als auch an die Endgeräte. Gerade für mobile, drahtlose angebundene Endgeräte sind jedoch diese Voraussetzungen nicht immer gegeben. So können sowohl die Endgeräteressourcen (CPU-Leistung, Speicher, …) als auch die Netzanbindung nicht ausreichend sein, um eine Multimedia-Präsentation in ihrer Originalqualität handhaben und darstellen zu können. Ein Ansatz zum Umgang mit derartigen Problemen besteht darin, dass die Medienströme während der Übertragung umcodiert werden, um so eine Reduktion der Ressourceanforderungen zu erzielen oder ein anderes Format zu erzeugen, welches vom Endgerät verarbeitet werden kann.

Aufgabenstellung

Im Rahmen dieser Diplomarbeit soll ein Multimedia-Gateway unter Linux entstehen, welches in der Lage ist, RTP-basierte Medienströme (Audio- und Video) in Echtzeit in ein gewünschtes Format umzuwandeln bzw. die Qualität eines Stromes zu verringern. Die vom Gateway unterstützten Formate sollen nicht starr festgelegt sein, sondern als Module eingebunden werden können, so dass für spätere Erweiterungen der Medienformate keine Änderungen am Gateway selbst notwendig sind.

Um eine möglichst hohe Effizienz bei möglichst geringem Ressourcebedarf erzielen zu können, sollten bei der Umcodierung nur möglichst wenige Schritte der Gesamtcodierung durchlaufen werden. Dies bedeutet, dass es für die unterstützen Formate wenn machbar möglich sein sollte, den Vorgang der Decodierung möglichst frühzeitig zu beenden und das dadurch entstehende Zwischenformat an das Codiermodul für das Zielformat zu übergeben. (ggf. ist eine Umwandlung oder Anpassung des Zwischenformats notwendig) oder eine Transcodierung auf einem codierten Format durchzuführen. Somit muss die Schnittstelle für die Codier- und Decodiermodule so universell ausgelegt und implementiert werden, dass möglichst keine Medienformate ausgeschlossen und dass für geeignete Formate eine schnelle Transcodierung möglich ist. In der Arbeit soll betrachtet werden, ob statt einer eher mehrere Schnittstellen (bzw. Teile in verschiedener Tiefe) sinnvoll sind.

Damit das Gateway die Ströme in geeigneten Formaten erzeugen und an das Endgerät weiterschicken kann, muss es über Kenntnisse verfügen, welche Formate und welche Ressourcenanforderungen geeignet sind.

Daher sind Überlegungen zu einer geeigneten Signalisierung solcher Informationen an das Gateway anzustellen. Wünschenswert wäre eine größtmögliche Transparenz, die Machbarkeit muss im Rahmen der Arbeit untersucht werden.

In erster Linie soll das Gateway zwar mobilen Clients dienen, jedoch sollte der Nutzung des Gateways durch stationäre Clients nichts im Wege stehen. Um die Einsatzbarkeit nicht nur auf leistungsfähige Server zu beschränken, sollte das Gateway möglichst in der Lage, auch mit relativ geringen Resourcen gute Ergebnisse zu liefern.

Erweiterungsmöglichkeiten bestehen bspw. bzgl. Untersuchungen, ob und in welchem Umfang mehrere Gateways miteinander kommunizieren können, um z.B. eine Lastverteilung zu erreichen und dass ein Weiterleiten des aktuellen Vorgangs an einen anderen Gateway möglich ist, ohne die Übertragung zum Client zu unterbrechen.
Andere optionale Arbeitspunkte beziehen sich auf Caching-Ansätze. So wäre neben der Umwandlungsfunktionalität eine Möglichkeit zur Zwischenpufferung von Medienstromteilen im Gateway sinnvoll, bspw. um Startverzögerungen von Clients zu verringern.

Die erstellte Software ist zu Testen und zu evaluieren, um ihre Eigenschaften sowie ihre Leistungsfähigkeit zu ermitteln.

Die Hinweise zur Durchführung von Studien- und Diplomarbeiten am IBR sind zu beachten.

Laufzeit: 6 Monate

Aufgabenstellung und Betreuung:
Prof. Dr. L. Wolf (Prof. Dr. L. Wolf)

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Seit ca. zehn Jahren ist digitales Video auf dem Weg, analoges Video nach und nach abzulösen. Die Zahl der digitalen Videosysteme wächst stetig und immer mehr analoge Systeme werden durch digitale ersetzt. Digital Versatile Disk (DVD), Video-CD, digitale Videorecorder, Video on Demand, digitale Videokonferenzen und E-Learning sind nur einige Beispiele hierfür. Eine Schlüsseltechnologie für den Erfolg des digitalen Video ist die Videocodierung. Die Videocodierung erlaubt eine effiziente Speicherung und Übertragung digitaler Videodaten, indem die Datenmenge, die zur Darstellung einer Videosequenz notwendig ist, drastisch reduziert wird. Die Verteilung dieser Daten erfolgt heutzutage häufig noch auf traditionellen Wegen. Mehr und mehr entstehen aber auch Angebote, digitales Video z. B. über das Internet oder per digitaler TV-Übertragung an die Nutzer zu liefern. Anders als beim Digital-TV ist das Internet ein sehr heterogenes Netz, bei dem die verschiedenen Zugangsmöglichkeiten und -geräte scheinbar unbegrenzt sind. Neben den klassischen kabelgebundenen Zugangswegen entstehen immer neue Technologien, wie z. B. W-LAN oder Bluetooth, die auch einen drahtlosen Zugang ermöglichen. Zusätzlich führt die fortschreitende Verknüpfung verschiedener Kommunikationsnetze mit dem Internet zu einer immer stärkeren Heterogenität des Internets. Im Gegensatz zu traditionellen, statischen Internetdaten, wie HTML-Seiten oder Grafiken, ist es bei zeitkritischen audiovisuellen Daten notwendig, eine gewisse Datenrate während der gesamten Übertragung einzuhalten. Bei digitalen Videodaten wird diese Datenrate bereits bei der Erzeugung durch die Videocodierung festgelegt. Gerade dieses Vorgehen ist aber aufgrund der Heterogenität des Internets wenig sinnvoll. Für Besitzer eines breitbandigen Internetzugangs stellt eine hohe Datenrate kein Problem dar. Beschränkt sich der Zugang jedoch auf ein Funknetzwerk oder gar auf eine ISDN-Leitung, können Videos mit hohen Datenraten nicht störungsfrei auf dem Client dargestellt werden. Neben der verfügbaren Bandbreite spielt aber auch das Codierungsformat eine wichtige Rolle. Denn nicht jeder Client ist in der Lage, jedes Medienformat darzustellen. Aufgrund immer kompakterer internetfähiger Geräte, wie z. B. PDA's oder Handys, ist auch die Auflösung eines Videos von Bedeutung. So ist es z. B. für einen PDA praktisch unmöglich, einen Videostrom zu decodieren und gleichzeitig an die geringe Auflösung des Displays anzupassen. Um den Bedürfnissen aller Nutzer gerecht zu werden, müssten die Anbieter digitaler Videos diese also

mit verschiedenen Datenraten, in verschiedenen Formaten und darüber hinaus in verschiedenen Auflösungen anbieten. Dies würde zu einer relativ hohen Speicherplatzanforderung führen, was wiederum höhere Kosten für die Anbieter verursachen würde. Alles in allem müssen also Wege gefunden werden, digitale Videos sowohl mit verschiedenen Datenraten als auch in verschiedenen Formaten und Auflösungen effizient anbieten zu können.

Ein Ansatz zur Bereitstellung digitaler Videos mit verschiedenen Datenraten besteht darin, die Daten in mehrere Schichten aufzuteilen. Bei dieser Vorgehensweise, die als Layered Videocoding bezeichnet wird, besteht eine Videosequenz aus einer Basisschicht sowie weiteren Detailschichten. Die Basisschicht enthält die Daten mit einer so geringen Datenrate, dass sie selbst bei sehr geringen Bandbreiten noch vernünftig dargestellt werden können. Für Clients mit höheren Bandbreiten enthalten die Detailschichten zusätzliche Detailinformationen, die die Qualität des digitalen Videos erhöhen können. Je mehr Schichten ein Client empfängt und darstellt, desto höher ist also die Qualität des dargestellten Videos. Aber auch hierbei entsteht ein erhöhter Speicherbedarf, da durch die verschiedenen Schichten ein gewisser Overhead hinzugefügt wird. Zusätzlich werden hierbei auch an den Client höhere Anforderungen gestellt, denn dieser muss in der Lage sein, die Videosequenz aus den einzelnen Schichten wieder zusammenzusetzen. Auch die Notwendigkeit, verschiedene Datenformate und Auflösungen anzubieten, wird durch diesen Ansatz nicht gelöst.

Eine Lösung des Gesamtproblems stellt eine weitere Schlüsseltechnologie des digitalen Videos – das Transcoding – zur Verfügung. Unter Videotranscoding versteht man den Vorgang der Umwandlung digitaler Videodaten aus einem komprimierten Format in ein anderes oder das gleiche komprimierte Format mit veränderten Codierungsparametern. Ein Gerät, das solch eine Transcodierung digitaler Videodaten durchführen kann, wird als Videotranscoder oder einfach nur als Transcoder bezeichnet. Ein Transcoder ist also in der Lage, digitale Videodaten auf ihrem Weg zum Client umzuwandeln, um dessen Ansprüchen gerecht zu werden. Eine sinnvolle Möglichkeit zur Platzierung des Transcoders innerhalb eines Netzwerkes besteht darin, diesen, wie in Abb. 1.1 dargestellt, auf einem Server in der Nähe der Clients zu installieren. Dieser könnte als Multimedia-Gateway dienen und sämtliche Multimedia-Anfragen von Clients bearbeiten.

1.1 Ein Multimedia-Gateway für mobile Clients

Im Rahmen dieser Arbeit wurde ein Multimedia-Gateway für mobile Clients entworfen. Dieses ist in der Lage, Videodaten, die als Medienströme über das Internet übertragen werden, umzuwandeln und somit an die Bedürfnisse eines Clients anzupassen. Beim Entwurf wurde sehr auf eine hohe Flexibilität geachtet,

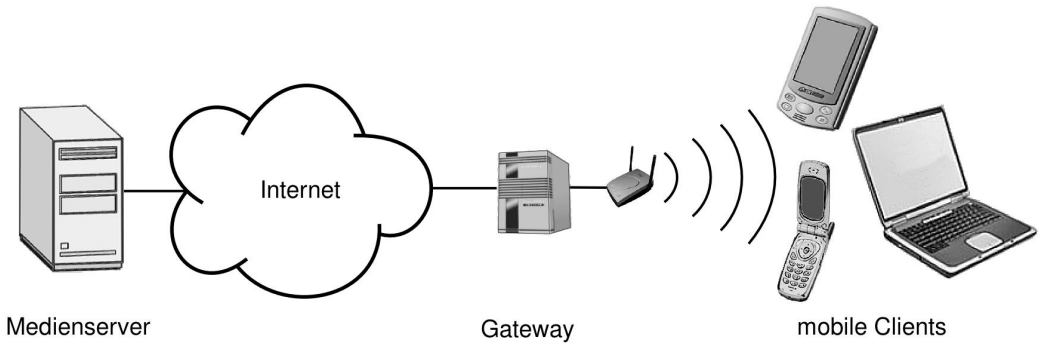


Abbildung 1.1: Ein Multimedia-Gateway für mobile Clients

um eine spätere Erweiterung des Gateways zu erleichtern. Nach Kenntnis des Autors existiert zur Zeit keine verfügbare Implementation eines derartigen RTP-basierten Multimedia-Gateways, dessen Funktionalität durch einfache Module erweitert werden kann. Als Grundlage für die Datenübertragung dient die freie RTSP/RTP-Implementierung Komssys, die so erweitert wurde, dass die in dieser Arbeit vorgestellten Techniken leicht in das Gateway integriert werden können.

1.2 Vorhandene Arbeiten

1.2.1 Unterstützung mobiler Clients

In der Literatur sind nur wenige Arbeiten zu finden, die sich mit Möglichkeiten befassen, mobile Geräte bei Multimedia-Anwendungen zu unterstützen. Ein knapper Überblick über die Möglichkeiten des Transcodings für mobile Clients wurde von A. Vetro in [1] gegeben. Die Umwandlung der Daten findet dort, ebenso wie auch in dieser Arbeit, auf einem einzelnen Netzwerkdevice statt. Im Gegensatz dazu wurde in [2] von B. H. Kang, Z. M. Mao und H. W. So ein Rahmenwerk vorgestellt, in dem mehrere Netzwerkknoten in den Transcodingprozess involviert sind.

1.2.2 Transcoding

Videotranscoding ist keine triviale Aufgabe. Einerseits wurde in die Entwicklung effizienter Videocodierungen sehr viel Energie investiert, wodurch bereits die Decodierung alleine nicht als trivial angesehen werden kann. Andererseits müssen die zeitkritischen Randbedingungen des Transcodingvorgangs eingehalten werden, damit für den Benutzer des Clients bei der Darstellung keine spürbaren Zeitverzögerungen entstehen. Daher ist das Thema des Transcodings digitaler

Videodaten in den letzten Jahren zu einem intensiv erforschten Gebiet geworden. Viele Arbeiten wurden veröffentlicht, die sich jedoch häufig nur mit einem speziellen Teilbereich des Transcodings befassen.

Ein Teil dieser Arbeiten befasst sich z. B. mit der direkten Bearbeitung komprimierter Videodaten. So wurde 1995 von S.-F. Chang und D. G. Messerschmitt in [3] ein Verfahren vorgestellt, das es erlaubt, komprimierte Videodaten ohne vorherige Dekompression zu manipulieren. Dieses Verfahren wurde von N. Merhav und V. Bahaskaran zuletzt 1997 in [4] weiter verbessert, indem bei der Berechnung strukturelle Eigenschaften der beteiligten Abbildungsmatrizen gezielt ausgenutzt wurden. H. Li und H. Shi stellten 1999 in [5] ein weiteres Verfahren vor, das bei der Manipulation nur einen Teil der komprimierten Daten nutzt, was zu einer Einsparung des Rechenaufwand bei sehr geringen Informationsverlusten führt. 2001 wurden in [6, 7] weitere Verfahren vorgestellt, die es erlauben, die Auflösung einzelner Bilder in komprimierter Form zu verändern. Ein anderer Aspekt des Transcodings, der in [8, 9, 10] untersucht wurde, ist die Anpassung von Abhängigkeiten, die zwischen aufeinander folgenden Bildern bestehen.

Neben diesen zum Teil sehr speziellen Arbeiten gibt es weitere, die sich mit konkreten Architekturen für Transcoder beschäftigen. So wurde von P. A. Assuncao und M. Ghanbari 1998 in [11] ein Transcoder zur Requantisierung vorgeschlagen, der gänzlich ohne eine Decodierung der Daten auskommt. Architekturen zur Verringerung der Auflösung der einzelnen Bilder eines Videos wurden 1999 und 2002 in [12, 13, 14] vorgestellt. Einen anderen Ansatz zur Reduktion der Datenrate stellt das Verwerfen einzelner Bilder dar. Hierfür wurde 2001 in [15] und 2002 in [16] ein Frameskipping-Transcoder vorgestellt, der teilweise auf eine Decodierung der Daten verzichtet. Während sich diese Arbeiten im Wesentlichen auf das Transcoding innerhalb eines bestimmten Videoformats befassen, wurden in [17] und [18] Verfahren vorgestellt, die eine Umwandlung des Codierungsformats erlauben.

1.3 Betrachtete Medienformate

Für die Codierung digitaler Videodaten existiert eine Vielzahl verschiedener Datenformate. Bei einigen handelt es sich um Formate, die in Form eines Standards veröffentlicht wurden. Andere hingegen wurden von Firmen entworfen und entwickelt, jedoch nicht als Standard veröffentlicht. Diese Arbeit konzentriert sich auf vier Standardformate, die von der *Moving Pictures Expert Group (MPEG)* und der *International Telecommunication Union (ITU)* entworfen wurden. Die vorgestellten Techniken sind jedoch nicht ausschließlich auf diese Formate beschränkt, sondern gelten auch für Daten, deren Kompression auf gleiche oder ähnliche Verfahren basiert. Insbesondere kann das entworfene Gateway auf ein-

fache Art und Weise an weitere Formate angepasst werden. Audiodaten werden nur kurz betrachtet, da das Gebiet des Audiotranscodings relativ unerforscht ist. Eine Ausweitung dieser Arbeit auf das Umcodieren von Audioströmen war daher nicht möglich. Der Entwurf und die Implementation berücksichtigen aber trotzdem Videoströme, die ebenfalls Audiodaten enthalten, die ggf. herausgefiltert und verworfen werden können.

1.4 Gliederung und Aufbau dieser Arbeit

Die vorliegende Arbeit gliedert sich in ihrem Aufbau im Wesentlichen in drei Teile: Einem grundlegenden Teil zu Formaten und Übertragungsmöglichkeiten multimedialer Daten, einem weiterführenden Teil zu verschiedenen Transcodertechniken und einem Teil zum Design sowie zur Implementation des im Rahmen dieser Arbeit entwickelten Multimedia-Gateways für mobile Clients. Der grundlegende Teil umfasst das Kapitel 2, in dem die betrachteten Videoformate vorgestellt werden und das Kapitel 3, das sich mit der Übertragung von Multimediadaten im Internet befasst. Daran anschließend werden im weiterführenden Teil dieser Arbeit, der aus dem Kapitel 4 besteht, verschiedene Techniken erläutert, die zur Umwandlung von Videodaten eingesetzt werden können. Im dritten Teil werden in Kapitel 5 das Design und der Entwurf des Multimedia-Gateways vorgestellt. Dieser Entwurf wurde im Rahmen dieser Arbeit auch ansatzweise in Software implementiert, worüber Kapitel 6 Auskunft gibt. Zum Ende dieser Arbeit werden in Kapitel 7 die vorgestellten Techniken und Aspekte des entwickelten Gateways zusammengefasst und ein Ausblick auf zukünftige Forschungsmöglichkeiten gegeben.

Für in dieser Arbeit verwendete Abkürzungen und Fachbegriffe befindet sich im Anhang E ein Glossar. Innerhalb des Textes sind diese Begriffe durch eine *kursive Hervorhebung* erkennbar. Bei Begriffen, die in `Schreibmaschinenschrift` gedruckt sind, handelt es sich um Namen von Programmen oder Programmobjekte. Diese sind hauptsächlich in den letzten drei Kapiteln dieser Arbeit zu finden.

2 Medienformate

Kontinuierliche Signale, wie z. B. Audio- und Videosequenzen, müssen für eine digitale Speicherung oder Übertragung zunächst digitalisiert werde. Hierzu werden die analogen Signalströme durch spezielle Hardware zu bestimmten Zeitpunkten abgetastet und die so ermittelten Werte (Abtastwerte) abgespeichert. Damit das analoge Originalsignal aus den Abtastwerten möglichst genau rekonstruiert werden kann, muss die Abtastrate (auch Samplingrate genannt) nach dem Abtasttheorem[1] das Zweifache der maximalen Signalfrequenz betragen. Daher entstehen hierbei sehr große Datenmengen. Eine Minute eines digitalen Audiosignals, wie es z. B. auf einer Musik-CD gespeichert ist belegt ca. 10,5 MB Speicherplatz[2] auf der CD. Eine Minute eines Videosignals mit einer Auflösung von 360×240 Pixel und 30 Bildern pro Sekunde benötigt ca. 445 MB Speicherplatz[3]. Auf einer CD mit einer Kapazität von 650 MB könnten also nur ca. 1,4 Minuten eines Videofilms (mit Stereoton) gespeichert werden. Sollen diese Daten nicht nur gespeichert, sondern auch übertragen werden, wird ein Übertragungsmedium mit einer Bandbreite über 60 MBit/s benötigt.

Dieses einfache Rechenbeispiel zeigt, dass multimediale Daten für eine effiziente Speicherung und Übertragung komprimiert werden müssen. Verlustfreie Kompressionsverfahren, wie sie häufig bei der Datenkompression eingesetzt werden, bieten jedoch keine ausreichenden Kompressionsraten. Da Audio- und Videosignale für die menschlichen Sinnesorgane sehr viele Redundanzen enthalten, kommen stattdessen verlustbehaftete Verfahren zum Einsatz, die diese Redundanzen entfernen. Aufgrund der Wahrnehmung der menschlichen Sinnesorgane führt dies trotz erheblicher Datenreduzierung nur zu einem für den Betrachter kaum feststellbaren Qualitätsverlust.

[1] Das Abtasttheorem nach Shannon und Raabe (1939) besagt, dass bei der Abtastung eines analogen Signals für eine vollständige Rekonstruktion des Signals die Abtastfrequenz mindestens doppelt so hoch sein muss, wie die maximale im Signal vorkommende Frequenz.

[2] Je Kanal: $44.100\ Abtastungen/s \times 16 Bit \times 60 s \approx 5,25 MB$

[3] $360 \times 240 \times 24 Bit \times 30\ Bilder/s \times 60 s \approx 445 MB$

2.1 Videoformate

Die häufigsten im Internet anzutreffenden Videoformate sind H.261 [19], H.263 [20], MPEG-1 [21] und MPEG-2 [22]. Bei diesen vier Formaten handelt es sich um hybride blockorientierte Verfahren, die räumliche und zeitliche Kompression einsetzen. Zur Reduktion der räumlichen Redundanzen wird ein Verfahren eingesetzt, das ursprünglich zur Kompression einzelner Bilder entwickelt wurde und im *JPEG*-Standard zum Einsatz kommt.

2.1.1 Farbraum

Bei der Videocodierung in den obigen Formaten wird nicht der aus der Bildbearbeitung bekannte *RGB*-Farbraum, sondern der YC_rC_b-Farbraum (auch YUV-Farbraum genannt) genutzt. Dieser stellt die drei Farbwerte (Rot, Grün und Blau) als Helligkeit (Luminanz) und zwei Farbdifferenzen (Chrominanz) dar. Eine Umwandlung der Räume kann nach den folgenden Formeln erfolgen:

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B \\ C_r &= R - Y \\ C_b &= B - Y \end{aligned}$$

Untersuchungen haben ergeben, dass die Helligkeitswerte eines Bildes für das menschliche Auge die meisten Informationen tragen. Änderungen der Helligkeit werden also stärker wahrgenommen, als Änderungen in der Farbe. Diese Tatsache kann bereits zur Kompression genutzt werden, indem z. B. nur die Hälfte der verfügbaren Chrominanzinformationen gespeichert werden, ohne dass ein spürbarer Qualitätsverlust auftritt. Diese Technik wird auch bei den hier betrachteten Videocodierungen eingesetzt. MPEG-1 nutzt z. B. ein Verhältnis von 4:2:2 bzgl. der einzelnen Komponenten.

2.1.2 Codierverfahren

Die Erzeugung von Videodaten in den oben genannten Formaten beruht im Wesentlichen auf gleichen oder sehr ähnlichen Verfahren, die im Folgenden kurz dargestellt werden. Da die einzelnen Standards nicht das eigentliche Codierverfahren, sondern nur eine Beschreibung des Codes enthalten, ist es kaum möglich die Funktionsweise, z. B. des MPEG-1-Codierers, zu beschreiben. Vielmehr können nur die grundsätzlichen Mechanismen erläutert werden, die in einem Codierer zum Einsatz kommen.

Die einzelnen Bilder (auch Frames genannt) werden in 8×8 Pixel große Blöcke aufgeteilt, von denen jeweils vier benachbarte Blöcke zu einem Makroblock zusammengefasst werden. Die Blöcke werden im YC_rC_b-Format im Verhältnis 4:2:2 codiert. Das bedeutet, dass ein Makroblock aus 16×16 Luminanzwerten und je 8×8 Chrominanzwerten besteht.

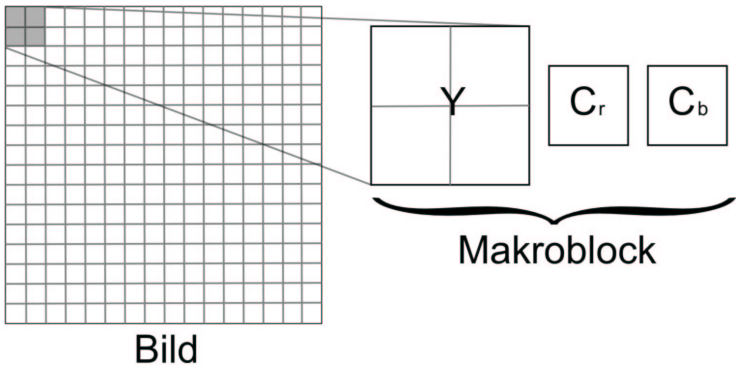


Abbildung 2.1: Zusammensetzung eines Makroblocks aus vier Luminanzblöcken und zwei Chrominanzblöcken

2.1.2.1 Intra-Frame-Codierung

Die Intra-Frame-Codierung dient zur Reduktion räumlicher Redundanzen und ist vom *JPEG*-Verfahren abgeleitet. Hierbei werden die einzelnen Makroblöcke mit Hilfe der *Diskreten Cosinus Transformation (DCT)* in den Frequenzbereich überführt.

$$F(u,v) = \frac{C(u)C(v)}{4} \sum_{j=0}^{7} \sum_{k=0}^{7} f(j,k) cos\left(\frac{(2j+1)u\pi}{16}\right) cos\left(\frac{(2k+1)u\pi}{16}\right)$$

mit

$$C(w) = \begin{cases} \frac{1}{\sqrt{2}} & \text{für } w = 0 \\ 1 & \text{für } w = 1,2,3,4,5,6,7 \end{cases}$$

Angewandt wird die *DCT* jeweils auf die 8×8 Pixel großen Luminanz- und Chrominanzblöcke. Das Ergebnis ist jeweils wieder ein 8×8 Block, in dem sich die niedrigste Frequenz links oben und die höchste Frequenz rechts unten befindet. Die niedrigste Frequenz, also der $(0,0)$-Koeffizient, wird auch als *DC-Koeffizient* bezeichnet, während die restlichen Werte als *AC-Koeffizienten* bezeichnet werden.

Da das menschliche Auge hohe Frequenzen schlechter wahrnehmen kann als niedrige, werden die Frequenzwerte mit einer Quantisierungsmatrix gewichtet, die für niedrige Frequenzen kleine und für hohe Frequenzen große Werte enthält. Jeder *DCT*-Wert wird also durch einen bestimmten Quantisierungswert dividiert,

um den Wertebereich der *DCT* einzuschränken. Die dadurch entstehenden Werte müssen anschließend auf ganzzahlige Werte gerundet werden, wodurch der Verlust bei den betrachteten Kompressionsverfahren entsteht. Da die Quantisierung also für die eigentliche Kompression zuständig ist, besteht die Möglichkeit, die Quantisierungsmatrix mit einem Quantisierungsparameter zu skalieren. Dadurch kann der Kompressionsgrad während der Codierung dynamisch angepasst werden, so dass die codierte Videosequenz eine konstante Bitrate besitzt.

Durch die Quantisierung enthält der entstandene Block nur sehr wenige von Null verschiedene Werte und kann daher effizient codiert werden. Da bei der Quantisierung Kenntnisse über die Wahrnehmung des menschlichen Auges berücksichtigt werden, befinden sich die von Null verschiedenen Werte in der Nähe des *DC-Koeffizienten*. Aus diesem Grund wird der Block zunächst durch einen Zick-Zack-Scan in einen Bitstrom umgewandelt, der mit Hilfe der *Value-Length-Codierung (VLC)* verkürzt wird.

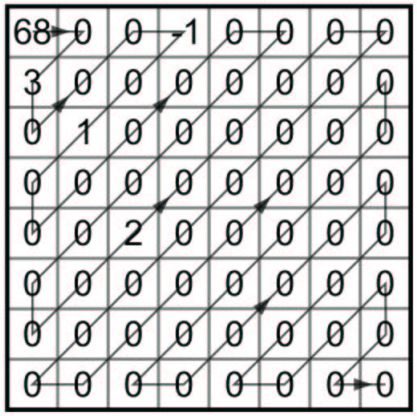


Abbildung 2.2: Zick-Zack-Scan zur Umwandlung eines Blockes in eine Zahlenfolge

Bei der *VLC* wird eine Folge von gleichen Werten als Paar der Länge und des Wertes gespeichert. Da in diesem Fall die meisten Werte Null sind, gibt der erste Wert eines Paares die Länge einer Nullfolge an. Tritt vor einem Wert keine Nullfolge auf, so wird lediglich der Wert selbst codiert. Auf den letzten Wert der nicht Null ist folgt das Paar $(0, 0)$, das das Ende der Zahlenfolge markiert.

Die Integerfolge

$$68\ 0\ 3\ 0\ 0\ 0\ {-1}\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 2\ 0\ \ldots\ 0$$

aus der Abb. 2.2 wird z. B. als

$$(68)\ (1,3)\ (3,-1)\ (1,1)\ (14,2)\ (0,0)$$

codiert. Diese Paare können dann mit Hilfe einer *Entropiecodierung* (z. B. *Huffmancodierung*) optimal codiert werden. Dieses Beispiel ist jedoch nur für den

ersten Block eines Bildes korrekt, denn der Wert der niedrigsten Frequenz, also der *DC-Koeffizient*, wird – anders als die restlichen Werte – als Differenz zum *DC-Koeffizienten* des vorherigen Blockes codiert. Diese Art der Codierung (*Differential Pulse Code Modulation – DPCM*) wird benutzt, da aufeinander folgende *DC-Koeffizienten* häufig einen ähnlichen Wert haben und somit die benötigte Bitzahl verringert werden kann.

2.1.2.2 Inter-Frame-Codierung

In einer Videosequenz sind sich aufeinander folgende Bilder häufig sehr ähnlich. Diese Ähnlichkeiten werden bei der Inter-Frame-Codierung ausgenutzt, um eine weitere Kompression zu erhalten. Hierbei wird für einen Makroblock des zu codierenden Bildes in einem vorherigen Bild (in gewissen Grenzen) nach einer möglichst großen Ähnlichkeit gesucht. Wird solch eine Übereinstimmung gefunden, so wird nicht der Makroblock selbst, sondern, wie in Abb. 2.3 dargestellt, ein Bewegungsvektor, der die Verschiebung bzgl. des gefundenen Blocks des vorherigen Bildes angibt sowie der dadurch entstehende Fehler codiert. Dieser Fehler wird wie bei der Intra-Frame-Codierung mit Hilfe der *DCT* und anschließender Quantisierung, *VLC* sowie *Entropiecodierung* codiert. Bei der Quantisierung wird jedoch eine andere Quantisierungsmatrix eingesetzt als bei der Intra-Frame-Codierung, bei der mehr Werte zu Null quantisiert werden. Wird keine Übereinstimmung gefunden, so wird der Makroblock mit Hilfe der Intra-Frame-Codierung codiert.

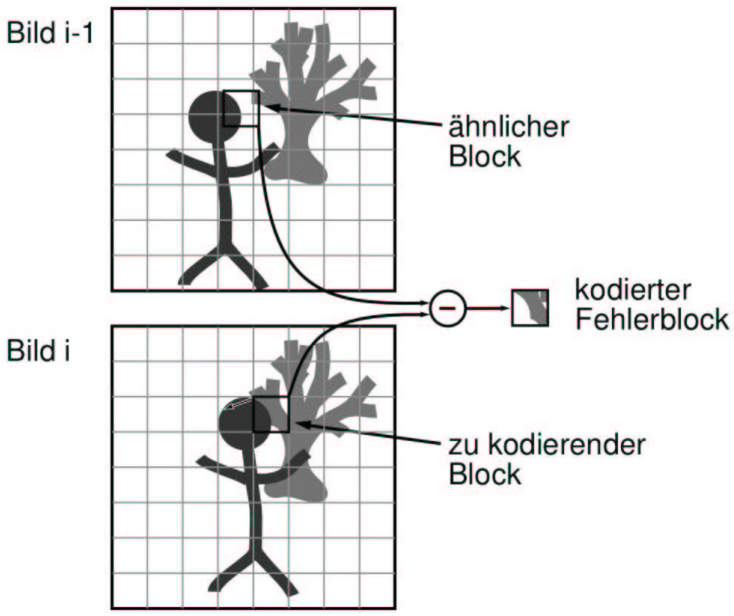


Abbildung 2.3: Ausnutzung von Bildähnlichkeiten bei der Inter-Frame-Codierung

2.1.2.3 Bildtypen

Innerhalb einer codierten Videosequenz existieren bis zu drei verschiedene Bildtypen, die sich in der Art ihrer Codierung unterscheiden. I-Frames enthalten keine Bewegungskompression und wurden ausschließlich mit Hilfe der Intra-Frame-Codierung erzeugt. P- und B-Frames wurden mittels Inter-Frame-Codierung erstellt, wobei P-Frames nur Bewegungsvektoren bzgl. früherer Frames enthalten und B-Frames auch Vektoren enthalten, die auf zukünftige Bilder verweisen. Die Bewegungsvektoren von P- und B-Frames dürfen sich jedoch nicht auf beliebige vorherige oder zukünftige Frames beziehen, sondern immer nur auf den unmittelbar vorherigen bzw. folgenden I- oder P-Frame. B-Frames sind bei den hier betrachteten Formaten nur in H.263, MPEG-1 und MPEG-2 zu finden. Bei H.261 kommt lediglich die unidirektionale Bewegungskompression der P-Frames zum Einsatz.

2.1.2.4 Bildreihenfolge

Der erste Frame einer Sequenz ist immer ein I-Frame, da zu diesem Zeitpunkt noch keine weiteren Informationen verfügbar sind. Anschließend können P- oder B-Frames folgen. Die genaue Anordnung der einzelnen Frametypen ist jedoch nicht in den einzelnen Standards festgehalten, sondern kann vom Programmierer eines Encoders festgelegt werden. Eine sehr häufige Anordnung bei MPEG-Video ist in Abb. 2.4 dargestellt.

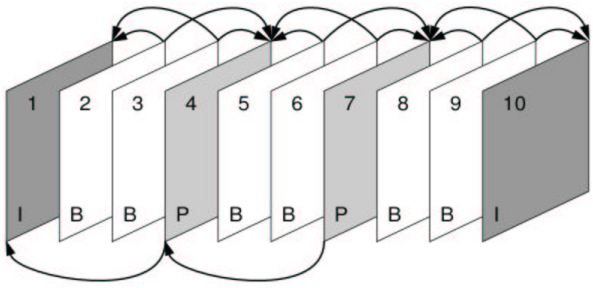


Abbildung 2.4: Bildreihenfolge und Abhängigkeiten innerhalb einer Videosequenz

In der Literatur ist für die Angabe einer konkreten Anordnung der Bilder häufig eine Notation zu finden, die aus der Länge N einer *Group of Pictures (GOP)* und der Länge M einer Untergruppe besteht. Als *GOP* wird hierbei eine Gruppe von Bildern bezeichnet, deren Bewegungskompression in sich abgeschlossen ist, wodurch jede *GOP* implizit mit einem I-Frame beginnen muss. Eine Untergruppe ist eine Gruppe von Bildern, deren bidirektionale Bewegungskompression in sich geschlossen ist. Mit anderen Worten gibt N also den Abstand aufeinander folgender I-Frames und M den Abstand aufeinander folgenden P-Frames an. Für die Länge einer *GOP* ist auch die Angabe $N = \infty$ erlaubt, die angibt, dass nur das

erste Bild einer Videosequenz ein I-Bild ist und alle folgenden P- oder B-Bilder sind. In der Abb. 2.4 bilden z. B. die Frames 1–9 eine *GOP* und die Frames 4–7 eine Untergruppe (also $N = 9$ und $M = 3$).

Aufgrund der bidirektionalen Bewegungskompression bei B-Frames ist es bei der Decodierung von H.263, MPEG-1 und MPEG-2 notwendig, die Reihenfolge der einzelnen Frames zu vertauschen. Es müssen also zunächst alle I- und P-Frames, auf die sich die B-Frames beziehen, decodiert werden. Daher werden die Frames nicht in der Reihenfolge codiert, in der sie abgespielt werden, sondern in der Decodierreihenfolge. Im Beispiel in Abb. 2.4 bedeutet dies, dass die Frames in der Reihenfolge 1, 4, 2, 3, 7, 5, 6, 10, 8, 9 übertragen und auch decodiert werden.

2.1.2.5 Bitstrom

Die Syntax des Bitstroms gehört zu den spezifischen Formatmerkmalen, die in den einzelnen Standards genau festgelegt wurden. Bei H.261 und H.263 besteht der Strom aus vier verschiedenen Ebenen. Jedes Bild des Videostroms ist in mehrere *Group of Blocks (GOB)* unterteilt, die jeweils eine bestimmte Anzahl Makroblöcke enthalten. Jeder Makroblock wiederum enthält sechs[4] einzelne Blöcke, die jeweils aus den entropiecodierten Werten der *VLC* und einem End-Of-Block-Code (EOB) bestehen.

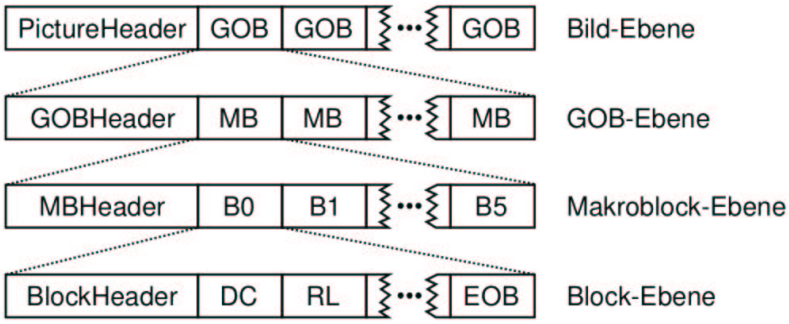


Abbildung 2.5: H.261 und H.263 Bitstrom

Bei MPEG-1/2 wird zwischen sechs verschiedenen Ebenen unterschieden. Der Videostrom besteht aus einzelnen Sequenzen, die jeweils eine bestimmte Anzahl *Group of Pictures* enthalten. Die einzelnen Bilder sind in Gruppen von Makroblöcken unterteilt, die als Slices bezeichnet werden. Die ersten drei Ebenen wurden zur Unterstützung von wahlfreiem Zugriff innerhalb des Videostroms eingeführt. Die zusätzliche Unterteilung eines Bildes in Slices dient zur einfachen Neusynchronisation, falls bei der Übertragung die Synchronisation verloren gehen sollte.

[4] vier Luminaz- und zwei Chrominanzblöcke

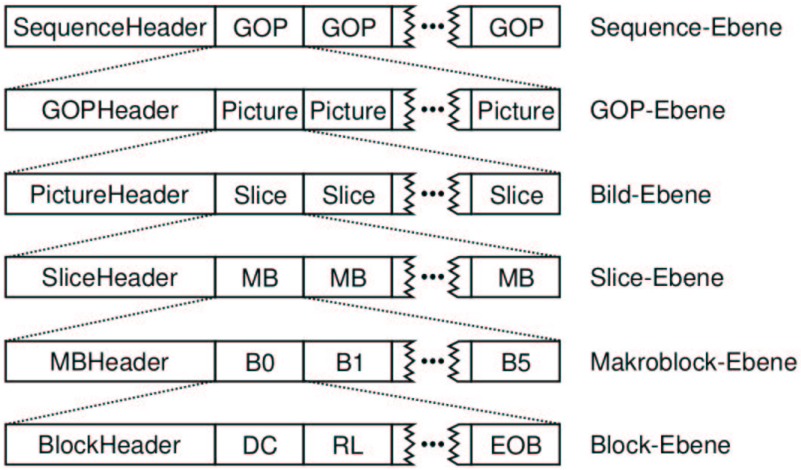


Abbildung 2.6: MPEG-1/2 Bitstrom

2.1.3 Die Videoformate H.261, H.263, MPEG-1 und MPEG-2

An dieser Stelle werden die vier Formate kurz vorgestellt und einige Feinheiten der jeweiligen Codierung genannt.

2.1.3.1 H.261

H.261 ist ein Codierungsstandard, der 1990 von der *ITU* für Videoübertragungen mit geringen Bandbreiten entwickelt wurde. Die betrachtete Bandbreite dieses Formats ist ein Vielfaches von 64 kBit, was darauf zurückzuführen ist, dass es ursprünglich für Videotelefonie in *ISDN*-Netzen entwickelt wurde. Insgesamt unterstützt dieser Standard zwei verschiedene Bildformate:

- *Common Interchange Format (CIF)* : 352 x 288 Pixel
- *Quater Common Interchange Format (QCIF)* : 176 x 144 Pixel

Anders als in den anderen drei Standards ist in H.261 nur die unidirektionale Bewegungskompression vorgesehen, bei der sich die Bewegungsvektoren ausschließlich auf zeitlich frühere Bilder beziehen.

2.1.3.2 H.263

Bei H.263 handelt es sich um eine Weiterentwicklung von H.261, die 1995 von der *ITU* veröffentlicht wurde. Das Codierverfahren ist dem von H.261 sehr ähnlich, unterscheidet sich allerdings in den folgenden Punkten:

- doppelte Genauigkeit der Bewegungsvektoren
- einige Angaben im Datenstrom sind optional
- durch vier optionale Parameter kann die Performance beeinflusst werden
- bidirektionale Bewegungskompression
- drei zusätzliche Auflösungen (SQCIF: 128x96, 4CIF: 704x576 und 16CIF: 1408x1152)

H.263 wurde wie H.261 zur Videoübertragung über *ISDN* entwickelt und ist daher für geringe Datenraten konzipiert. Die Einführung der bidirektionalen Bewegungskompression bietet die Möglichkeit, höhere Kompressionsraten als H.261 zu erreichen. Anders als bei MPEG-1 und MPEG-2 wird diese Art der Bewegungskompression jedoch nur sehr wenig eingesetzt. Stattdessen wird überwiegend unidirektionale Bewegungskompression genutzt und auf den Einsatz intraframe-codierter Bilder verzichtet, um eine möglichst geringe Datenrate zu erreichen.

2.1.3.3 MPEG-1

Der Standard MPEG-1 wurde 1991 von der *Moving Pictures Expert Group* veröffentlicht und ist für eine Auflösung von 352 x 240 Pixel bei 30 Bildern/s bzw. 352 x 288 Pixel bei 25 Bildern/s vorgesehen. Diese Auflösungen werden als *Source Input Format (SIF)* bezeichnet, wobei der Standard Auflösungen bis zu 4096 x 4096 Pixel bei 60 Bildern/s unterstützt. MPEG-1 wurde für Datenraten um 1,5 MBit/s optimiert.

2.1.3.4 MPEG-2

MPEG-2 wurde seit 1990 für den Bereich des digitalen Fernsehens entwickelt und 1994 fertig gestellt. Die Datenrate wurde gegenüber MPEG-1 auf einen Wert zwischen 4 und 9 MBit/s angehoben und die unterstützten Bildformate in Level und Profile aufgeteilt. Die vier existierenden Level bestimmen die Auflösung sowie die maximale Datenrate, während die sieben Profile verschiedene Funktionalitäten wie Skalierbarkeit und Farbgenauigkeit festlegen. Die häufigste Kombination ist das Main-Profil und der Main-Level (MP@ML), was eine Auflösung von 720 x 480 Pixel bei 30 Bildern/s und einer Datenrate bis zu 15 MBit/s definiert. Anders als die anderen Videoformate unterstützt MPEG-2 interlaced Videosignale. Bei interlaced Videosignalen besteht eine Sequenz nicht aus einer Folge von einzelnen Bildern, sondern aus Halbbildern, sog. Feldern. Jeweils ein Feld enthält alle geraden und das andere alle ungeraden Zeilen, die zeitlich verschoben

sind[5]. Aufgrund der zeitlichen Verschiebung können die Halbbilder nicht einfach zusammengesetzt und anschließend in einem der vorherigen Formate codiert werden. Daher enthält der MPEG-2 Standard neben der vorgestellten framebasierten auch feldbasierte *DCT* und Bewegungskompression. Bei der feldbasierten Variante enthält ein Block immer nur Zeilen eines Feldes. Ein Makroblock besteht dann aus zwei Blöcken mit ungeraden und zwei Blöcken mit geraden Zeilen. Bei der feldbasierte Bewegungskompression werden die Bewegungsvektoren pro Feld bestimmt, was bis zu vier Vektoren pro Makroblock[6] liefert.

2.1.3.5 Weitere Formate

Neben diesen vorgestellten vier Formaten gibt es eine Vielzahl anderer Formate, auf die an dieser Stelle hingewiesen wird. Eine genauere Betrachtung findet jedoch nicht statt, da dies den Rahmen der Arbeit sprengen würde.

Der neueste Video-Standard der *Moving Pictures Expert Group* ist MPEG-4. Die Arbeit zu diesem Standard begann bereits 1994 und Version 1 wurde 1999 veröffentlicht. Ziel dieser Entwicklung war ein Standard zur Videocodierung für geringe Datenraten, ähnlich wie bei H.263. Viele Konzepte und Mechanismen wurden von MPEG-2 übernommen, eine Neuerung ist jedoch der Ansatz der objektorientierten Videocodierung. Hierbei wird versucht, innerhalb eines Videos Objekte zu finden und diese sowie deren Bewegung zu codieren. Da dadurch die Bewegungskompression nicht, wie bei den obigen vier Formaten, auf einen kleinen Block beschränkt ist, kann eine höhere Kompression erreicht werden. Einige der in dieser Arbeit vorgestellten Techniken können auch auf MPEG-4 angewandt werden, müssen dazu jedoch an die objektorientierte Bewegungskompression angepasst werden.

Zusätzlich zu den bereits beschriebenen Formaten findet man im Internet häufig Formate, die von Softwarefirmen entwickelt wurden. Hierzu zählen z. B. RealMedia, Apple Quicktime und AVI. Diese Formate stellen keine eigenen Videoformate zur Verfügung, sondern dienen vielmehr als Container für Video und Audiodaten. Die Dateien enthalten also neben den Audio- und Videodaten in einem der bereits vorgestellten Formate zusätzliche Informationen zu den Daten, die von spezieller Software interpretiert werden kann. Leider stellen die Firmen, die diese proprietären Formate entwickelt haben, nur sehr spärliche Informationen über die Formate selbst zur Verfügung, so dass eine genauere Betrachtung entfallen muss.

[5] Der Ursprung dieser Videosignale liegt in der frühen Entstehungsphase des Fernsehens, in der TV-Geräte noch nicht in der Lage waren, 50 (bzw. 60 in den USA) Bilder pro Sekunde anzuzeigen. Die Hälfte war jedoch durchaus realisierbar. Es konnten also 50 bzw. 60 Halbbilder pro Sekunde angezeigt werden. Aufgrund der Wahrnehmung des menschlichen Auges wurden die Videosignale in zwei Felder geteilt, die nacheinander auf dem Bildschirm angezeigt wurden. Dies führte zusätzlich zu 'weicheren' Bewegungen und geringerem Flimmern.

[6] zwei pro Feld, jeweils vor- und rückwärts

2.2 Audioformate

Zur Digitalisierung eines analogen Audiosignals wird dieses mit einer bestimmten Samplingrate abgetastet und die so erhaltenen Werte abgespeichert. Bei der *Pulse Code Modulation (PCM)* wird jeder Abtastwert quantisiert und als Integer abgespeichert, wobei eine sehr große Anzahl Bits entsteht. Um den benötigten Speicherplatz zu verringern, werden bei der *Differential Pulse Code Modulation (DPCM)* nur die Differenzen aufeinander folgender Abtastwerte abgespeichert, was zu einer deutlichen Reduktion der Bitanzahl bei gleicher Qualität gegenüber der *PCM* führt. Bei der *Adaptive Differential Pulse Code Modulation (ADPCM)* wird zusätzlich die Signalcharakteristik in den Abtastvorgang mit einbezogen, wodurch eine höhere Qualität bei gleicher Speicherplatzanforderung gegenüber der *DPCM* erreicht wird. Wie bereits in der Einleitung dieses Kapitels erwähnt, ist diese Art der Speicherung jedoch zu ineffektiv, und das Audiosignal sollte komprimiert werden. Hierzu gibt es verschiedene Ansätze, von denen an dieser Stelle die Standards der *Moving Pictures Expert Group* zur Audiokompression kurz vorgestellt werden.

2.2.1 MPEG-Audio

Neben Standards zur Videokompression hat die *Moving Pictures Expert Group* auch Standards zur Audiokompression entwickelt. MPEG-1-Audio [23] spezifiziert drei verschiedene Layer der Audiocodierung, die sich in ihrer Komplexität und erreichbaren Kompression voneinander unterscheiden. Layer I ist die einfachste Kompression und das als *MP3* bekannte Layer III die komplizierteste. Die einzelnen Layer sind abwärtskompatibel, so dass ein Layer-III-Decoder auch Layer I und II Audiosignale decodieren kann. Der Standard unterstützt die Codierung von Mono- und Stereosignalen sowie von zwei unabhängigen Kanälen, die z. B. unterschiedliche Sprachen enthalten können.

Zur Kompression der Audiosignale wird versucht, Teile des Audiosignals, die für das menschliche Gehör nicht hörbar sind, zu entfernen. Wenn es z. B. in einem Audiosignal eine dominierende Frequenz gibt, so können andere, leisere Frequenzen vom menschlichen Gehör nicht wahrgenommen werden und sind somit redundant. Um diese redundanten Frequenzen herauszufiltern, wird das Signal zunächst in 32 verschiedene Frequenzbänder zerlegt, die anhand eines sog. psycho-akustischen Modells bewertet werden. Für jedes Frequenzband wird anhand eines Schwellwertes entschieden, ob dieser Teil des Signals für den Menschen hörbar ist. Liegt die Lautstärke eines Frequenzbands unterhalb des Schwellwertes, so wird dieses nicht codiert. Andernfalls wird das zugehörige Signal quantisiert, so dass die gewünschte Bitrate erreicht wird und anschließend in den endgültigen Bitstrom codiert.

Der MPEG-2-Standard [24] enthält einige Verbesserungen gegenüber MPEG-1-Audio, wobei sehr auf die Kompatibilität der Formate geachtet wurde. MPEG-2-Audio ist auf- und abwärtskompatibel zu MPEG-1-Audio, da die Struktur des Bitstroms übernommen wurde. Die Verbesserungen umfassen die Unterstützung geringerer Samplingraten (neben 48, 44,1 und 32 kHz auch 16, 22,05 und 24 kHz) für eine bessere Qualität bei geringen Datenraten sowie die Unterstützung von bis zu fünf Kanälen. Zusätzlich hat die *Moving Pictures Expert Group* einen Standard für *Advanced Audio Coding (AAC)* [25] entwickelt, der eine bessere Unterstützung für fünf Kanäle bietet, wie sie z. B. in heutigen Kinofilmen genutzt werden. Auch die Qualität des codierten Audiosignals wurde gegenüber dem vorherigen Standard verbessert, indem Verbesserungen am Frequenzband-Filter, der Quantisierung sowie der Bitstromcodierung vorgenommen wurden. Beim Zerlegen des Signals in verschiedene Frequenzbänder wurden bekannte Schwächen von MP3 entfernt, die Granularität der Quantisierung wurde verfeinert und durch statistische Vorausbestimmung verbessert. Der Bitstrom wird bei *AAC* mittels *Huffmancodierung* codiert, was zu einer effizienteren Nutzung der verfügbaren Datenrate führt.

2.3 Audio-Video-Ströme

Eine Trennung von Audio- und Videodaten ist aufgrund der verschiedenen Datentypen bei der Datenkompression unerlässlich. Bei der Wiedergabe ist es aber häufig notwendig, beide Datentypen gleichzeitig und synchron darzustellen. Damit hierzu nicht mehrere Datenströme an den Client übertragen werden müssen, wurden Formate entwickelt, in denen beide Daten enthalten sind. An dieser Stelle soll nun der *MPEG*-Systemstrom vorgestellt werden, der jeweils für MPEG-1 und MPEG-2 standardisiert wurde [26, 27]. Für H.261 und H.263 existiert kein Standard, der beide Datentypen in einem Strom vereint.

2.3.1 MPEG-Systems

Der MPEG-Systemstrom vereint sowohl Audio- und Videodaten als auch zugehörige Zusatzinformationen in einem Datenstrom. Hierzu werden die unterschiedlichen Daten in Pakete aufgeteilt, die zu sog. Packs zusammengefasst werden und zusammen den Datenstrom ergeben. In Abb. 2.7 ist der hierarchische Aufbau des Datenstroms für MPEG-1 schematisch dargestellt. Der Systemheader ist nur beim ersten Pack notwendig, kann aber in folgenden Packs wiederholt werden. Die Anzahl der Pakete innerhalb der Packs ist nicht vorgegeben, sondern kann vom Encoder frei gewählt werden. Die einzelnen Bestandteile des Stroms werden durch 32-Bit-Werte voneinander getrennt, so beginnt z. B. der Packheader mit

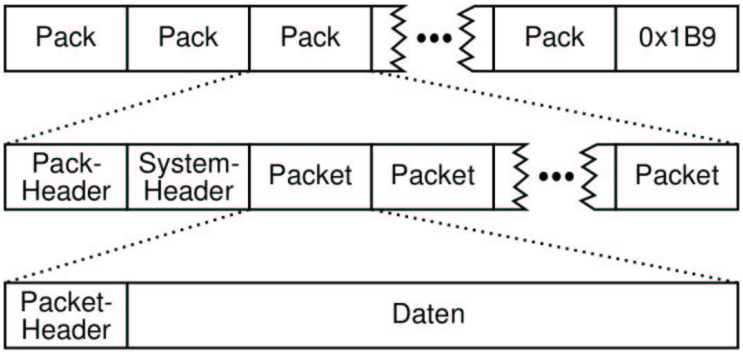


Abbildung 2.7: MPEG-1/2-Systemstrom

dem hexadezimalen Wert 0x1BA und der Systemheader mit 0x1BB. Um welche Daten es sich innerhalb eines Pakets handelt, ist ebenfalls an den ersten 32 Bits des Headers zu erkennen. Videodatenpakete beginnen z. B. mit dem Wert 0x1E0. Die Daten, die zur Synchronisation von Audio und Video notwendig sind, sind in den einzelnen Paketheadern enthalten. Bei MPEG-2 wurde der Aufbau des Datenstromes im Wesentlichen beibehalten, wobei die Header um weitere Informationen erweitert wurden.

3 Medienübertragung

Die wohl bekanntesten und am weitesten verbreiteten Protokolle zur Datenübertragung im Internet sind das *Hyper Text Transfer Protocol (HTTP)* und das *File Transfer Protocol (FTP)*. Beide Protokolle dienen dazu, Daten in Form von Dateien zu übertragen. Bei Multimediadaten handelt es sich jedoch um sehr große Datenmengen, bei denen eine Übertragung in Form einer Datei nicht praktikabel wäre, da zwischen Anforderung und Wiedergabe der Daten inakzeptabel lange Wartezeiten entstünden. Stattdessen sollten die Daten in einer Form übertragen werden, die es dem Client erlaubt, die Wiedergabe bereits während der Übertragung zu starten. Hierzu existiert eine Gruppe von Protokollen, die die zeitkritischen Anforderungen von Audio- und Videodaten unterstützt und diese als kontinuierliche Datenströme überträgt. Zur Signalisierung einer Multimediaübertragung wird das *Real Time Streaming Protocol (RTSP)* eingesetzt. Die Daten selbst werden mit Hilfe des *Real-Time Transport Protocol (RTP)* übertragen und für die Kontrolle dieser Datenübertragung kommt das *Real Time Control Protocol (RTCP)* zum Einsatz. Auf der Transportschicht wird bei *RTP* und *RTCP* *UDP* und bei *RTSP* wahlweise *TCP* oder *UDP* eingesetzt.

3.1 Real Time Streaming Protocol

Bei *RTSP* [28] handelt es sich um ein Klartextprotokoll der Anwendungsschicht, das eine Kontrolle von zeitsynchronisierten Datenströmen kontinuierlicher Daten wie Audio und Video bietet. Die Daten selbst werden nicht mit *RTSP*, sondern typischerweise mit *RTP* oder einem beliebigen anderen Transportmechanismus übertragen. *RTSP* bietet aber Möglichkeiten, diese Übertragung zu unterbrechen oder zu stoppen. Das Protokoll dient also als eine Art Fernbedienung für Medienserver. Bei der Definition von *RTSP* wurden viele Konzepte und Formate von *HTTP* übernommen, weshalb es häufig als "HTTP-friendly" bezeichnet wird. Trotzdem bestehen wesentliche Unterschiede zwischen diesen beiden Protokollen:

- RTSP-Server sind im Gegensatz zu HTTP-Servern zustandsbehaftet.
- Sowohl Server als auch Client können Kommandos senden.
- Daten werden mit einem separaten Protokoll übertragen.

Zwischen Client und Server wird eine sog. Session aufgebaut, die die Medienübertragung begleitet und steuert. Eine Session ist der gesamte Prozess der Übertragung eines oder mehrerer Medienströme vom Server zum Client, beginnend mit dem Aufbau der Verbindung, Initialisierung eines Transportmechanismus, Übertragung der Daten und anschließender Trennung der Verbindung. Während der Sitzung ist es nicht notwendig, auch eine Verbindung auf der Transportebene zum Server aufrecht zu erhalten. Stattdessen kann z.B. für jede Nachricht eine eigene TCP-Verbindung erzeugt werden. Daher wird jede Session mit einer eindeutigen ID gekennzeichnet, die es dem Server erlaubt, verschiedene Sessions zu verwalten. Da sich eine Session über den gesamten Prozess der Medienübertragung erstreckt, müssen sowohl Client als auch Server zustandsbehaftet sein, denn nicht zu jedem Zeitpunkt ist jede *RTSP*-Nachricht erlaubt (es macht z.B. wenig Sinn, nach dem Aufbau einer Sitzung diese noch einmal zu initiieren). Die Beschreibung einer Session wird im *RTSP* nicht genauer definiert, typischerweise kommt jedoch das *Session Description Protocol (SDP)* zum Einsatz, das in [29] definiert wurde. Beim *SDP* handelt es sich ebenfalls um ein Klartextprotokoll, das Definitionen enthält, wie bestimmte Eigenschaften und Parameter einer Session dargestellt werden. *RTSP* definiert insgesamt elf verschiedene Nachrichten, die hier jedoch nicht näher erläutert werden. Stattdessen wird der Leser auf das *RFC* [29] verwiesen. Jede versendete Anforderungsnachricht wird durch eine Bestätigungsnachricht quittiert, die ggf. die angeforderten Informationen enthält. Zur korrekten Zuordnung von Bestätigungen zu vorherigen Anforderungen trägt jede Nachricht eine eindeutige Sequenznummer, die während der gesamten Session inkrementiert wird.

3.2 Real-Time Transport Protocol

RTP ist ein Internet-Standard Protokoll [30] zum Transport von Echtzeitdaten, wie z.B. Video und Audio. Der Standard umfasst neben dem Transportprotokoll zusätzlich die Spezifikation eines Kontrollprotokolls, dem *Real Time Control Protocol*.

RTP bietet der Anwendungsschicht Möglichkeiten zur Übertragung von kontinuierlichen Daten unter Echtzeitbedingungen, wobei *RTP* selbst keine Mechanismen für eine zuverlässige Übertragung zur Verfügung stellt. Somit wird durch *RTP* kein voll funktionsfähiges Transportprotokoll implementiert, sondern es muss immer zusammen mit einem Protokoll der Transportschicht betrieben werden. Typischerweise wird hierzu das *User Datagram Protocol (UDP)* genutzt, was dazu führt, dass die Anwendung neben der Einhaltung der Echtzeitbedingungen auch die zuverlässige Übertragung und die richtige Reihenfolge der Pakete gewährleisten muss.

Ein Medienstrom, der mittels *RTP* übertragen werden soll, wird in Fragmente zerlegt, die jeweils mit einem *RTP*-Header versehen und als Paket verschickt werden. Der Header umfasst u.a. einen Zeitstempel des Sendezeitpunktes, eine Payload-ID, die den Typ der Daten angibt, eine Sequenznummer, anhand derer der Empfänger Paketverluste erkennen kann und eine eindeutige Kennzeichnung des Medienstroms.

Das *Real Time Control Protocol* dient für Rückmeldungen an den Sender eines Datenstroms, der diese Informationen dazu nutzen kann, den Sendevorgang anzupassen. Im Gegensatz zur *RTP*-Kommunikation handelt es sich bei *RTCP* allerdings um eine bidirektionale Kommunikation. Innerhalb einer *RTP*-Übertragung übernimmt *RTCP* im Wesentlichen vier Funktionen:

- Quality of Service und Staukontrolle
- eindeutige Identifikation eines Medienstroms
- Steuerung der Anzahl von RTCP-Paketen
- einfache Sitzungskontrolle

Im Vergleich zu *RTSP*, das als Kontrollprotokoll auf der Anwendungsebene dient, kann *RTCP* als dessen Äquivalent auf der Transportebene betrachtet werden.

4 Transcoding

Ziel dieser Arbeit ist das Design eines Gateways, das in der Lage ist, Medienströme für mobile Geräte umzuwandeln. Diese Umwandlung von Multimediadaten wird auch als Transcoding bezeichnet, welches in diesem Kapitel näher erläutert werden soll. Insbesondere werden Techniken vorgestellt, die zur Umwandlung von Videodaten eingesetzt werden können. Zunächst werden kurz die Ziele des Transcodings erläutert. Daran anschließend folgt ein Abschnitt, der sich mit effizienten Algorithmen zur Umwandlung von Bewegungsvektoren und den Bilddaten beschäftigt, wie sie beim Transcoding eingesetzt werden können. Die letzten beiden Teile dieses Kapitels befassen sich mit konkreten Transcoderarchitekturen zur Datenreduktion und zur Formatumwandlung.

4.1 Problemstellung

Ziel des Transcodings ist einerseits die Reduktion der Datenrate und andererseits die Umwandlung des verwendeten Datenformates. Eine Reduktion der Datenrate kann durch die Erhöhung des Kompressionsgrades bzw. des Quantisierungsfaktors (Requantisierung), die Verringerung der Bildrate (Frameskipping), die Reduzierung der Auflösung der einzelnen Bilder (Skalierung) oder eine Kombination dieser drei Möglichkeiten erreicht werden. Eine Umwandlung des Datenformates ist dann erforderlich, wenn der Client nicht in der Lage ist, ein bestimmtes Format darzustellen. Meistens wird die Umwandlung jedoch im Zusammenhang mit der Datenreduktion eingesetzt, da bei der Formatumwandlung durchaus größere Datenmengen entstehen können. Denn bei der Umwandlung ist es zum Teil notwendig, die Bewegungskompression zu entfernen.

Eine direkte Umwandlung der einzelnen Bilder sowie deren Abhängigkeiten innerhalb des Datenstromes ist jedoch nicht möglich, da die Videodaten in codierter Form vorliegen. Der einfachste Ansatz für das Transcoding von Videodaten ist daher die Decodierung mit anschließender Bearbeitung und erneuter Codierung der einzelnen Bilder. Dieser Ansatz ist schematisch in Abb. 4.1 dargestellt. Sowohl Decodierung als auch Codierung sind sehr rechen- und zeitintensiv und können hintereinander nicht in Echtzeit durchgeführt werden. Aus diesem Grund ist die-

Abbildung 4.1: Erster Ansatz für einen Transcoder

ser erste Ansatz nicht für eine Realzeitbearbeitung von Multimediaströmen geeignet. Es bestehen jedoch Möglichkeiten, diese Architektur so zu vereinfachen, dass die Realzeitbedingungen des Transcodings eingehalten werden können. Einerseits ist es möglich, die Umwandlungen der Videodaten teilweise im Frequenzbereich durchzuführen, wodurch also die Anwendung der inversen *DCT* zur Erzeugung der Pixeldaten entfallen würde. Zusätzlich können die im Datenstrom enthaltenen Bewegungsinformationen bei der Erzeugung der umgewandelten Daten wiederverwendet werden, so dass die Suche nach Bewegungsvektoren entfallen würde.

4.2 Wiederverwendung von Bewegungsvektoren

Bei der Intra-Frame-Codierung werden Ähnlichkeiten zwischen zeitlich aufeinander folgenden Bildern ausgenutzt, um die Datenmenge der codierten Videosequenz zu reduzieren. Hierzu werden in früheren und ggf. auch in späteren Bildern nach Ähnlichkeiten gesucht und diese in Form von Bewegungsvektoren gespeichert. Bei der Transcodierung werden zwar die einzelnen Bilder verändert, aber die Ähnlichkeiten zwischen den Bildern bleiben weitestgehend erhalten. Eine erneute Suche nach Bewegungsvektoren ist also überflüssig, wenn die im Datenstrom enthaltenen Bewegungsinformationen bei der Transcodierung wiederverwendet werden. Eine Anpassung der Informationen ist aber dennoch notwendig, damit der Transcodierungsfehler möglichst gering gehalten wird.

4.2.1 Interpolation von Bewegungsvektoren

Durch den Transcodierungsvorgang können die vorhandenen Bewegungsvektoren ungültig werden, wenn sie sich z. B. auf verworfene oder skalierte Frames beziehen. In diesen Fällen ist es notwendig, einen neuen Vektor in geeigneter Weise aus mehreren vorhandenen zu interpolieren.

4.2.1.1 Verringerung der Auflösung

Wird die Auflösung eines Videostroms verringert, müssen die Bewegungsvektoren skaliert und zusammengefasst werde. Angenommen die Auflösung der Bilder wird auf die Hälfte reduziert, dann müssen jeweils vier Vektoren von benachbarten Makroblöcken zusammengefasst und skaliert werden. In [18] wurden drei Methoden zur Erzeugung der neuen Bewegungsvektoren miteinander verglichen: Auswahl eines mittleren Vektors (median filtering), Berechnung eines Durchschnittsvektors (averaging) und Berechnung eines Durchschnittsvektors aus der Mehrheit gleichgerichteter Vektoren (moving with majority). Es wurde gezeigt, dass die Auswahl eines mittleren Vektors die besten Ergebnisse liefert. Hierbei wird zunächst für jeden Vektor v_i die Summe d_i der euklidischen Abstände zu den anderen drei Vektoren berechnet:

$$d_i = \sum_{\substack{j=1 \\ j \neq i}}^{4} \|v_i - v_j\|$$

Der mittlere Vektor ist definiert als derjenige, der den geringsten Abstand zu den anderen besitzt. Dieser muss anschließend noch in seiner Länge an die Skalierung der Bilder angepasst werden. In [31] wurde diese Methode weiter verbessert, indem der Abstand durch die Aktivität des betrachteten Makroblocks gewichtet wurde.

$$d_i = \frac{1}{ACT_i} \sum_{\substack{j=1 \\ j \neq i}}^{4} \|v_i - v_j\|$$

Die Aktivität ACT_i eines Makroblocks kann hierbei als die quadrierte oder absolute Summe der DCT-Werte, die Anzahl der von Null verschiedenen DCT-Werte oder einfach als Wert des *DC-Koeffizienten* berechnet werden. Der gesuchte Vektor v ist dann, bei einer Skalierung um den Faktor 2, gegeben durch:

$$v = \frac{1}{2} \arg \min_{v_i \in \{v_1, v_2, v_3, v_4\}} d_i$$

Bei der Betrachtung von nur vier Bewegungsvektoren besteht die Gefahr, dass das entstehende Bild eine sichtbare Blockstruktur aufweist. In [32] werden daher zusätzlich, wie in Abb. 4.2 dargestellt, acht weitere Bewegungsvektoren der umliegenden Makroblöcke berücksichtigt. Der neue Vektor v wird hierbei nach der Formel

$$v = \frac{r \sum_{i=1}^{4}(w_i v_i) + (1-r) \sum_{j=1}^{8}(w_{nj} v_{nj})}{2}$$

berechnet, wobei die Gewichtungsfaktoren w_i, w_{nj} und r adaptiv bestimmt werden.

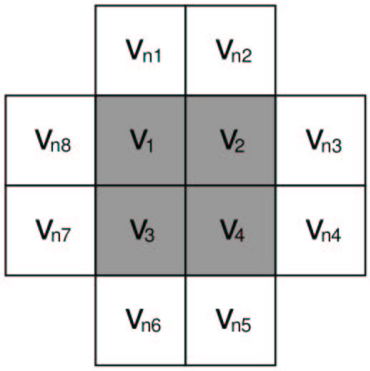


Abbildung 4.2: Skalierung von Bewegungsvektoren

Für die Bestimmung dieser Faktoren werden die folgenden Regeln verwendet:

- Sind die Vektoren v_i alle gleich, so werden $r = 1$ und $w_i = \frac{1}{4}$ gesetzt.
- Sind die Vektoren v_i paarweise verschieden, so werden $r = 1$ und w_i proportional zur Aktivität des jeweiligen Blocks, mit $\sum_{i=1}^{4} w_i = 1$, gesetzt.
- Sonst werden $r = \frac{4}{5}$, $w_i = \frac{1}{4}$ und $w_{nj} = \frac{1}{8}$ gesetzt.

Ein weiterer Ansatz zur Bestimmung der skalierten Bewegungsvektoren wurde in [9] gegeben, bei dem nur die *DC-Koeffizienten* der beteiligten Blöcke betrachtet werden. Bei einer Skalierung um den Faktor 2 bedeutet dies, dass der Vektor v des skalierten Makroblocks wie folgt bestimmt wird:

$$v = \frac{1}{2} \arg \max_{v_i \in v_1, \ldots, v_4} DC(MB_i) \ ,$$

wobei $DC(MB_i)$ die Summer der *DC-Koeffizienten* des Makroblocks MB_i ist. Dieses Verfahren hat den Vorteil, dass es sehr leicht zu berechnen ist und trotzdem visuell bessere Ergebnisse liefert als z. B. die Berechnung eines Durchschnittsvektors.

Handelt es sich bei dem Zielformat um H.263 oder MPEG-4, so ist es möglich, einen Bewegungsvektor für jeden 8×8 Subblock zu codieren [13]. Dies kann dann vorteilhaft sein, wenn die Vektoren der zu skalierenden Makroblöcke sich zu sehr unterscheiden. Eine Interpolation der Bewegungsvektoren ist dann nicht mehr notwendig. Stattdessen können die Vektoren der beteiligten Makroblöcke skaliert und jeweils direkt einem Subblock des skalierten Makroblocks zugeordnet werden. In den meisten Fällen liefert diese Methode bessere Ergebnisse als die Interpolation nur eines Vektors [32]. Da hierbei jedoch vier Vektoren pro Makroblock codiert werden müssen, ist es wenig sinnvoll, diese Strategie grundsätzlich einzusetzen. Stattdessen sollte diese Methode in Abhängigkeit der Varianz der beteiligten Vektoren eingesetzt werden [13].

4.2.1.2 Verwerfen von Frames

Werden einzelne Bilder des Videostroms verworfen, werden diejenigen Bewegungsvektoren, die sich auf verworfene Frames beziehen, ungültig und müssen so angepasst werden, dass sie sich auf den letzten übertragenen Frame beziehen. Es muss also der Pfad der Vektoren vom aktuellen Bild über die verworfenen Bilder bis zum letzten übertragenen Bild verfolgt und zu einem neuen Vektor zusammengefasst werden. In den meisten Fällen verweist ein Bewegungsvektor nicht auf einen Makroblock im vorherigen Bild, sondern auf einen Block, der zwischen vier benachbarten Makroblöcken liegt. Für diesen Block existiert jedoch kein Bewegungsvektor, sondern dieser muss aus den umliegenden Vektoren berechnet werden. Eine Situation, in der ein Frame verworfen wurde ist in Abb. 4.3 dargestellt. In [33] wurde zur Berechnung des fehlenden Vektors W_{t-1} die bilineare Interpolation vorgeschlagen, die folgendermaßen definiert ist:

$$W_{t-1} = (1-\alpha)(1-\beta)V_1^{t-1} + \alpha(1-\beta)V_2^{t-1} + (1-\alpha)\beta V_3^{t-1} + \alpha\beta V_4^{t-1}$$

Wobei V_i^{t-1} die Bewegungsvektoren der umliegenden Makroblöcke zum Zeitpunkt $(t-1)$ sind und α und β der horizontale bzw. vertikale Abstand zum ersten Makroblock ist. Wird dieses Verfahren über alle verworfenen Frames hinweg durchgeführt, erhält man die interpolierten Bewegungsvektoren für den aktuellen Frame. Hier zeigt sich jedoch bereits der Nachteil dieses Verfahrens, denn es müssen alle Vektoren der verworfenen Bilder bis zum nächsten zu übertragenen Bild gespeichert werden. Außerdem kann es vorkommen, dass der durch die vier Makroblöcke abgedeckte Bereich zu divergent ist, um einen guten Vektor für den Block B zu bestimmen. Daher wurde in [8] die *Forward Dominant Vector Selection (FDVS)* vorgestellt, die das Abspeichern verworfener Frames überflüssig macht. Hierbei wird für alle Makroblöcke eines verworfenen Frames jeweils ein dominanter Vektor gespeichert. Ein dominanter Vektor für einen Block ist der Bewegungsvektor desjenigen Blocks, der mit dem Block, auf den der aktuelle Bewegungsvektor verweist, die größte Überdeckung besitzt. In Abb 4.3 wäre dies z. B. für den Block B der Vektor V_1^{t-1} des Makroblocks MB_1^{t-1}.

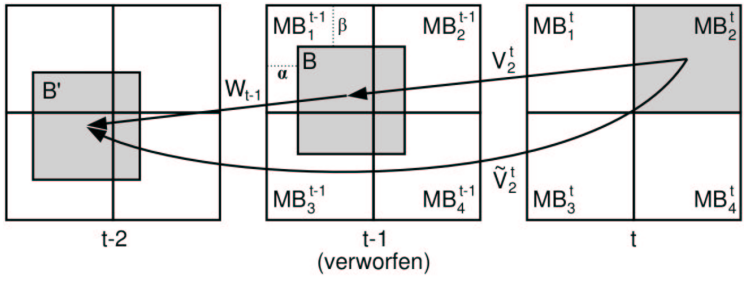


Abbildung 4.3: Bewegungsvektorsuche beim Frameskipping

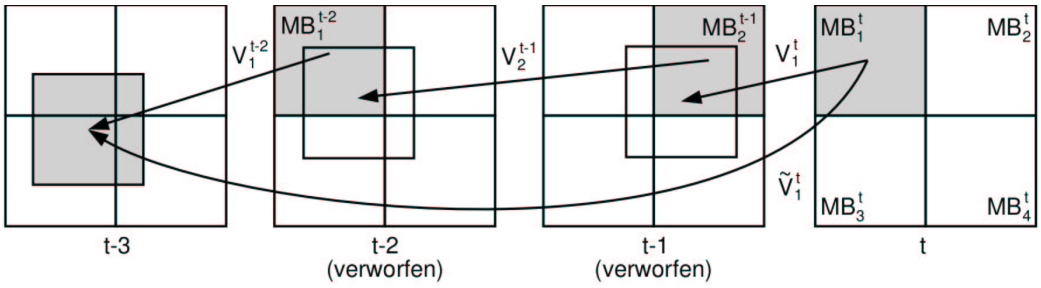


Abbildung 4.4: Forward Dominant Vector Selection (*FDVS*)

In Abb. 4.4 ist eine Situation dargestellt, in der mehr als ein Frame verworfen wurde. Bei den angegebenen Vektoren handelt es sich jeweils um die dominanten Bewegungsvektoren für die Blöcke, auf die jeweils vom folgenden Frame verwiesen wird. Bei der *FDVS* werden die Bewegungsvektoren des ersten verworfenen Frames im Transcoder abgespeichert. Wird ein weiterer Frame verworfen, so werden die gespeicherten Vektoren aktualisiert, indem für jeden Bewegungsvektor des aktuellen Blocks ein dominanter Vektor im vorherigen Bild ermittelt wird. In der Abb. 4.4 ist das z. B. notwendig, wenn der Frame zum Zeitpunkt $(t-1)$ verworfen werden soll. Für den Makroblock MB_2^{t-1} wird der Vektor V_1^{t-2} als dominanter Vektor ermittelt, so dass für den Makroblock MB_2^{t-1} die Summe $V_2^{t-1} + V_1^{t-2}$ abgespeichert wird. Wenn für jeden Makroblock des zu verwerfenden Bildes solch eine Summe ermittelt wurde, werden die gespeicherten Vektoren durch diese ersetzt. Wird nun zum Zeitpunkt t der nächste Frame bearbeitet und soll dieser nicht verworfen werden, so wird der ungültige Vektor V_1^t durch die Summe $[V_2^{t-1} + V_1^{t-2}] + V_1^t$ aktualisiert. Bei diesem Verfahren wird also nur eine Matrix zum Speichern der dominanten Vektoren benötigt statt eine pro verworfenem Frame bei der bilinearen Interpolation. Da die *FDVS* allerdings bei nahezu gleicher Überdeckung und bei stark unterschiedlichen Aktivitäten der beteiligten Makroblöcke schlechte Ergebnisse liefern kann, wurde in [10] eine Verbesserung dieses Verfahrens, die *Activity Dominant Vector Selection (ADVS)*, vorgeschlagen. Hierbei wird nicht nur die Größe der Überdeckung, sondern auch die Aktivität der beteiligten Makroblöcke berücksichtigt. Zum Messen der Aktivität wird die Anzahl der von Null verschiedenen *DCT*-Werte der überdeckten Teilblöcke genutzt.

4.2.2 Anpassung von Bewegungsvektoren

Die Bildinformationen, auf die sich die Bewegungsvektoren beziehen, werden bei der Transcodierung verändert. Daher kann es notwendig sein, die interpolierten Vektoren weiter anzupassen, indem eine Suche nach besseren Vektoren in einem kleinen Suchfenster um die interpolierten Vektoren herum durchgeführt wird.

Die Suche von Bewegungsvektoren basiert normalerweise auf der *Summe absoluter Differenzen (SAD)* der zugehörigen Luminanz-Pixel. Für einen optimalen Bewegungsvektor I wird in einem Referenzframe innerhalb eines Suchfensters S um den aktuellen Makroblock herum ein Punkt gesucht, für den die SAD minimal ist. Dieser Vektor ist gegeben durch

$$\begin{aligned} I &= \arg \min_{(m,n)\in S} SAD(m,n) \\ SAD(m,n) &= \sum_{i}^{M}\sum_{j}^{N} |P(i,j) - R(i+m, j+n)| \end{aligned}$$

Wobei $P(i,j)$ und $R(i,j)$ die Pixelwerte im aktuellen bzw. im Referenzframe sind und N, M die Größe des betrachteten Makroblocks angeben. Wird nun der interpolierte Bewegungsvektor als Mittelpunkt des Suchfensters gewählt, kann die Größe des Fensters erheblich reduziert werden. In [8] wurde vorgeschlagen, ein Suchfenster S^R mit einer Suchweite von ± 2 statt ± 15 Pixel bei einer vollständigen Suche zu nutzen. Ausgehend von einem Basisvektor $B = (B_x, B_y)$, d.h. dem interpolierten Vektor des aktuellen Blocks, wird dort ein Differenzvektor D gesucht, so dass die SAD minimiert wird.

$$\begin{aligned} D &= \arg \min_{(m,n)\in S^R} SAD_R(m,n) \\ SAD_R(m,n) &= \sum_{i}^{M}\sum_{j}^{N} |P(i,j) - R(i+B_x+m, j+B_x+n)| \end{aligned}$$

4.2.2.1 Schneller Suchalgorithmus

Damit nicht für alle Punkte innerhalb des Suchfensters die SAD berechnet werden muss, wurde in [8] das *Horizontal and Vertical Search (HAVS)* Schema vorgeschlagen. Hierbei wird zunächst ein Minimum in horizontaler und anschließend in vertikaler Richtung gesucht. Wird in eine Richtung ein Punkt gefunden, an dem die SAD größer ist als beim vorherigen, so wird die Suche in der entgegengesetzten Richtung fortgesetzt. Ist ein horizontales Minimum gefunden, so wird in vertikaler Richtung in der gleichen Weise gesucht. Hierbei werden mindestens fünf und maximal sieben Punkte untersucht [8]. Dieses Verfahren führt aber nicht immer zum gesuchten globalen Minimum. Daher wurde in [10] ein weiteres Suchschema vorgeschlagen, das zwar immer neun Punkte untersucht, dafür jedoch häufiger das globale Minimum findet. Anders als beim *HAVS* ist die Größe des Suchfensters hier nicht festgelegt, sondern wird anhand der Länge des benutzten Basisvektors ermittelt.

Der Suchalgorithmus kann folgendermaßen beschrieben werden:

Schritt 0: $(m, n) = (B_x, B_y)$; $//B$ ist Basisvektor
$s = \frac{|B_x|+|B_y|}{2} - 2$; // Schrittweite s

Schritt 1: $(i, j) = (m, n)$;
finde einen Punkt (m, n) mit minimaler SAD unter den Punkten $(i - s, j), (i, j), (i + s, j)$;

Schritt 2: $(i, j) = (m, n)$;
finde einen Punkt (m, n) mit minimaler SAD unter den Punkten $(i, j - s), (i, j), (i, j + s)$;

Schritt 3: $(i, j) = (m, n)$;
finde einen Punkt (m, n) mit minimaler SAD unter den Punkten $(i, j), (i - 1, j - 1), (i - 1, j + 1), (i + 1, j - 1)$;

Die Abbildung 4.5 zeigt ein Beispiel dieses Verfahrens. Die Schrittweite ergibt sich hier aus dem Basisvektor $B = (6, 4)$ zu $S = 3$, was zu einem 9×9 großes Suchfenster führt.

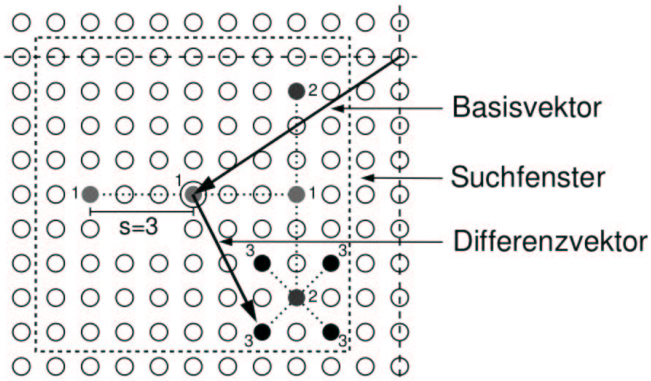


Abbildung 4.5: Schneller Suchalgorithmus zur Bewegungsvektoranpassung

4.2.2.2 Adaptive Anpassung

Eine Anpassung der Vektoren ist jedoch nicht bei allen Bewegungsvektoren sinnvoll, sondern nur dann, wenn der Transcodierungsfehler ohne eine Anpassung zu groß werden würde. Für eine adaptive Anpassung der Vektoren wurde daher in [8] ein Kriterium vorgestellt, das es erlaubt, die Notwendigkeit einer Anpassung anhand der Berechnung der so genannten *Sum of Differential Reconstruction*

Error (SDRE) zu erkennen. Dieses Kriterium berücksichtigt einerseits das Verhältnis des im Datenstrom enthaltenen Quantisierungsfaktors zu dem bei der Quantisierung im Transcoder genutzten Faktor und andererseits den entstehenden Transcodierungsfehler.

$$SDRE(B_x, B_y) = \left| \left(\frac{q_1^c}{q_2^p} \right)^2 - 1 \right| \sum_i \sum_j |R(i + B_x, j + B_y) - P(i + B_x, j + B_y)|$$

$P(i, j)$ und $R(i, j)$ sind die Pixelwerte im aktuellen bzw. im Referenzframe, q_1^c und q_2^p sind der Quantisierungsparameter des aktuellen Blocks bzw. desjenigen Blocks, auf den der Basisvektor B verweist. Der Index 2 bei q_2^p bedeutet, dass dies der bei der Quantisierung im Transcoder benutzte Parameter ist. Ist die *SDRE* für einen Makroblock größer als ein Schwellwert, so wird der Bewegungsvektor angepasst. Bewegungsvektoren der Länge Null sollten jedoch gesondert behandelt werden. Da sie weniger Bits für die Codierung benötigen, sollten Nullvektoren möglichst erhalten werden. Dies kann dadurch erreicht werden, dass für solche Vektoren ein höherer Schwellwert benutzt wird.

4.3 Bildbearbeitung im Frequenzbereich

Die Videodaten in den betrachteten Formaten liegen als *DCT*-Werte vor. Zur Erzeugung der einzelnen Pixel der Bilder ist es daher notwendig, die inverse *DCT* zu berechnen. Diese Berechnung ist jedoch sehr aufwendig, so dass der Großteil der Bearbeitungszeit eines Decoders hierfür benötigt wird. Da die transcodierten Daten wieder als *DCT*-Werte vorliegen müssen, ist es nahe liegend, die Manipulation der Bilder nicht auf den Pixelwerten, sondern im Frequenzbereich durchzuführen. Mathematisch handelt es sich bei der *DCT* um eine lineare, orthogonale Abbildung, bei der daher das Distributivgesetzt gilt. Somit kann jede lineare Abbildung der Pixel in den Frequenzbereich überführt werden. Deutlich wird dieser Sachverhalt, wenn die Bearbeitung eines Bildes als Matrizenmultiplikation einzelner Bildblöcke betrachtet wird. Hierbei wird z. B. jeder Bildblock x mit einer konstanten Abbildungsmatrix s und deren Transponierter s^t multipliziert:

$$y = sxs^t$$

Soll diese Abbildung im Frequenzbereich durchgeführt werden, so müssen die Abbildungsmatrizen ebenfalls in diesen überführt werden:

$$DCT(y) = DCT(sxs^t) = DCT(s)DCT(x)DCT(s^t)$$

Da die einzelnen Bilder einer Videosequenz in den hier betrachteten Formaten bereits als *DCT*-Blöcke vorliegen, ist es möglich, auf die Anwendung der *DCT*

und deren Inverser zu verzichten. Die nötigen Berechnungen zur Bestimmung der Abbildungsmatrizen im Frequenzbereich können im Voraus erledigt werden und fallen nicht zur Laufzeit an. Ein weiterer Vorteil der Berechnungen im Frequenzbereich ist die Tatsache, dass die DCT-Blöcke einzelner Bilder dünn besetzt sind, also nur wenige von Null verschiedene Werte enthalten. Dies kann ausgenutzt werden, um effiziente Algorithmen [4, 5] zu entwerfen, die zwischen 37% und 50% weniger Operationen als die normale Matrizenmultiplikation benötigen.

4.3.1 Inverse Bewegungskompression

Bei der Decodierung der inter-frame-codierten Makroblöcke ist es notwendig, die Bewegungskompression umzukehren. Hierzu wird ein bestimmter Bereich eines oder mehrerer Referenzbilder zum codierten Fehler des aktuellen Blocks addiert. Diese Operation wird in einem Decoder im Pixelbereich, also nach Anwendung der inversen DCT, durchgeführt. Es ist ist jedoch möglich, ein äquivalentes Verfahren im Frequenzbereich durchzuführen. Da der Bereich des Referenzbildes normalerweise nicht auf den Grenzen eines DCT-Blocks liegt, müssen die Frequenzen dieses Bereiches zunächst berechnet werden. Ein Verfahren hierzu wurde erstmals in [3] vorgestellt. Ziel ist es, aus vier benachbarten DCT-Blöcken die DCT-Werte eines Blocks zu finden, der jeweils einen Teil der vier vorhandenen Blöcke überdeckt. Ein derartiger Fall ist in Abb. 4.6 dargestellt.

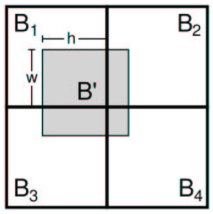


Abbildung 4.6: Berechnung von B' aus den benachbarten Blöcken $B_1 \dots B_4$

Im Pixelbereich würde man hierfür den jeweils überdeckten Teil der vier beteiligten Blöcke $B_1, \dots, B_4$ maskieren und zum gesuchten Block B' zusammensetzen:

$$b' = \sum_{i=1}^{4} h_{1i} b_i h_{2i}$$

mit

$$h_{11} = h_{12} = u_h = \begin{bmatrix} 0 & I_h \\ 0 & 0 \end{bmatrix}, \quad h_{13} = h_{14} = u_{8-h},$$

$$h_{21} = h_{23} = l_w = \begin{bmatrix} 0 & 0 \\ I_w & 0 \end{bmatrix}, \quad h_{22} = h_{24} = l_{8-w}$$

Wobei I_h und I_w jeweils Identitätsmatrizen der Dimension $h \times h$ bzw. $w \times w$, mit $0 \leq h, w \leq 8$, sind.

Die Notation der Blöcke in Kleinbuchstaben soll andeuten, dass es sich um die zugehörigen Blöcke im Pixelbereich handelt. Aufgrund der Linearität der DCT ergibt sich im Frequenzbereich

$$B' = DCT(b') = DCT(\sum_{i=1}^{4} h_{1i} b_i h_{2i}) = \sum_{i=1}^{4} DCT(h_{1i}) B_i DCT(h_{2i}). \qquad (4.1)$$

Da nur endlich viele Matrizen h_{ij} existieren, können die entsprechenden Matrizen $DCT(h_{ij})$ vorberechnet werden, so dass auf die Verwendung der DCT verzichtet werden kann. Ein Problem dieser Berechnung besteht allerdings darin, dass die Matrizen $DCT(h_{ij})$ dicht besetzt sind. Daher wurden verschiedene Algorithmen entwickelt, die die Struktur der beteiligten Matrizen ausnutzen, um Rechenzeit einzusparen. N. Merhav und V. Bhaskaran [4] haben dazu eine Zerlegung der beteiligten Matrizen vorgeschlagen, die zu einer Einsparung zwischen 37% und 50% der benötigten Rechenzeit führt. In [11] wurde stattdessen vorgeschlagen, die Elemente der Matrizen $DCT(h_{ij})$ durch Binärzahlen mit einer maximalen Abweichung von $\frac{1}{32}$ zu approximieren. Dadurch werden bei der Berechnung von (4.1) nur einfache Integeroperationen wie Rechts-Shift und Addition benötigt, was den Rechenaufwand auf 28% des in [4] benutzten Verfahrens reduziert. Eine weitere Beschleunigung der Berechnung kann dadurch erreicht werden, dass, wie in [5, 7] vorgeschlagen, bei der Berechnung nur die signifikanten niederfrequenten DCT-Werte benutzt werden.

4.3.2 Reduktion der Auflösung

Die einfachste Methode, die Auflösung von Bildern zu verringern, ist das Ersetzen benachbarter Pixel-Werte durch deren Durchschnitt. Erreicht werden kann dies durch die bereits vorgestellte bilineare Interpolation. Bei einer Skalierung um den Faktor 2 sieht diese folgendermaßen aus:

$$b' = \sum_{i=1}^{4} h_{1i} b_i h_{2i}$$

mit

$$h_{11} = h_{12} = h_{21}^t = h_{23}^t = \begin{bmatrix} q_{4\times 8} \\ 0_{4\times 8} \end{bmatrix},$$

$$h_{13} = h_{14} = h_{22}^t = h_{24}^t = \begin{bmatrix} 0_{4\times 8} \\ q_{4\times 8} \end{bmatrix},$$

mit

$$q_{4\times 8} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \text{ und einer } 4 \times 8 \text{ Nullmatrix } 0_{4\times 8}.$$

Aufgrund der Linearität der DCT kann diese Abbildung in den Frequenzbereich überführt werden:

$$B' = \sum_{i=1}^{4} DCT(h_{1i} b_i h_{2i}) = \sum_{i=1}^{4} DCT(h_{1i}) DCT(b_i) DCT(h_{2i}) = \sum_{i=1}^{4} H_{1i} B_i H_{2i}$$

Die Matrizen H_{ij} können hierbei vorberechnet werden, so dass hierfür keine zusätzliche Rechenzeit zur Laufzeit anfällt. Leider führt die Tatsache, dass die Matrizen h_{ij} dünn besetzt sind, dazu, dass H_{ij} dicht besetzt ist. Daher wurde in [18] eine Methode vorgeschlagen, die nur die ersten 4×4 DCT-Werte der benachbarten Blöcke benutzt. Diese werden mit der inversen DCT in den Frequenzbereich überführt, dort zu einem neuen 8×8 Block zusammen gefügt und anschließend wieder mit der DCT in den Frequenzbereich überführt. Dieses Verfahren liefert eine signifikante Beschleunigung gegenüber der bilinearen Interpolation. Außerdem liefert es visuell bessere Ergebnisse, da die niedrigen DCT-Frequenzen die meisten Informationen der Pixel tragen. R. Dugad und N. Ahuja überführten dieses Verfahren in [6] vollständig in den Frequenzbereich:

Seien B_1, B_2, B_3, B_4 die 8×8 DCT-Blöcke und $\hat{B}_1, \hat{B}_2, \hat{B}_3, \hat{B}_4$ die zugehörigen 4×4 Subblöcke, die die niedrigen Frequenzen enthalten. Weiter sei $\hat{b_i} = DCT^{-1}(\hat{B}_i)$, $i = 1, \ldots, 4$. Dann ist

$$\hat{b} = \begin{bmatrix} \hat{b}_1 & \hat{b}_2 \\ \hat{b}_3 & \hat{b}_4 \end{bmatrix}_{8\times8}$$

die um den Faktor 2 skalierte Version von

$$b = \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix}_{16\times16}$$

Dann kann $\hat{B} \stackrel{\text{def}}{=} DCT(\hat{b})$ direkt aus den Blöcken B_1, B_2, B_3, B_4 berechnet werden:

$$\begin{aligned} \hat{B} &= T\hat{b}T^t \\ &= \begin{bmatrix} T_L & T_R \end{bmatrix} \begin{bmatrix} \hat{b}_1 & \hat{b}_2 \\ \hat{b}_3 & \hat{b}_4 \end{bmatrix} \begin{bmatrix} T_L^t & T_R^t \end{bmatrix} \\ &= \begin{bmatrix} T_L & T_R \end{bmatrix} \begin{bmatrix} T_4^t \hat{B}_1 T_4 & T_4^t \hat{B}_2 T_4 \\ T_4^t \hat{B}_3 T_4 & T_4^t \hat{B}_4 T_4 \end{bmatrix} \begin{bmatrix} T_L^t & T_R^t \end{bmatrix} \\ &= (T_L T_4^t)\hat{B}_1 (T_L T_4^t)^t + (T_L T_4^t)\hat{B}_2 (T_R T_4^t)^t + (T_R T_4^t)\hat{B}_3 (T_L T_4^t)^t + \\ &\quad (T_R T_4^t)\hat{B}_4 (T_R T_4^t)^t \end{aligned} \tag{4.2}$$

Wobei $T = [T_L | T_R]$ die 8×8 DCT-Operatormatrix mit

$$T_{i,j} = \begin{cases} \frac{1}{2\sqrt{2}}, & i = 0,\ 0 \le j \le 7 \\ \frac{1}{2} \cos \frac{\pi(2j+1)i}{16}, & 1 \le i \le 7,\ 0 \le j \le 7 \end{cases}$$

und T_4 die entsprechende 4×4 DCT-Matrix ist.

Zusätzlich wurde der Rechenaufwand für die Berechnung von (4.2) in [6] durch eine Dekomposition der beteiligten Matrizen weiter reduziert. Dies führt dazu, dass pro Pixel des Originalbildes nur noch 1,25 Additionen und 1,25 Multiplikationen notwendig sind, um das Bild zu skalieren.

4.3.3 Einschränkungen

Die Bearbeitung im Frequenzbereich hat einige Vorteile bzgl. des notwendigen Rechenaufwandes. Trotzdem ist es nicht in allen Situationen sinnvoll, eine Berechnung im Frequenzbereich durchzuführen. Stattdessen muss für eine Transcoderarchitektur individuell geprüft werden, welche weiteren Berechnungen durchgeführt werden müssen. So ist es z. B. nicht sinnvoll, eine Bildbearbeitung oder inverse Bewegungskompression im Frequenzbereich durchzuführen, wenn die Bewegungsvektoren, wie in Abschnitt 4.2.2 vorgestellt, angepasst werden sollen. Denn für diese Anpassung sind die Pixel der einzelnen Blöcke notwendig. Außerdem bleibt die Berechnung im Frequenzbereich trotz der vorgestellten Möglichkeiten zur Vereinfachung eine sehr komplexe und aufwendige Operation. In einigen Situationen kann auch eine Kombination aus Bildbearbeitung im Frequenzbereich und Berechnungen der Pixel sinnvoll sein. In den folgenden Abschnitten werden z. B. einige Architekturen vorgestellt, bei denen solch eine Kombination eingesetzt wird.

4.4 Transcoderarchitekturen

Nachdem in den letzten Abschnitten einige grundlegende Techniken für den Transcodingvorgang erläutert wurden, werden nun konkrete, in der Literatur dokumentierte Architekturen vorgestellt. Die verschiedenen Transcoder werden jeweils als schematisches Blockschaltbild dargestellt. Innerhalb dieser Darstellungen werden für die verschiedenen Operationen die folgenden Bezeichnungen benutzt:

- VLC = Value Length Coding
- Q = Quantisierung
- DCT = Diskrete Cosinus Transformation
- FB = Frame Buffer
- MC = Motion Compensation
- MV = Motion Vector

Eine inverse Operation wird mit einer hochgestellten -1 dargestellt.

4.4.1 Requantisierung

Eine Möglichkeit zur Reduzierung der Datenrate eines Videostromes ist die Erhöhung der Kompression. Dies kann erreicht werden, indem die Quantisierung der *DCT*-Werte (siehe 2.1.2.1) mit einem höheren Quantisierungsparameter erneut durchgeführt wird. Betrachten wir hierzu zunächst einen Decoder mit nachgeschaltetem Encoder, der bereits die Bewegungsinformationen des Videostromes wiederverwendet. Diese Architektur, die als *Cascaded Pixel-Domain Transcoder (CPDT)* bezeichnet wird, ist in Abb. 4.7 als Blockschaltbild dargestellt.

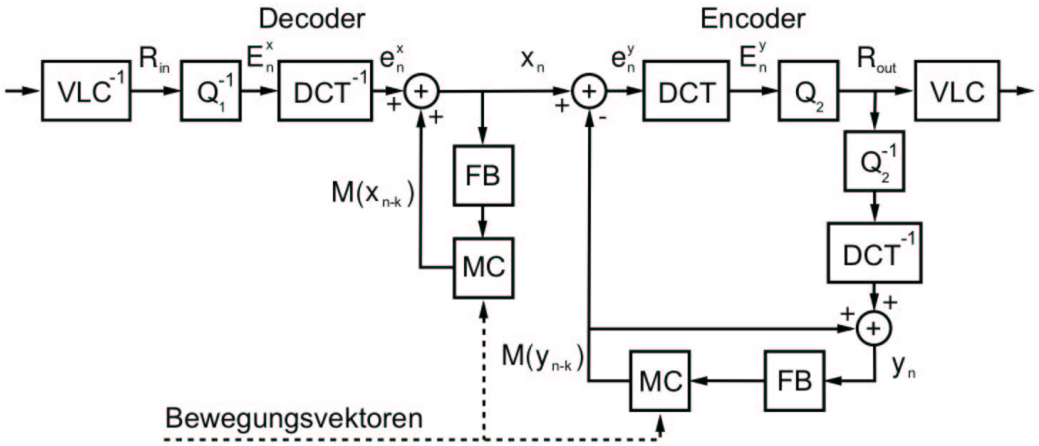


Abbildung 4.7: Cascaded Pixel-Domain Transcoder (*CPDT*)

In [11] wurde gezeigt, dass dieser Transcoder erheblich vereinfacht werden kann, indem sämtliche Operationen im Frequenzbereich durchgeführt werden. Aufgrund der Tatsache, dass I-Bilder keine Bewegungsinformationen enthalten, gilt für R^I_{out}, den transcodierten Datenstrom vor der VLC:

$$R^I_{out} = Q_2[DCT(x_n)], \tag{4.3}$$

wobei der decodierte Frame x_n sich in Abhängigkeit von R^I_{in}, dem ankommenden Datenstrom nach der VLC, berechnet zu

$$x_n = DCT^{-1}[Q_1^{-1}(R^I_{in})]. \tag{4.4}$$

Wird nun Gleichung (4.4) in (4.3) eingesetzt, erhält man für das Transcoding von I-Frames

$$R^I_{out} = Q_2[Q_1^{-1}(R^I_{in})], \tag{4.5}$$

Die in Abb. 4.7 dargestellte Rückkopplungsschleife zur Bewegungskompensation wird also nicht verwendet.

Bei P-Bildern hingegen wird diese benutzt, da diese Bilder Bewegungsinformationen bzgl. früherer Bilder enthalten. Bei der Bearbeitung des P-Frames n wird also das Bild $(n - N)$ benötigt, wobei N der Abstand zum früheren Referenzbild ist. Aus Abb. 4.7 lassen sich nun die folgenden Beziehungen für den ankommenden und den transcodierten Datenstrom bei P-Frames ablesen.

$$e_n^x = DCT^{-1}[Q_1^{-1}(R_{in}^P)] \tag{4.6}$$
$$R_{out}^P = Q_2[DCT(e_n^y)] \tag{4.7}$$

Wobei das decodierte Bild x_n und der transcodierte Fehler e_n^y gegeben sind durch

$$x_n = e_n^x + M_P(x_{n-N}) \tag{4.8}$$
$$e_n^y = x_n - M_P(y_{n-N}). \tag{4.9}$$

Die Funktion M_P stellt hierbei die Bewegungskompensation dar. Setzt man nun die Gleichungen (4.8) und (4.9) ineinander ein, so erhält man

$$e_n^y = e_n^x + M_P(x_{n-N}) - M_P(y_{n-N}). \tag{4.10}$$

Aus der Gleichung (4.10) ist zu erkennen, dass der transcodierte Fehler e_n^y aus dem ankommenden Fehler e_n^x berechnet werden kann, indem die Differenz der beiden Bewegungskompensationen hinzu addiert wird. Bei dieser Differenz handelt es sich also offensichtlich um den Transcodierungsfehler. Da die beiden Schleifen zur Bewegungskompensation die gleichen Bewegungsvektoren benutzen, lässt sich (4.10) weiter vereinfachen zu:

$$e_n^y = e_n^x + M_P(x_{n-N} - y_{n-N}) \tag{4.11}$$

Hierbei ist jedoch anzumerken, dass die Bewegungskompensation M_P keine lineare Operation darstellt und bei der Vereinfachung zu (4.11) Fehler eingeführt werden. Diese sind jedoch so gering, dass sie vernachlässigt werden können [11]. Wird nun die Gleichung (4.11) in (4.7) unter Berücksichtigung der Linearität der DCT eingesetzt, erhält man die endgültige Gleichung zum Transcodieren von P-Bildern:

$$R_{out}^P = Q_2[E_n^x + DCT(M_P(x_{n-N} - y_{n-N}))]. \tag{4.12}$$

Für B-Bilder gelten die gleichen Beziehungen wie bei P-Bildern, wobei sich die Bewegungskompensation allerdings nicht nur auf ein vorheriges, sondern auch auf ein nachfolgendes Referenzbild bezieht. Daher sieht die Transcodierungs-Gleichung (4.12) für B-Frames folgendermaßen aus:

$$R_{out}^P = Q_2[E_n^x + DCT(M_P(x_{n-N} - y_{n-N}, x_{n+K} - y_{n+K}))]. \tag{4.13}$$

K stellt hierbei den Abstand zum nachfolgenden Referenzbild dar. Diese Überlegungen führen zusammen mit der Bewegungskompensation im Frequenzbereich

(siehe Abschnitt 4.3.1) zum *DCT-Domain Transcoder (DDT)*, der als Blockschaltbild in Abb. 4.8 dargestellt ist. Wie bereits anhand der Gleichungen (4.12) und (4.13) zu erkennen ist, enthält in dieser Architektur der Speicher vor der Bewegungskompensation nicht die rekonstruierten Frames, sondern die Differenz der beiden ursprünglichen Bewegungskompensationen aus Abb. 4.7.

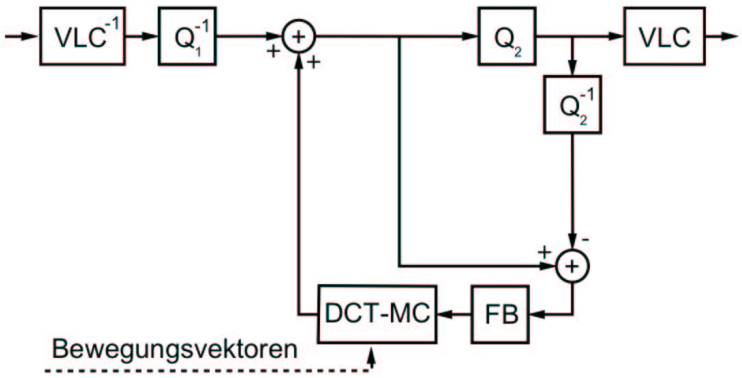


Abbildung 4.8: DCT-Domain Transcoder (*DDT*)

Ein Nachteil dieser Architektur besteht jedoch in ihrer geringen Flexibilität. Eine Zusammenfassung der beiden Schleifen zur Bewegungskompression war nur möglich, da die gleichen Bewegungsinformationen benutzt wurden. Werden diese Informationen aber durch den Transcodingvorgang verändert, ist diese Voraussetzung nicht mehr gegeben und diese Architektur nicht nutzbar.

4.4.2 Frameskipping

Ein weiterer Ansatz, die Datenrate eines Videostroms zu reduzieren, ist das Verwerfen einzelner Frames, so dass die Bildrate der Videosequenz verringert wird. Dieses Verfahren wird als Frameskipping bezeichnet. B-Frames können ohne Auswirkung auf die übrigen Bilder verworfen werden, da diese nicht als Referenzen für andere Bilder dienen. Sollen auch P- oder I-Bilder verworfen werden, entsteht jedoch das Problem, dass die Bewegungsinformationen nicht verworfener Frames, die sich auf verworfene Frames beziehen, ungültig werden. Eine Anpassung der Bewegungsvektoren, wie in Abschnitt 4.2.1 dargestellt, und der codierten Fehler ist somit unerlässlich. Somit kann die Architektur aus dem vorherigen Abschnitt nicht genutzt werden. In [16] wurde daher eine Architektur eines Transcoders vorgestellt, die teilweise im Frequenzbereich arbeitet. Das Blockschaltbild ist in Abb. 4.9 dargestellt. Der Schalter S_3 regelt in dieser Abbildung, ob ein Frame verworfen wird (Stellung B) oder an den Client übertragen wird (Stellung A). Ankommende I-Frames werden durch das Frameskipping nicht beeinflusst, da sie keine Bewegungskompression enthalten, die aktualisiert werden muss. Daher beschränken sich die folgenden Betrachtungen ausschließlich auf P- und B-Frames.

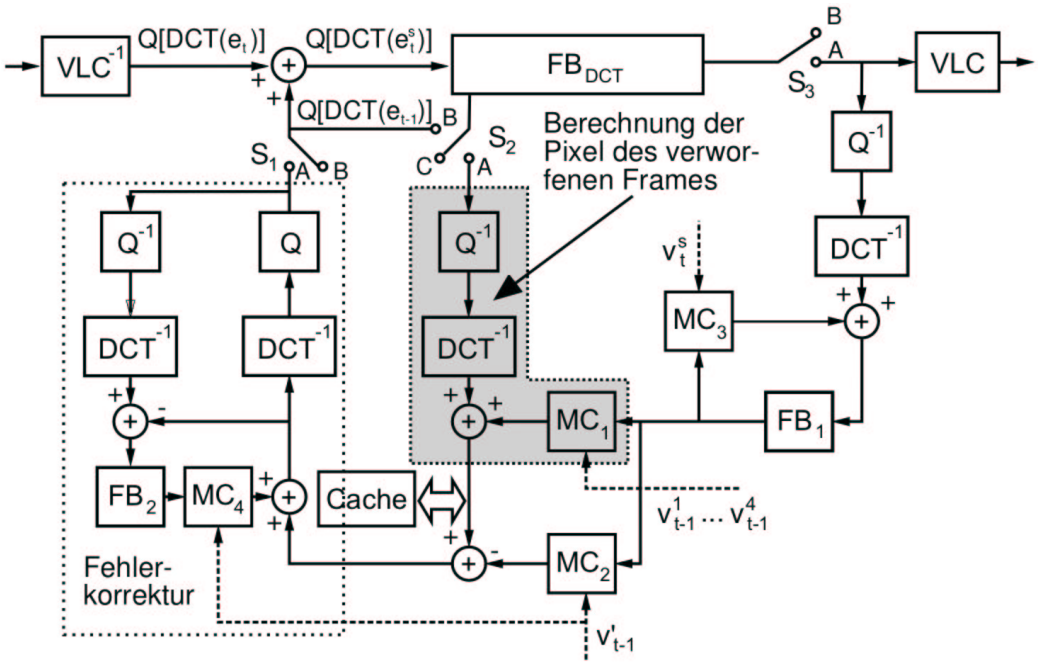


Abbildung 4.9: Frameskipping-Transcoder

In realen Videosequenzen treten häufig unbewegte Hintergründe auf, was zu Makroblöcken führt, deren Bewegungsvektor der Nullvektor ist [16]. Solche Blöcke werden im Folgenden auch als Makroblöcke ohne Bewegungskompression bezeichnet, da der zugehörige Block des Referenzbildes nicht bestimmt werden muss. Diese Blöcke dürfen jedoch nicht mit intra-frame-codierten Blöcken verwechselt werden. Intra-frame-codierte Makroblöcke werden durch den Transcoder nicht verändert, sondern nur für eine spätere Nutzung als Referenzblöcke folgender Bilder gespeichert. In Abb. 4.9 wird dies durch die gezeigte initiale Stellung der Schalter S_1 und S_2, in den Positionen B bzw. C erreicht.

4.4.2.1 Makroblöcke ohne Bewegungskompression

Betrachten wir zunächst Makroblöcke ohne Bewegungskompression. Eine Situation, in der ein Frame verworfen wurde, ist in Abb. 4.10(a) schematisch dargestellt. Aufgrund der Anpassung der Bewegungsvektoren beim Verwerfen einzelner Bilder, ist es notwendig, den codierten Fehler des aktuellen Makroblocks anzupassen. Da der Bewegungsvektor Null ist, gilt:

$$MB_t = MB_{t-1} + e_t \tag{4.14}$$

$$MB_{t-1} = B_{t-2} + e_{t-1} \tag{4.15}$$

wobei e_t bzw. e_{t-1} jeweils der codierte Fehler des Blocks MB_t bzw. MB_{t-1} ist.

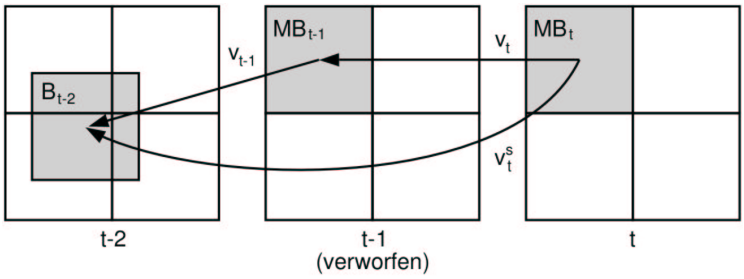
Der gesuchte Fehler e_t^s des aktuellen Blocks MB_t bzgl. des letzten nicht verworfenen Frames ergibt sich aus:

$$e_t^s = MB_t - B_{t-2} = e_t + e_{t-1} \tag{4.16}$$

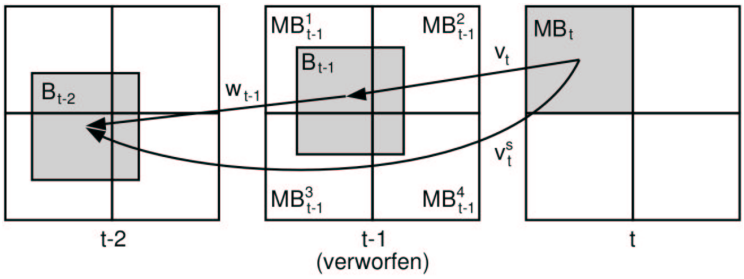
Der neue Fehlerblock kann also durch Addition des aktuellen Fehlers mit dem Fehler des verworfenen Frames berechnet werden. Diese Berechnung kann aufgrund der Linearität der DCT im Frequenzbereich durchgeführt werden. Da sich in dieser Transcoderarchitektur der Quantisierungsparameter nicht ändert, kann die Berechnung direkt mit den quantisierten DCT-Werten durchgeführt werden[1]:

$$Q[DCT(e_t^s)] = Q[DCT(e_t)] + Q[DCT(e_{t-1})]$$

In dem Blockschaltbild in Abb. 4.9 wird dies dadurch erreicht, dass bei Makroblöcken ohne Bewegungskompression die Schalter S_1 und S_2 jeweils in Position B gestellt werden. Der neue Bewegungsvektor v_t^s für den aktuellen Makroblock ergibt sich aus dem Vektor v_{t-1}.



(a) ohne Bewegungskompression



(b) mit Bewegungskompression

Abbildung 4.10: Berechnung des Fehlerblocks bei Makroblöcken (a) ohne und (b) mit Bewegungskompression

[1]Es ist anzumerken, dass die Quantisierung keine lineare Operation darstellt, in diesem Fall jedoch aufgrund gleicher Parameter das Distributivgesetzt angewandt werden kann.

4.4.2.2 Makroblöcke mit Bewegungskompression

Bei Makroblöcken, die einen von Null verschiedenen Bewegungsvektor besitzen, ist diese einfache Berechnung nicht möglich, da der Block des vorherigen Frames, auf den der aktuelle Bewegungsvektor verweist, normalerweise nicht als Makroblock vorliegt. Solch eine Situation ist in Abb. 4.10(b) dargestellt. Für die Aktualisierung des Fehlerblocks e_t^s ist es notwendig, die Pixel von B_{t-1} aus den umliegenden Makroblöcken zu bestimmen. Hierzu werden in der Abb. 4.9 die Schalter S_1 und S_2 jeweils in die Position A gebracht, so dass im grau unterlegten Teil des Schaltbildes die Pixel des Blocks B_{t-1} bestimmt werden. Die Fehlerblöcke der umliegenden Blöcke $MB_{t-1}^1, \ldots, MB_{t-1}^4$, die sich im Speicher FB_{DCT} befinden, werden durch inverse Quantisierung und DCT bestimmt und zu den zugehörigen Blöcken des Referenzframes $(t-2)$ addiert. Es werden jedoch nur die Pixel bestimmt, die von B_{t-1} überdeckt werden. Anschließend (am unteren Ende des grauen Bereichs in Abb. 4.9) liegen also die Pixel von B_{t-1} vor. Zur weiteren Bestimmung des Fehlers e_t^s wird der Block B_{t-2} des letzten nicht verworfenen Frames und hierzu der Bewegungsvektor w_{t-1} benötigt. Statt w_{t-1} aus den umliegenden Vektoren zu bestimmen, wird in [16] die *Forward Dominant Vector Selection* genutzt, um einen dominanten Vektor v'_{t-1} zu bestimmen (siehe auch Abschnitt 4.2.1.2). Mit Hilfe des dominanten Vektors wird die Differenz der berechneten Pixel von B_{t-1} und der Pixel des entsprechenden Blocks des Frames $(t-2)$ bestimmt. Der so berechnete Fehler e_t^s muss anschließend durch Anwendung der DCT in den Frequenzbereich überführt und quantisiert werden. Durch die erneute Quantisierung entsteht allerdings ein zusätzlicher Fehler, der sich bei der Transcodierung zukünftiger Frames akkumuliert. Daher wird versucht, diesen Fehler mit Hilfe einer zusätzlichen Schleife zu eliminieren.

Zur Vermeidung einer mehrfachen Berechnung der Pixel eines verworfenen Frames wurde zusätzlich ein Cache vorgeschlagen, der die im grau unterlegten Teil des Schaltbildes berechneten Pixel speichert. Dieser Cache kann den Rechenaufwand stark reduzieren, da Experimente gezeigt haben, dass bei realen Videosequenzen sehr häufige Cache-Hits auftreten [16].

Insgesamt benötigt diese Architektur drei Speicher, um die einzelnen Frames bzw. den akkumulierten Quantisierungsfehler für eine spätere Benutzung abzulegen. FB_{DCT} enthält die aktualisierten quantisierten DCT-Werte der ankommenden Frames, die an den Client übertragen werden oder andernfalls für die Anpassung folgender Frames benötigt werden. Bei Makroblöcken ohne Bewegungskompression wird der zugehörige Block des vorherigen Frames direkt aus diesem Speicher zum aktuellen Block addiert. Der Speicher FB_1 enthält die Pixel der nicht verworfenen Frames, die als Referenzbilder für nachfolgende Frames dienen. Bei Makroblöcken mit Bewegungskompression ist es notwendig, den entstehenden Quantisierungsfehler zu kompensieren. Daher wird der akkumulierte Fehler im Speicher FB_2 abgelegt, damit dieser bei der Bearbeitung nachfolgender Blöcke eliminiert werden kann.

4.4.2.3 Dynamisches Frameskipping

Wird die Framerate eines Videos durch Frameskipping reduziert, muss auch entschieden werden, welche Bilder im Transcoder verworfen werden sollen. Ein konstanter Abstand zwischen nicht verworfen Frames ist hierfür sicherlich der einfachste Ansatz. Die Bewegung innerhalb eines Videos ist aber in den seltensten Fällen über die gesamte Darstellungszeit gleich verteilt. Stattdessen gibt es in realen Videosequenzen Abschnitte mit höherer und mit geringerer Bewegung. Wird nun der Abstand zwischen nicht verworfenen Frames konstant gewählt, werden in den Teilen des Videos mit starker Bewegung die gleiche Anzahl Bilder verworfen wie in Teilen mit geringer Bewegung. Dieses Vorgehen kann jedoch zu ruckartigen Bewegungsabläufen führen. Sinnvoll wäre es also, für die Abschnitte mit starker Bewegung eine größere Anzahl Bilder zu codieren als für Teile mit geringer Bewegung, um insgesamt einen möglichst weichen Bewegungsablauf zu erhalten. Also sollte der Transcoder nach dem Verwerfen des ersten Frames die Bewegungsaktivität der folgenden Bilder messen, um entscheiden zu können, welcher Frame als nächstes codiert werden soll. In [16] wurde hierfür eine Metrik eingeführt, die neben der Bewegungsaktivität auch den Transcodierungsfehler berücksichtigt:

$$FSC_t^s = \frac{\sum_{m=1}^{M} (MA_t^s)_m}{\sum_{m=1}^{M} (RE_{t-1}^s)_m}$$

Hierbei ist M die Anzahl der Makroblöcke des aktuellen Frames zum Zeitpunkt t. Die Bewegungsaktivität $(MA_t^s)_m$ sowie der Transcodierungsfehler $(RE_t^s)_m$ des m-ten Makroblocks ist jeweils gegeben durch

$$(MA_t^s)_m = |(u_t^s)_m| + |(v_t^s)_m| \tag{4.17}$$

$$(RE_t^s)_m = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \Delta_{t-1}^s(i,j) \tag{4.18}$$

Die Werte $(u_t^s)_m$ und $(v_t^s)_m$ bezeichnen hierbei die horizontale und vertikale Komponente des Bewegungsvektors des m-ten Makroblocks, N ist dessen Größe und $\Delta_{t-1}^s(i,j)$ der (i,j)-Koeffizient des Requantisierungsfehlers, der im Transcoder in der Schleife zur Fehlerkorrektur berechnet wird. Liegt der Wert dieser Metrik für einen ankommenden Frame über einem Schwellwert, so sollte dieser Frame codiert und nicht verworfen werden. Dieser Schwellwert kann anhand der Datenrate des transcodierten Stroms an die Bandbreite des Clients angepasst werden, damit diese möglichst effizient ausgenutzt wird. Liegt also die Datenrate am Ausgang des Transcoders unterhalb der Bandbreite, wird der Schwellwert verringert und andernfalls erhöht.

4.4.3 Skalierung

Mobile Geräte besitzen häufig nur ein relativ kleines Display, das nicht in der Lage ist, eine hochauflösende Videosequenz anzuzeigen. Somit ist es notwendig, die einzelnen Bilder des Videos in ihrer Auflösung zu verringern. Wie bereits in Abschnitt 4.3.2 gezeigt, ist es möglich, die Bilder innerhalb des Frequenzbereichs zu skalieren. Diese Technik führt zu der in [31] vorgestellten Architektur eines Transcoders, die in Abb. 4.11 dargestellt ist. Aus den ankommenden Daten werden zunächst innerhalb der ersten Schleife mit Hilfe der inversen Bewegungskompression, wie im Abschnitt 4.3.1 dargestellt, die Abhängigkeiten der Frames untereinander entfernt. Die dann vorliegenden DCT-Blöcke der einzelnen Bilder werden im Frequenzbereich skaliert. Gleichzeitig werden die Bewegungsvektoren aus dem Datenstrom interpoliert und skaliert, um anschließend in der zweiten Schleife die Abhängigkeiten der Frames zu erzeugen. Bei der abschließenden Quantisierung kann für eine weitere Verringerung der Datenmenge ein höherer Quantisierungsparameter gewählt werden, so dass zusätzlich eine Requantisierung der Daten durchgeführt wird.

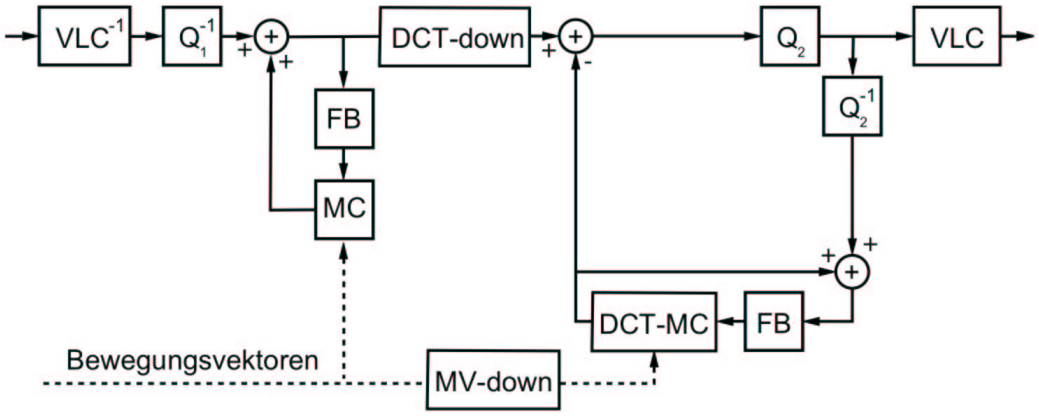


Abbildung 4.11: DCT-Domain Transcoder zur Skalierung

Aufgrund der Tatsache, dass diese Architektur vollständig im Frequenzbereich arbeitet, ist der entstehende Transcodierungsfehler jedoch relativ hoch. Um dieses Problem zu beheben, müssten die skalierten Bewegungsvektoren, wie in Abschnitt 4.2.2 erläutert, weiter angepasst werden. Diese Anpassung ist jedoch nicht im Frequenzbereich möglich, da zur Berechnung der SAD die Pixeldaten benötigt werden. Dies führt zu einer in [14] und [12] vorgestellten Architektur, die nur für I-Bilder im Frequenzbereich arbeitet. P- und B-Bilder werden hingegen, wie in Abb. 4.12 dargestellt, zunächst vollständig decodiert, skaliert und anschließend wieder codiert. Die Bewegungsvektoren werden, wie im Abschnitt 4.2.1 dargestellt, interpoliert und anschließend adaptiv angepasst. Wie bereits am Blockschaltbild zu erkennen, ist diese Architektur sehr rechenintensiv und daher schlecht für

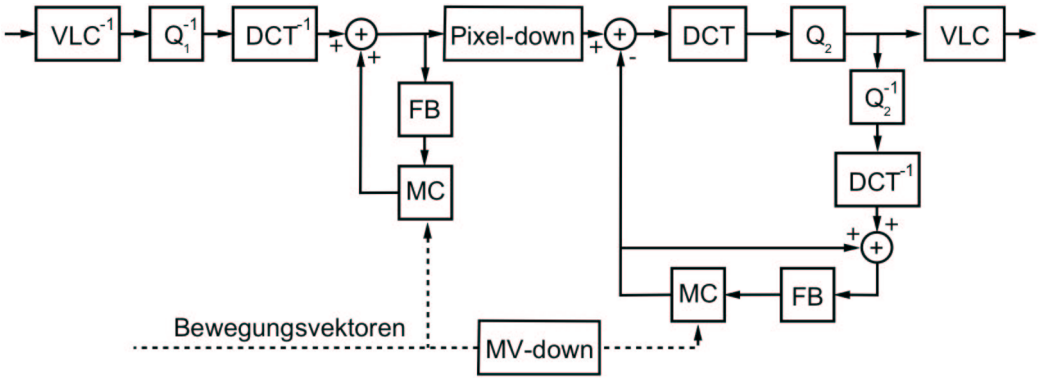


Abbildung 4.12: Pixel-Domain Transcoder zur Skalierung

ein Transcoding in Echtzeit zu benutzen. Diese Architektur wurde jedoch von B. Shen und S. Roy in [34] weiter angepasst und beschleunigt. Die Beschleunigung beruht darauf, dass die Skalierung im Wesentlichen im Frequenzbereich durchgeführt wird und dass das nicht skalierte Bild für die Bewegungskompression durch eine Vergrößerung des skalierten Bildes approximiert wird. Dadurch muss z. B. bei einem Skalierungsfaktor 2 die inverse DCT nur auf ein Viertel der ursprünglichen Blöcke angewandt werden. Bei der Skalierung von P- und B-Bildern wird in Abhängigkeit vom Typ (inter- oder intra-frame-codiert) der beteiligten Makroblöcke entschieden, welchen Typ der skalierte Makroblock hat. Zur Vereinfachung werden im Folgenden auch die Bezeichnungen Intra-Block und Inter-Block für intra-frame-codierte bzw. inter-frame-codierte Blöcke benutzt.

Angenommen beim Skalierungsprozess werden k Makroblöcke zu einem neuen Block zusammengefasst. Der skalierte Block ist ein Intra-Block, wenn mindestens $m \leq k$ der beteiligten Blöcke intra-frame-codiert sind. Ist dies der Fall, so wird anhand eines weiteren Wertes n mit $m \leq n \leq k$ entschieden, welches Verfahren zur Skalierung genutzt wird. Insgesamt werden also vier Fälle unterschieden:

Fall 1: weniger als m der beteiligten Makroblöcke sind intra-frame-codiert

Fall 2: i mit $m \leq i < n$ der beteiligten Makroblöcke sind intra-frame-codiert

Fall 3: i mit $n \leq i < k$ der beteiligten Makroblöcke sind intra-frame-codiert

Fall 4: alle k beteiligten Makroblöcke sind intra-frame-codiert

Im ersten Fall wird der skalierte Block ein Inter-Block und in den letzten drei Fällen ein Intra-Block. Bei einer Skalierung um den Faktor 2 werden jeweils $k = 4$ Makroblöcke zu einem neuen Block zusammengefasst. Dann können m und n als $m = 2$ und $n = 3$ gewählt werden.

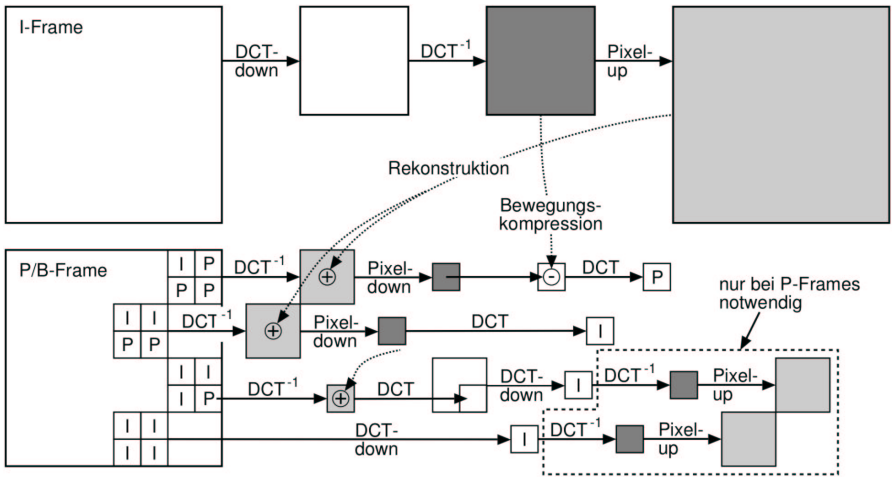


Abbildung 4.13: Schnelle Transcoderarchitektur zur Skalierung

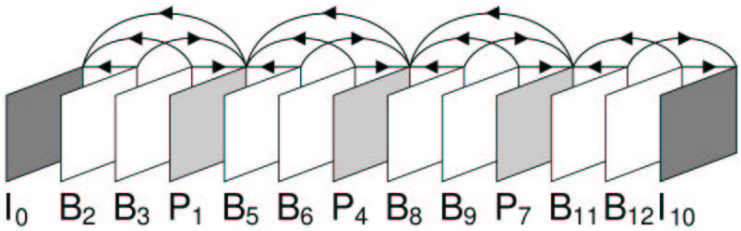
Der schematische Ablauf des Verfahrens für diese Werte ist in Abb. 4.13 dargestellt. Die dunkelgrau unterlegten Blöcke stellen jeweils die skalierten Pixelblöcke und die hellgrau unterlegten die entsprechenden Pixelblöcke in der ursprünglichen Auflösung dar. Wurden diese Blöcke zuvor im Frequenzbereich skaliert ist es bei I- und P-Bildern für die Bewegungskompression nachfolgender Bilder notwendig, diese wieder in die ursprüngliche Auflösung zu skalieren. In den ersten beiden Fällen ist dies unnötig, da die Pixel der nicht skalierten Blöcke zuvor bereits mit Hilfe der inversen DCT erzeugt wurden.

Ein ganz ähnlicher Ansatz wurde in [13] als Intra-Refresh-Transcoder vorgeschlagen, der ebenfalls anhand der beteiligten Makroblöcke entscheidet, welchen Typ der skalierte Block hat. Die Zielsetzung dieses Ansatzes ist es jedoch primär, den akkumulierten Transcodierungsfehler zu reduzieren. Da bei Intra-Blöcken die Fehlerfortpflanzung gestoppt wird (es wird keine Bewegungskompression eingesetzt) wird auch abhängig von der verfügbaren Datenrate entschieden, wann ein Intra-Block erzeugt wird. Die Qualität dieser Architektur ist jedoch etwas geringer, als die des in [34] vorgeschlagenen Transcoders, da eine Anpassung der Bewegungsvektoren bei Inter-Blöcken nicht stattfindet.

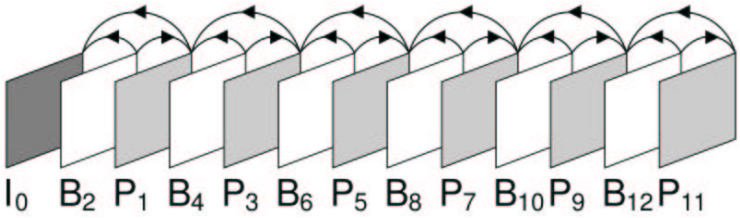
4.4.4 Heterogenes Transcoding

In manchen Situationen kann es notwendig sein, den ankommenden Datenstrom in ein anderes Format umzuwandeln. Ist der Client z. B. nur in der Lage, H.261 oder H.263 anzuzeigen, obgleich ein Videostrom aber nur in MPEG-1/2 vorliegt, so müssen die Daten in das gewünschte Format umcodiert werden. Hierbei ist es einerseits notwendig, die Syntax an das andere Format anzupassen, andererseits aber auch, die Frametypen umzuwandeln. H.261 unterstützt z. B. keine bidirek-

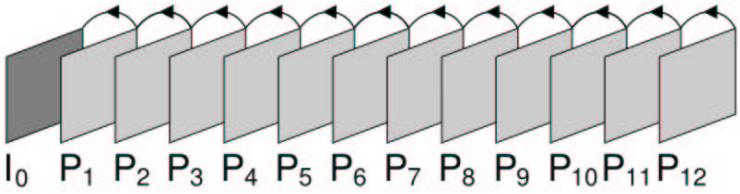
tionale Bewegungskompression. Die Umwandlung der Syntax wird hier jedoch nicht näher erläutert, da dies keinen großen Aufwand erfordert. Stattdessen konzentriert sich dieser Abschnitt auf die Umwandlung der Frametypen. In [18] wurden von T. Shanableh und M. Ghanbari exemplarisch zwei Situationen erläutert, in denen eine Typumwandlung notwendig ist. Ausgehend von einem MPEG-2-Strom mit einer *GOP* der Länge $N = 12$ und einer Länge der Untergruppen von $M = 3$, wird die Umwandlung in einen Strom mit $N = \infty$ und $M = 1$ bzw. $N = \infty$ und $M = 2$ betrachtet. Die Anordnung der einzelnen Bilder ist in Abb. 4.14 dargestellt. Die einzelnen Frames sind in der Reihenfolge dargestellt, in der sie angezeigt werden, nummeriert sind sie jedoch in Codierreihenfolge. Betrachten wir den Fall $N = \infty$ und $M = 1$ für den transcodierten Datenstrom. Hierbei ist es notwendig, für einige Frames neue Bewegungsvektoren zu finden. Für den ersten B-Frame einer Untergruppe, z. B. B_5 in Abb. 4.14(a), muss aus den zwei vorhandenen Vektoren $v_{1\leftarrow 5}$ und $v_{5\rightarrow 4}$ ein neuer rückwärts gerichteten Vektor $v_{3\leftarrow 4}$ für den Frame P_4 bestimmt werden. Hierzu werden mehrere Kandidaten bestimmt, unter denen anhand der *SAD* der beste ausgewählt wird.



(a) umzuwandelnder Strom ($N = 12,\ M = 3$)



(b) umgewandelter Strom ($N = \infty,\ M = 2$)



(c) umgewandelter Strom ($N = \infty,\ M = 1$)

Abbildung 4.14: Frametypen des umzuwandelnden und umgewandelten Stroms in Darstellungsreihenfolge, nummeriert in Codierreihenfolge

Für den eben genannten Fall können z. B. die folgenden Vektoren als Kandidaten gewählt werden:

- der rückwärts gerichtete Vektor von B_5: $v_{1\leftarrow 5}$
- die Summe aus dem vorwärts gerichteten Vektor von B_5 und dem rückwärts gerichteten Vektor des Referenzframes P_4: $v_{5\rightarrow 4} + v_{4\leftarrow 1}$
- ein Drittel des Vektor des zukünftigen Referenzframes von B_5: $\frac{1}{3}v_{4\leftarrow 1}$
- die Hälfte des inversen vorwärts gerichteten Vektor von B_5: $-\frac{1}{2}v_{5\rightarrow 4}$
- die Hälfte des rückwärts gerichteten Vektors des folgenden B-Frames: $\frac{1}{2}v_{6\leftarrow 1}$

Anhand dieser Kandidaten ist zu erkennen, dass die Bewegung zwischen den einzelnen Bildern in [18] als uniforme Bewegung angenommen wird. Sollten also einige der oben genannten Vektoren nicht verfügbar sein, kann die Liste beliebig erweitert werden bis eine ausreichende Anzahl Kandidaten gefunden wurde. In ähnlicher Art und Weise werden auch für die anderen Fälle jeweils Kandidaten für einen neuen Bewegungsvektor gesucht und anhand der daraus resultierenden *SAD* ausgewählt. Zusätzlich können die jeweils gefundenen Vektoren, wie in Abschnitt 4.2.2 beschrieben, weiter angepasst werden. Experimente haben ergeben, dass eine Suchfenstergröße von 1 hierbei sehr gute Ergebnisse liefert [18].

4.4.5 Kontrolle der Bitrate

Ein Ziel des Transcodings ist die Reduktion der Datenrate, um einen Videostrom z. B. an die Bandbreite eines Clients anzupassen. Damit eine hohe Qualität des transcodierten Videos erreicht werden kann, ist es notwendig, die verfügbare Bandbreite so weit als möglich auszunutzen. Daher sollte ein Transcoder eine Kontrolle der ausgehenden Bitrate enthalten, die es ermöglicht, die Transcoding-Parameter so anzupassen, dass die verfügbare Bandbreite optimal genutzt wird. Beim Frameskipping wird die Bitrate des transcodierten Videostroms durch die Anzahl der verworfenen Frames dynamisch kontrolliert. Bei der Skalierung und der Requantisierung muss die Datenrate durch die Quantisierung angepasst werden. Der Quantisierungsparameter muss also für jeden Makroblock so gewählt werden, dass einerseits der Quantisierungsfehler gering gehalten und andererseits die verfügbare Bandbreite nicht überschritten wird. In [11] wird dieses Problem als Aufgabe aus dem Bereich der Mathematischen Optimierung betrachtet, das mit Hilfe einer Lagrange-Optimierung gelöst werden kann. Ein anderer Ansatz [35] benutzt eine Laplace-Verteilung, um die Verteilung der *DCT*-Werte der Makroblöcke und somit die Anzahl der benutzten Codewörter zu approximieren und daraus geeignete Quantisierungsparameter abzuleiten. In [36] wurde ausgehend

von einem Modell für die Bitratenkontrolle, das verschiedene Wechselwirkungen zwischen Quantisierungsparametern und Datenrate modelliert, ein weiterer Algorithmus zur Bestimmung der Quantisierungsparameter angegeben.

Obgleich eine weitere Betrachtung dieser verschiedenen Möglichkeiten zur Kontrolle der Bitrate im Rahmen dieser Arbeit nicht möglich ist, soll zumindest darauf hingewiesen werden, dass bei allen Ansätzen der Quantisierungsparameter des Transcoders so gewählt werden muss, dass kritische Verhältnisse des Parameters zum ursprünglich codierten Parameter vermieden werden. Eine Untersuchung dieser kritischen Verhältnisse ist in [37] zu finden. Dort wird auch ein Schema vorgeschlagen, anhand dessen der Quantisierungsparameter so gewählt werden kann, dass einerseits keine kritischen Verhältnisse auftreten und andererseits die benötigte Bitzahl weiter reduziert wird.

5 Ein Multimedia-Gateway für mobile Clients

Für die Konzeption eines Transcoders, der in der Lage ist, audiovisuelle Medienströme, die über das Internet übertragen werden, umzuwandeln, gibt es zwei verschiedene Ansätze:

1. Der Transcoder wird so in den Medienserver integriert, dass er die Daten umwandeln kann, bevor diese an den Client übertragen werden.

2. Der Transcoder wird als Gateway realisiert, das Anfragen von Clients entgegennehmen, diese an einen Medienserver weiterleiten, die Daten vom Server empfangen und schließlich an den Client übertragen kann.

Beide Ansätze haben ihre Berechtigung, jedoch bietet der erste nur eine sehr geringe Flexibilität. Ein Client, der auf ein Transcoding des Medienstroms angewiesen ist, könnte dann nur von solchen Medienservern Daten empfangen, die ein Transcoding anbieten. Außerdem müsste der Transcoder an die jeweils verwendete Software des Medienservers angepasst werden. Andererseits würde durch die Integration des Transcoders in den Medienserver die Netzwerklast gesenkt werden, da die Daten nur mit einer geringen Datenrate übertragen werden würden. Darüber hinaus bietet diese Variante die Möglichkeit, dem Transcodingprozess zusätzliche Informationen über den Medienstrom bereitzustellen, da der Prozess neben Zugriff auf die Daten auch Zugriff auf Zusatzinformationen erhalten könnte. Bei der Realisierung des Transcoders als Gateway hingegen, stehen für den Transcodingvorgang jeweils nur die codierten Medienströme zur Verfügung. Allerdings bietet diese Architektur eine größtmögliche Flexibilität. So könnte das Gateway innerhalb des Providernetzwerkes des Clients platziert werden, so dass der Transcoding-Service für alle erreichbaren Medienserver zur Verfügung stünde. Die Daten würden beim zweiten Ansatz zwar mit einer höheren Datenrate vom Server an das Gateway übertragen, was eine höhere Netzwerklast verursachen würde, allerdings könnte dem mit der Integration einer Caching-Funktionalität im Gateway entgegengewirkt werden. Ein Preis für die höhere Flexibilität ist natürlich, dass sich mit dem Gateway ein "Single Point of Failure" ergibt. Aber auch diesem Problem kann begegnet werden, indem z. B. mehrere redundante Gateways im Providernetzwerk platziert werden. Der letzte Punkt kann andererseits

auch als Vorteil betrachtet werden, denn daraus ergibt sich ebenso, dass nur ein Transcoder an die Bedürfnisse der Clients angepasst werden müsste. Insgesamt bietet also die zweite Variante, den Transcoder als Gateway zu realisieren, die meisten Vorteile.

Im Folgenden wird nun das Design des entwickelten Multimedia-Gateways vorgestellt. Hierzu werden zunächst die Designziele, die bei der Entwicklung des Gateways verfolgt wurden, erläutert. Im darauf folgenden Abschnitt wird die als Grundlage für die Datenkommunikation gewählte, frei verfügbare RTSP/RTP-Implementierung *Komssys* vorgestellt. Der entwickelte Entwurf des Multimedia-Gateways wird daran anschließend im zweiten Teil dieses Kapitels vorgestellt und näher erläutert.

5.1 Designziele

Bei der Entwicklung des Multimedia-Gateways wurden primär die folgenden Ziele verfolgt, wobei durch die Reihenfolge der einzelnen Ziele bereits eine Priorisierung, beginnend mit der höchsten, angegeben ist.

Flexibilität: Das Gateway muss so modular aufgebaut sein, dass eine spätere Erweiterung der Funktionalität kein Problem darstellt. Für einer Erweiterung der Funktionalität sollte keine erneute Compilierung des gesamten Source-Codes notwendig sein.

Real-Time Transcoding: Die Umwandlung der Medienströme muss in Echtzeit erfolgen, so dass bei der Wiedergabe auf dem Client keine Verzögerungen auftreten. Der Entwurf muss dies berücksichtigen und darf den eigentlichen Transcodingprozess in keiner Form einschränken.

Integration: Um eine möglichst einfache Weiterentwicklung und Wiederverwendbarkeit zu gewährleisten, sollte der Transcoder so weit als möglich in die vorhandene *Komssys*-Umgebung eingebunden werden. Dies bietet auch den Vorteil, dass das Gateway gut für weitere Forschungsprojekte eingesetzt werden kann.

Transparenz: Aus Sicht des Clients sollte sich das Gateway wie ein regulärer Medienserver verhalten. Lediglich zur Konfiguration muss der Client von der Existenz des Gateways Kenntnis erhalten.

Konfiguration: Für die Benutzung des Gateways durch einzelne Clients muss dieser an die Bedürfnisse der Clients angepasst werden (z. B. Displaygröße oder bevorzugtes Format). Dies sollte für den Benutzer des Clients möglichst einfach sein.

Caching: Beim Entwurf sollte auch die Möglichkeit der Integration einer Cachingfunktionalität berücksichtigt werden, um Medienströme für eine erneute Anforderung durch einen Client abzuspeichern.

5.2 RTSP/RTP-Basisimplementation

Für die Datenkommunikation des Multimedia-Gateways wurde die frei verfügbare RTSP/RTP-Umgebung *KOM(S) Streaming System (Komssys)* [38] als Basisimplementation gewählt. Ausschlaggebend für diese Wahl waren mehrere Gründe: Da es sich bei *Komssys* um ein Open-Source-Projekt handelt, steht der Source-Code dieser Implementation zur Verfügung und kann für das Gateway angepasst werden. Außerdem wurde *Komssys* bereits mehrfach zu Forschungszwecken eingesetzt und umfasst nicht zuletzt dadurch eine große Funktionalität. Dies zeigt auch, dass die Erweiterungsmöglichkeiten dieser Software sehr gut sind. Ein weiterer Grund war auch die Zukunftsfähigkeit dieser Implementation. *Komssys* befindet sich weiterhin in der Entwicklung und wird ständig erweitert. Außerdem wurden für die Implementierung Techniken der objektorientierten Programmierung eingesetzt, was für eine zeitgemäße Implementation spricht. Andere verfügbare Implementationen, wie z. B. das RTSP Proxy Kit 2.0 [39], Darwin [40] oder LIVE.COM Streaming Media [41], bieten hingegen jeweils nur einen Teil der genannten Vorteile.

Für ein leichteres Verständnis des Entwurfs des Multimedia-Gateways wird in den folgenden Abschnitten die Implementation des *KOM(S) Streaming Systems* kurz vorgestellt. Insbesondere wird auf Eigenschaften und Besonderheiten hingewiesen, die für den Entwurf des Gateways relevant sind.

5.2.1 Komssys

Die Implementation von *Komssys* umfasst insgesamt drei Programmteile: einen Server, einen Proxy und einen Client. Beim Server handelt es sich um einen voll funktionsfähigen Medienserver, der zur Zeit die Medienformate MPEG-1 Video, MPEG-1 Systems, MP3, H.261 und Scalable MPEG[1] unterstützt. Der Proxy ist ein modifizierter Server, der in der Lage ist, Anfragen eines Clients an den entsprechenden Medienserver weiterzuleiten. Zusätzlich besitzt der Proxy Caching-Funktionalitäten, die es ihm erlauben, einen Medienstrom für eine spätere erneute Übertragung abzuspeichern. Sowohl der Server als auch der Proxy verhalten sich wie standardmäßige RTSP/RTP-Server und sollten daher mit jeder standardkonformen Clientsoftware zusammenarbeiten. Da jedoch viele verfügbare Clientprogramme an spezielle Serversoftware angepasst wurden, enthält *Komssys* auch einen Client. Zur Darstellung der Medienströme werden vom Client verschiedene projektfremde Softwareprodukte, wie z. B. die KDE2-Bibliothek `mpeglib`, das Programm `mplayer` oder das Gnome-Streaming-Subsystem `gstreamer`, eingesetzt.

[1] Bei Scalable MPEG handelt es sich um ein Layered Video-Format, das von Buck Krasic am Oregon Graduate Institute (http://www.ogi.edu) entwickelt worden ist.

Die Konfiguration des gesamten *Komssys*-Systems erfolgt über eine gemeinsame Konfigurationsdatei, die beim Start eingelesen wird. Während der Laufzeit stehen die Einträge dieser Datei sowie Standardwerte für fehlende Einträge dem jeweiligen Programm über eine einfache Schnittstelle zur Verfügung. Dieser Konfigurationsmechanismus ist sehr flexibel und kann leicht erweitert werden.

5.2.2 Stream-Handler

Alle Programmteile innerhalb von *Komssys* nutzen für die Verarbeitung der Medienströme *Stream-Handler (SH)*. Dabei handelt es sich um einzelne Programmmodule, die in der Lage sind, einen Datenstrom zu empfangen, zu bearbeiten und weiterzuleiten. Einzelne *SH* können von einer Kontrolleinheit zu einem azyklischen, gerichteten Datenpfad zusammengesetzt werden, durch den der Medienstrom fließen und in geeigneter Weise bearbeitet werden kann. Für den Datenaustausch besitzt jeder *SH* einen oder mehrere Ports, über die Daten empfangen oder gesendet werden können. Jeder Port wird von einem Endpunkt eingekapselt, der weitere Informationen über den Port bereitstellt. Anhand dieser Informationen kann z. B. entschieden werden, ob Ports verschiedener *SH* miteinander verbunden werden können. So gibt der Typ des Endpunkts u. a. darüber Auskunft, ob es sich um einen Port handelt, der Daten empfangen oder senden kann. Neben den eigentlichen Daten werden zwischen den Endpunkten auch sog. Reports ausgetauscht, die eine zusätzliche bidirektionale Kommunikation zwischen den *SH* erlauben. Kann ein empfangener Report innerhalb eines *SH* nicht interpretiert werden, so muss er an den nächsten *SH* weitergeleitet werden. Zur Verdeutlichung wurden die genannten Begriffe in der Abb. 5.1 zusammengefasst. Der exemplarisch dargestellte Datenpfad wird z. B. vom Server benutzt, um einen Medienstrom aus einer Datei an den Client zu übertragen.

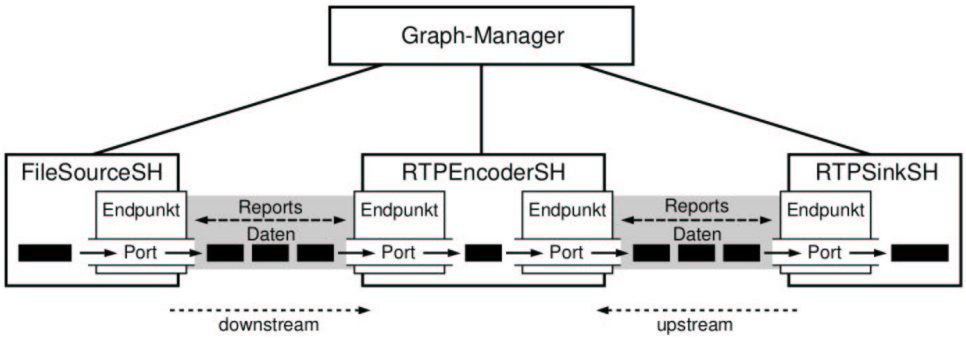


Abbildung 5.1: Schematischer Aufbau eines Datenpfades

Die *Stream-Handler* werden in *Komssys* anhand ihrer Timingkontrolle unterschieden und können in drei verschiedene Typen aufgeteilt werden [42]:

- **Active:** Aktive *SH* implementieren eine eigene Timingkontrolle. Arbeitet solch ein *SH* als Quelle, so sendet dieser aktiv Daten aus einer Datenquelle (z. B. einer Datei) in Stromrichtung (downstream) an den oder die nächsten *SH*. Das Senden der Daten wird dadurch realisiert, dass der aktive *SH* eine Push-Funktion des Endpunktes des verbundenen *SH* aufruft. Wird ein aktiver *SH* als Senke genutzt, so holt dieser aktiv bei vorherigen *SH* Daten ab, indem eine Pull-Funktion der Endpunkte der verbundenen *SH* aufgerufen wird. Die implementierte Timingkontrolle kann durch einen Timer, einen eigenen Thread oder die Überwachung externer Zustände, wie z. B. der Ankunft von Netzwerkpaketen oder Benutzereingaben, realisiert werden. Zwei aktive *SH* können nicht miteinander verbunden werden, es sei denn, ein passiver *SH* wird zwischengeschaltet.

- **Passive:** Ein passiver *SH* besitzt keine eigene Timingkontrolle, sondern reagiert nur auf ein aktives Senden oder Abholen von Daten eines verbundenen *SH*. Dient ein passiver *SH* als Quelle und Senke gleichzeitig, so muss er einen Puffer besitzen, der die Daten ggf. zwischenspeichern kann. Das Verbinden zweier passiver *SH* ist nicht möglich, da keine Daten zwischen ihnen ausgetauscht werden können.

- **Through:** Ein *SH* dieses Typs ist im Prinzip eine Kombination der ersten beiden Typen und muss immer mindestens je einen Quell- und einen Senken-Endpunkt besitzen. Werden Daten an einen through[2] *SH* gesendet bzw. übergeben, werden sie nach einer Bearbeitung an den nächsten *SH* in Stromrichtung (downstream) gesendet. Werden die Daten durch einen aktiven *SH* von einem through *SH* abgeholt, so muss letzterer die Daten von einem vorherigen verbundenen *SH* gegen die Stromrichtung (upstream) aktiv abholen. Gegenüber einem aktiven *SH* verhält sich ein through *SH* also wie ein passiver und gegenüber einem passiven wie ein aktiver *SH*. Ein Verbinden von *SH* diesen Typs ist also mit jedem beliebigen anderen *SH* möglich.

Aufgrund dieser Typisierung der *Stream-Handler* muss ein Datenpfad also immer mindestens einen aktiven und einen passiven *SH* enthalten. In dem in Abb. 5.1 dargestellten Datenpfad sind zwei verschiedene *SH*-Typen zu finden: Beim `FileSourceSH` und `RTPSinkSH` handelt es sich jeweils um einen passiven *SH*, der `RTPEncoderSH` ist ein aktiver *SH*.

[2] Auf eine Übersetzung dieses Begriffes wurde zu Gunsten der Lesbarkeit verzichtet.

5.2.3 Graph-Manager

Die Kontrolleinheit, die die *Stream-Handler* zu einem Datenpfad verbindet, wird bei *Komssys* als *Graph-Manager (GM)* bezeichnet. Dieser ist dafür verantwortlich, bei einer Anforderung eines Clients den Datenpfad aufzubauen, durch den der Medienstrom fließen kann. Hierzu werden geeignete *SH* miteinander verbunden und entsprechend konfiguriert. In einigen Situationen ist es notwendig, den Datenpfad während der Datenübertragung zu verändern. Ein Proxy mit Caching-Funktionalität kann z. B. so konfiguriert werden, dass dieser, wenn ein Client eine Übertragung unterbricht oder pausiert, den Medienstrom weiter vom Server empfängt und abspeichert. Hierzu muss derjenige Teil des Datenpfades, der die Daten an den Client sendet entfernt und derjenige Teil, der die Daten speichert, erhalten werden. Setzt der Client anschließend die Übertragung fort, muss ein neuer Subpfad angelegt werden, der die Daten nun aus der gespeicherten Datei an den Client überträgt.

5.2.4 Datentransport

Zwischen den einzelnen *Stream-Handlern* werden die Daten in logischen Einheiten, die als Pakete bezeichnet werden, ausgetauscht. Transportiert werden die untypisierten Daten mit Hilfe von Ropes [43]. Bei Ropes handelt es sich um eine Abstraktion von normalen Puffern, die ein Kopieren der Daten weitestgehend überflüssig macht. Innerhalb eines Ropes werden verschiedene Referenzen und Puffer der Daten verwaltet. Werden z. B. Daten an ein Rope angefügt, so wird lediglich eine Referenz auf die Daten hinzugefügt, ohne die Daten selbst zu kopieren. Intern liegen die Daten also in mehreren Puffern vor, obgleich es sich aus Sicht des Programms, das ein Rope benutzt, um einen einzelnen Puffer handelt, der die Daten enthält.

5.2.5 Konfiguration

Da die Konfiguration von *Komssys* für den Entwurf des Gateways erweitert wurde, wird sie an dieser Stelle kurz erläutert. Es wird jedoch nicht auf die Funktionen einzelner Einstellungen, sondern lediglich auf die Syntax und Struktur der Konfigurationsdatei eingegangen. Eine detaillierte Beschreibung aller Einstellungen ist auf den Internetseiten des *Komssys*-Projekts zu finden [38]. Ein einfaches Beispiel für eine Konfigurationsdatei befindet sich im Anhang A dieser Arbeit.

Die Konfiguration von *Komssys* erfolgt mit Hilfe einer Datei, die eine XML-ähnliche Syntax benutzt. Die einzelnen Einstellungen sind innerhalb dieser Datei in Blöcke aufgeteilt, die jeweils von einem Start- und einem End-*Tag* eingeschlossen sind. Für jedes System (Client, Server oder Proxy) existiert innerhalb

dieser Datei ein eigener Block, der mit `<system>` eingeleitet wird. Für welches System dieser Block zuständig ist, wird durch einen `type`-Parameter angegeben. Der Block für den Proxy z. B. wird also durch `<system type="proxy">` eingeleitet. Durch die Angabe eines Hostnamens mit Hilfe des `name`-Parameters ist es auch möglich, spezialisierte Abschnitte für bestimmte Hosts anzulegen. Sind für ein System mehrere Blöcke in der Datei enthalten, so werden die Einstellungen aller Blöcke für dieses System benutzt. Treten Einstellungen innerhalb eines Blocks oder innerhalb verschiedener Blöcke für das gleiche System mehrfach auf, ist die Reihenfolge des Einlesens und somit auch der Wert dieser Einstellung undefiniert. Innerhalb der Systemblöcke sind die Einstellungen wiederum in Blöcke unterteilt. So gibt es z. B. einen Block für *RTSP*-Einstellungen, der durch das *Tag* `<rtsp>` eingeleitet wird. Innerhalb dieses Blocks sind z. B. die zu benutzenden Netzwerkports für den Server und den Proxy zu finden.

5.2.6 Der Proxyserver

Als Grundlage für den vorliegenden Entwurf eines Multimedia-Gateways bietet sich der Proxyserver des *Komssys*-Projekts an. Dieser enthält bereits die Funktionalität, eine Anfrage eines Clients entgegenzunehmen, den Datenstrom beim entsprechenden Medienserver anzufordern und diesen dann an den Client weiterzuleiten. Diese Funktionalität wird auch von einem Gateway benötigt.

Da der Proxy für den Entwurf des Gateways also eine zentrale Rolle spielt, wird dieser an dieser Stelle genauer betrachtet und vorgestellt. Die Beschreibung beschränkt sich jedoch auf einige wichtige Komponenten, die an der Kommunikation beteiligt sind. Eine genauere Beschreibung des Proxys ist in [44] zu finden. Die Abbildung 5.2 gibt einen Überblick über die beteiligten Komponenten sowie den Datenfluss innerhalb des Proxys. Die benutzten Bezeichnungen einzelner Komponenten sowohl im Text als auch in der Abbildung entsprechen jeweils den Klassen, die im Proxy des *Komssys*-Projekts verwendet wurden.

Für jede *RTSP*-Session wird innerhalb des Proxys jeweils ein Objekt der Klasse `RTSPProxySession` erzeugt, das die weitere Kommunikation mit dem Client und mit dem Server steuert. Für diese Kommunikation besitzt die `RTSPProxySession` einen `ProxyClient`, der eine *RTSP*-Session zum Medienserver aufbaut und einen `ProxyServer`, der die Kommunikation zum Client übernimmt. Bei diesen beiden letzten Komponenten handelt es sich jeweils um Zustandsautomaten, die sehr eng miteinander gekoppelt sind. Eine Zustandsänderung des einen verursacht in den meisten Fällen auch eine Änderung im anderen Automaten.

Wenn die beiden *RTSP*-Sessions zwischen Client und Proxy sowie zwischen Proxy und Server initialisiert sind, kann die eigentliche Datenkommunikation über *RTP* erfolgen. Hierzu wird vom `ProxyClient` der `MNStreamer` aufgerufen, der wiederum einen geeigneten *Graph-Manager* und somit den nötigen Datenpfad erzeugt.

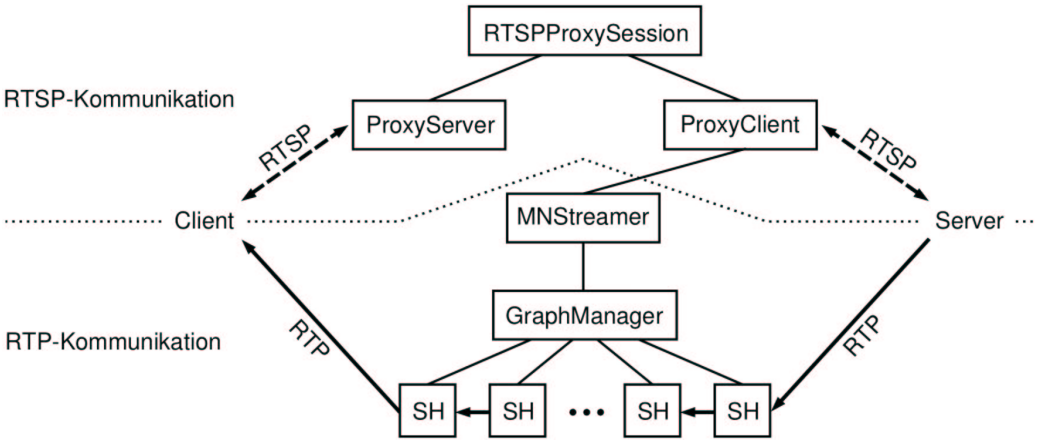


Abbildung 5.2: Komponenten zur Kommunikation des Proxyservers

Ist im Proxy kein Caching konfiguriert, so wird der Medienstrom durch einen Datenpfad geleitet, der lediglich aus den *Stream-Handlern* `RTPSourceSH` und `RTPSinkSH` besteht. `RTPSourceSH` nimmt hierbei die *RTP*-Pakete aktiv aus dem Netzwerk entgegen und leitet diese an `RTPSinkSH` weiter, der diese über das Netzwerk an den Client sendet. Bei konfiguriertem Caching wird zwischen diesen beiden *SH* ein `MultiplierSH` eingesetzt, der den Datenstrom dupliziert und mit Hilfe eines `RTPDecoderSH` und eines `FileSinkSH` in eine Datei speichert.

5.3 Entwurf

Wie bereits erwähnt, bietet sich der Proxyserver von *Komssys* als Basis für das Multimedia-Gateway an. Dieser muss so modifiziert werden, dass eine Bearbeitung des Medienstroms vor dem Senden der Daten an den Client möglich ist. Hierzu ist es notwendig, den Teil des Proxys, der für die *RTP*-Kommunikation zuständig ist, anzupassen. Es müssen also der *Graph-Manager* und somit auch der Datenpfad angepasst bzw. erweitert werden. Aufgrund der Tatsache, dass es sehr verschiedene Verfahren für den Transcodingvorgang gibt (vgl. Kap. 4), ist es jedoch nahezu unmöglich, einen monolithischen *Graph-Manager* zu entwerfen, der verschiedene Verfahren benutzen und verschiedene Medienformate behandeln kann. Stattdessen kommt ein modularer Aufbau zum Einsatz, der es dem *GM* erlaubt, einen Teil des Datenpfades zur Laufzeit des Gateways aus einer Bibliothek zu laden.

5.3.1 Ein Transcoding-Datenpfad

Zur Umwandlung des Medienstroms ist es zunächst notwendig, die *RTP*-Header der einzelnen Pakete zu entfernen. Hierzu kann der in *Komssys* enthaltene *Stream-Handler* `RTPDecoderSH` genutzt werden. Daran anschließend können verschiedene weitere *Stream-Handler* das eigentliche Transcoding durchführen. Die umgewandelten Daten müssen daraufhin wieder in *RTP*-Pakete aufgeteilt und mit entsprechenden Headern versehen werden, bevor sie an den Client übertragen werden können. Das bedeutet, dass sich der Transcoding-Datenpfad in drei Teile aufspalten lässt: Der erste Teil empfängt die *RTP*-Pakete aus dem Netzwerk, entfernt die Header und übergibt sie an den nächsten Teil, der die Daten umwandelt. Nach dieser Umwandlung werden die Daten an den dritten Teil übergeben, der neue *RTP*-Pakete erzeugt und über das Netzwerk an den Client überträgt. Somit ergibt sich, dass der erste und der letzte Teil des Datenpfades bei jeder Übertragung und jedem Transcodingverfahren identisch sind. Nur der mittlere Teil ändert sich je nach Medienformat bzw. einzusetzendem Verfahren. Es liegt also nahe, genau diesen Teil des Datenpfades, der im Folgenden als Transcoding-Subpfad bezeichnet wird, dynamisch zur Laufzeit des Gateways zu laden. Diese Architektur ist schematisch in Abb. 5.3 dargestellt. Der gestrichelt umrandete Teil des dargestellten Datenpfades entspricht hierbei dem Transcoding-Subpfad, der dynamisch zur Laufzeit geladen wird. Der Typ der äußeren *SH* ist jeweils durch den Buchstaben in der rechten unteren Ecke der *Stream-Handler* angegeben.

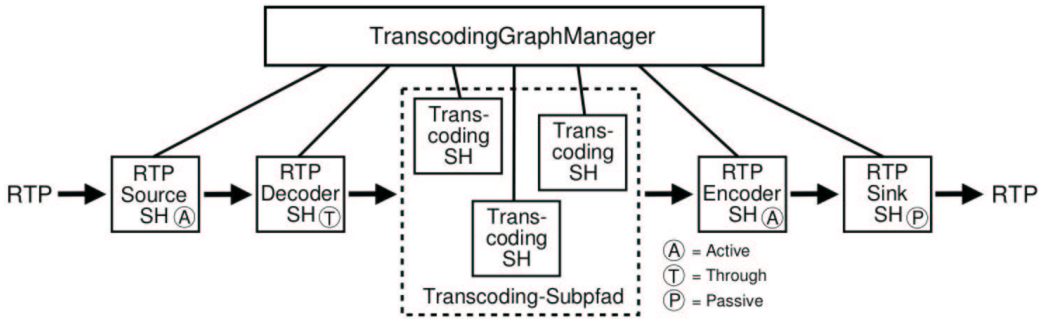


Abbildung 5.3: Allgemeiner Transcoding-Datenpfad

Dieser bis jetzt beschriebene Datenpfad führt jedoch aufgrund der Typen der beteiligten äußeren *SH* zu gewissen Einschränkungen für den Transcoding-Subpfad. Da es sich beim `RTPSourceSH` um einen aktiven und beim `RTPDecoderSH` um einen through *SH* handelt, muss der Typ des damit verbundenen *SH* des Transcoding-Subpfades passiv oder through sein. Ebenso muss der Typ des letzten *SH* des Subpfades passiv oder through sein. Diese Einschränkung kann jedoch leicht aufgehoben werden, indem vor und nach dem Transcoding-Subpfad ein passiver `PushPullSH` eingesetzt wird, der als Puffer zwischen den einzelnen Teilen des Datenpfades dient. Dieser *SH* ist bereits in *Komssys* enthalten und wird dort in verschiedenen Datenpfaden eingesetzt.

Es stellt sich jedoch die Frage, woher der *Graph-Manager* die Informationen erhält, welche *Stream-Handler* geladen und in welcher Reihenfolge diese für das Transcoding miteinander verbunden werden müssen. Um eine möglichst große Flexibilität des Gateways bzgl. zukünftiger Erweiterungen zu erhalten, könnte eine Beschreibungssprache entwickelt werden, die es dem *Graph-Manager* erlaubt, die notwendigen Informationen aus einer Konfigurationsdatei zu laden. Dies stellt jedoch einen sehr großen Aufwand un eine zusätzliche Fehlerquelle dar. Dieses kann vermieden werden, indem eine Schnittstelle geschaffen wird, die es ermöglicht, den Transcoding-Subpfad aus der Sicht des *Graph-Managers* durch ein einzelnes Objekt zu realisieren. Dieses Objekt könnte dynamisch zur Laufzeit des Gateways aus einer Bibliothek geladen werden. In diesem Fall müsste der *Graph-Manager* neben der zu ladenden Bibliothek nur noch wenige Parameter aus einer Konfigurationsdatei lesen, um den Datenpfad aufbauen und konfigurieren zu können.

Zur Realisierung dieses letzteren Ansatzes wurde im Rahmen dieser Arbeit die abstrakte Klasse `SubGraphSH` entworfen, die als Basisklasse für Klassen dient, die einen Transcoding-Subpfad implementieren. Als *Graph-Manager* wurde die Klasse `TranscodingGM` entworfen, die abhängig vom Client und vom Medienformat mit Hilfe der Klasse `SubGraphFactory` ein Objekt einer von `SubGraphSH` abgeleiteten Klasse aus einer Bibliothek laden kann. Diese drei Klassen werden in den folgenden Abschnitten vorgestellt und näher erläutert. Aus Gründen einer leichteren Lesbarkeit wird im Folgenden meist auf die Unterscheidung zwischen Objekt bzw. Instanz einer Klasse und der Klasse selbst verzichtet. Ob es sich beim Gebrauch eines Klassennamens um eine Instanz oder um die Klasse selbst handelt, sollte jedoch aus dem Zusammenhang erkennbar sein.

5.3.2 Ein Transcoding-Objekt

Die Klasse `SubGraphSH` wurde als Schnittstelle entwickelt, die es erlaubt, einen Transcoding-Subpfad, wie im letzten Abschnitt beschrieben, als einzelnes Objekt zu laden. Es handelt sich hierbei um eine abstrakte Klasse, die sowohl Eigenschaften eines *Stream-Handlers* als auch Funktionalitäten eines *Graph-Managers* besitzt. Gegenüber dem `TranscodingGM` und den verbundenen *SH* verhält sich ein Objekt dieser Klasse wie ein aktiver *SH* und besitzt je einen Quell- und Senken-Endpunkt. Intern dient der `SubGraphSH` jedoch als *Graph-Manager*, der verschiedene *SH* zu einem Datenpfad verbinden kann, innerhalb dessen der Medienstrom transcodiert werden kann. Dieses Verhalten wird dadurch realisiert, dass diese Klasse zwei Basisklassen besitzt[3]. Einerseits wurde sie von `SHGraphManager` und andererseits von `SH::Base` abgeleitet. Bei `SHGraphManager` handelt es sich um die Basisklasse aller *Graph-Manager* und `SH::Base` ist die Basisklasse aller *Stream-Handler* von *Komssys*.

[3]Es wird an dieser Stelle das Konzept der Mehrfachvererbung genutzt, da die bei der Implementation des Gateways eingesetzte Programmiersprache (C++) dieses unterstützt.

Damit der Datenstrom durch die internen *Stream-Handler* des `SubGraphSH` hindurch fließen kann, müssen diese mit den externen *SH* verbunden werden. Um ein direktes Verbinden durch den `TranscodingGM` zu ermöglichen, muss dieser Zugriff auf die Endpunkte der äußeren internen *SH* bekommen. Hierfür bietet die Klasse `SubGraphSH` zwei Methoden, die es einer abgeleiteten Klasse erlauben, den Senken- bzw. Quell-Endpunkt eines internen *SH* als den eigenen zu übernehmen. Durch den Aufruf dieser Methoden im Konstruktor einer von `SubGraphSH` abgeleiteten Klasse ist der `TranscodingGM` in der Lage, eine Instanz dieser Klasse mit den anderen Teilen des Datenpfades zu verbinden.

Eine gewisse Verzögerung bei der Datenübertragung ist bei einem Transcoding des Medienstroms nicht vermeidbar. Daher muss die Klasse `SubGraphSH` zusätzlich eine Methode `getDelay()` bereitstellen, die diese Verzögerung in Mikrosekunden liefert. Diese Methode kann dann vom `TranscodingGM` dazu genutzt werden, die Datenübertragung zu verzögern, indem der Rückgabewert dieser Methode als Verzögerung des `RTPEncoderSH` gesetzt wird.

5.3.3 Ein Transcoding-Graph-Manager

Der *Graph-Manager* ist in *Komssys* dafür zuständig, den Datenpfad, durch den der Medienstrom geleitet wird, aufzubauen. Für das Gateway wurde daher die Klasse `TranscodingGM` entworfen, die vom `MNStreamer` (siehe Abschnitt 5.2.6) instanziiert wird, nachdem zwischen Client und Gateway einerseits sowie zwischen Gateway und Medienserver andererseits eine *RTSP*-Session aufgebaut wurde. Bei der Instanziierung dieser Klasse werden zunächst die *Stream-Handler* der äußeren bei jeder Übertragung identischen Teile des Datenpfades geladen. Nachdem dem `TranscodingGM` die zu benutzenden Netzwerkports, die Namen des Servers und des Clients sowie das Format des Medienstroms mitgeteilt wurden, kann dieser den Transcoding-Subpfad mit Hilfe der Klasse `SubGraphFactory` aus einer Bibliothek laden. Das so geladene Objekt verhält sich für den `TranscodingGM` wie ein normaler aktiver *Stream-Handler* und kann daher mit den bereits bestehenden Teilen des Datenpfades verbunden werden. Wird nun die *RTP*-Übertragung gestartet, fließt der Medienstrom durch den kompletten Datenpfad und wird im Transcoding-Subpfad umgewandelt.

Der Datenpfad setzt sich aus drei aktiven Teilen zusammen, da sie jeweils einen aktiven *SH* enthalten. Werden alle Teile innerhalb eines einzelnen Threads realisiert, kann es zu einem Deadlock kommen, bei dem der gesamte Datenpfad blockiert ist. Bei dem hier vorgestellten Datenpfad tritt solch ein Fall mit hoher Wahrscheinlichkeit bereits beim Start der Medienübertragung auf. Der aktive `RTPEncoderSH` versucht beim vorherigen `PushPullSH` Daten abzuholen. Da dieser jedoch zu Beginn noch keine Daten enthält, wird der Aufruf der Pull-Methode seines Endpunkts und damit der gesamte Prozess blockiert. Eine vergleichbare Situation kann auch zwischen dem Transcoding-Subpfad und dem ersten `PushPullSH`

auftreten. Dadurch ergibt sich eine Blockierung des gesamten Datenpfades, die eine weitere Datenübertragung unmöglich macht. Diese Situation kann vermieden werden, indem jeder Subpfad durch den `TranscodingGM` innerhalb eines eigenen Threads gestartet wird. Dadurch führt die Blockierung eines Subpfades nicht zu einer Blockierung des vorherigen Subpfades, so dass dieser weiterhin Daten übertragen kann.

5.3.4 Dynamisches Laden eines Subpfades

Für die Funktionalität des dynamischen Ladens eines Transcoding-Subpfades aus einer Bibliothek wurde die Klasse `SubGraphFactory` entworfen. Diese Klasse ist dafür zuständig, anhand eines übergebenen Clientnamens sowie eines Medienformats ein Objekt einer von `SubGraphSH` abgeleiteten Klasse zu laden und an den aufrufenden *Graph-Manager* zurückzuliefern. Der Namen der Bibliothek wird hierbei aus der Konfigurationsdatei ermittelt. Ein Transcoding-Subpfad muss jedoch nicht notwendigerweise aus einer Bibliothek geladen werden. Stattdessen kann das Gateway bereits Subpfade für einige Formate und Techniken enthalten, die dann direkt zurückgeliefert werden können. Für den `TranscodingGM`, der die `SubGraphFactory` benutzt bleibt das Verhalten jedoch transparent, so dass dieser nicht erkennen kann, ob der Subpfad aus einer Bibliothek geladen oder direkt erzeugt wurde.

5.3.5 Medienformate und Transcodingtechniken

Wie anhand der letzten Abschnitte bereits zu erkennen ist, sind im Entwurf des Gateways die Medienformate und die einzusetzenden Transcodingtechniken sehr eng miteinander gekoppelt. Sowohl Medienformat als auch Transcodingverfahren sind in einem einzelnen Objekt gekapselt. Das Gateway kann also nicht jedes implementierte Transcodingverfahren für jedes implementierte Medienformat einsetzen. Stattdessen muss für jede Kombination ein eigenes Transcoding-Objekt implementiert werden.

Der Grund hierfür liegt in der Komplexität der verschiedenen Videocodierungen. Für die Möglichkeit einer Trennung zwischen Format und Transcodingtechnik, so dass diese unabhängig voneinander geladen werden könnten, wäre es notwendig, ein Zwischenformat zu entwickeln, welches als Eingabeformat für die verschiedenen Transcodingverfahren dienen könnte. Die Entwicklung eines derartigen Zwischenformats, das die Vorteile existierender und zukünftiger Videocodierungen vereinen müsste, ist jedoch – abgesehen von der vollständigen Decodierung der Daten – kaum vorstellbar. Doch selbst wenn ein solches Format existieren würde, wäre die Umwandlung des eigentlichen Medienformats in dieses Zwischenformat mit sehr hoher Wahrscheinlichkeit so aufwendig, dass ein Transcoding nicht in Realzeit möglich wäre.

In Kapitel 4 wurde bereits dargestellt, dass das Beispiel der vollständigen Decodierung der Videodaten für ein Transcoding in Echtzeit wenig sinnvoll ist. Einerseits, da hierdurch ein erheblicher zusätzlicher Rechenaufwand entstehen würde – die Umwandlung müsste vor und nach dem Transcoding erfolgen. Andererseits, da hierdurch Detailinformationen der codierten Daten, wie z. B. die Bewegungsvektoren, verloren gingen.

Die enge Kopplung des Medienformats und des Transcodingverfahrens innerhalb eines Objekts, wie es in Abschnitt 5.3.2 vorgestellt wurde, stellt daher einen guten Kompromiss für dieses Problem dar. Einerseits ist es möglich, den Transcodingvorgang für ein bestimmtes Medienformat zu optimieren. Andererseits besteht auch die Möglichkeit einen Transcoding-Subpfad zu implementieren, der mehrere verschiedene Formate verarbeiten kann. Die Einschränkung, für die verschiedenen zu nutzenden Kombinationen jeweils eigene Objekte zu implementieren, kann durch die Fähigkeit des Gateways, diese dynamisch zur Laufzeit zu laden, sicherlich vernachlässigt werden.

5.3.6 Konfiguration

Da es sich bei dem hier entwickelten Gateway um einen modifizierten Proxy handelt, war es nicht notwendig, für das Gateway in der Konfigurationsdatei einen eigenen System-Block zu erzeugen. Stattdessen wurde der Block des Proxys um einige Einträge erweitert. Es wurden ein neues Start-Tag `<transcoding>` und ein zugehöriges End-Tag `</transcoding>` hinzugefügt, zwischen denen die Einstellungen für den Transcodingprozess angegeben werden. Ein Block, der sämtliche Einstellungen enthält, könnte beispielsweise wie folgt aussehen:

```
<transcoding>
    <use-transcoding value="yes"/>
    <screenwidth value="240"/>
    <screenheight value="320"/>
    <libraries>
        <transcoding-lib pt="MPV" lib="./libmpeg.so"/>
        <transcoding-lib pt="H261" lib="./libh261.so"/>
    </libraries>
    <client name="faust">
        <screenwidth value="640"/>
        <screenheight value="200"/>
        <libraries>
            <transcoding-lib pt="MPV" lib="./libmpeg2h263.so"/>
        </libraries>
    </client>
```

```
        <client name="gretchen">
            <use-transcoding value="no"/>
        </client>
    </transcoding>
```

Die erste Angabe nach dem Start-*Tag* (`<use-transcoding value="yes"/>`) gibt an, ob überhaupt Transcoding genutzt werden soll. Wird hier der Wert `no` angegeben, so wird der eigentliche Proxy des *Komssys*-Projekts gestartet. Daran anschließend folgen einige Standard-Einstellungen, die genutzt werden, wenn für einen Client keine speziellen Einstellungen vorhanden sind:

`<screenwidth value="240"/>` und `<screenheight value="320"/>` geben die Displaygröße an, die bei einer Verringerung der Auflösung genutzt werden soll.

`<libraries>` leitet einen Block ein, in dem die zu benutzenden Bibliotheken mit Hilfe des *Tags* `<transcoding-lib>` angegeben werden können.

`<transcoding-lib pt="MPV" lib="./libmpeg.so"/>` gibt an, dass für das Medienformat mit der internen Bezeichnung "MPV" die Bibliothek "libmpeg.so" im aktuellen Verzeichnis des Proxys genutzt werden soll.

`<transcoding-lib pt="H261" lib="./libh261.so"/>` gibt in gleicher Weise eine Bibliothek für das Format "H261" an. Statt der dargestellten verkürzten Schreibweise können in diesem Tag statt `pt` und `lib` auch `payloadType` bzw. `library` benutzt werden.

Nach diesen Standardwerten können mehrere Blöcke mit Einstellungen für spezielle Clients folgen. Im obigen Beispiel ist für einen Client mit dem Namen "faust" z. B. eine spezifische Displaygröße und eine andere Bibliothek für das Format "MPV" angegeben. Die restlichen Einstellungen werden aus den Standard-Einstellungen übernommen. Für den Client mit dem Namen "gretchen" hingegen, wurde das Transcoding vollständig deaktiviert.

Zur Einstellung einer Bibliothek für ein bestimmtes Medienformat ist noch anzumerken, dass für die Bezeichnung der Formate die bereits innerhalb von *Komssys* genutzten Bezeichnungen verwendet werden. Ein Großteil dieser Bezeichnungen stammt aus dem RFC 1890 [45], eine komplette Auflistung ist im Anhang B zu finden.

5.3.7 Klassenhierarchie der entworfenen Klassen

Die Hierarchie der entworfenen Klassen ist in der Abbildung 5.4 dargestellt. Zur Verdeutlichung wurden die neu entworfenen Klassen grau unterlegt. Der dargestellte Graph zeigt einerseits die Vererbungshierarchie und andererseits die Aggregation zwischen diesen Klassen.

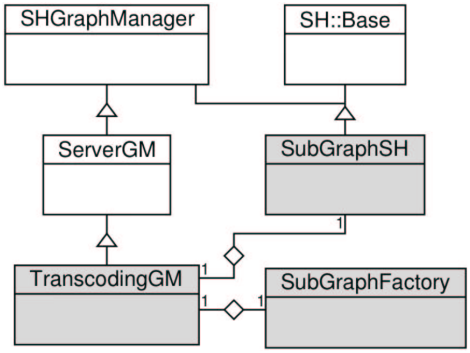


Abbildung 5.4: Vererbungs- und Aggregationsgraph der neu entworfenen Klassen

Auf eine Darstellung der weiteren Klassen, die vom `TranscodingGM` genutzt werden, wurde in dieser Abbildung zu Gunsten einer besseren Übersichtlichkeit bewusst verzichtet. Stattdessen ist eine Darstellung des `TranscodingGM` mit einem kompletten Datenpfad in der Abbildung 5.5 zu finden. Da gegenüber dem Proxy nur die *RTP*-Kommunikation angepasst wurde, ist in dieser Abbildung nur der *RTP*-Datenfluss dargestellt. Der Datenfluss der *RTSP*-Kommunikation ist identisch zu dem innerhalb des Proxys (siehe auch Abb. 5.2).

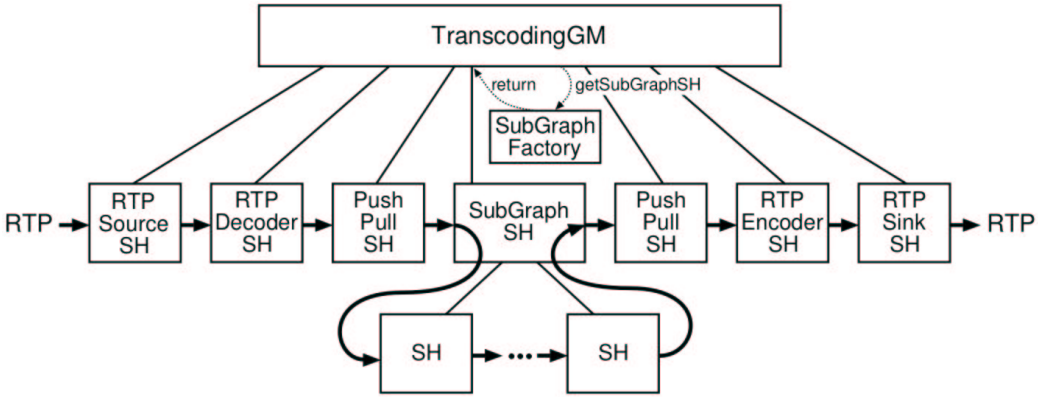


Abbildung 5.5: Datenfluss im Transcoding-Datenpfad

Eine genauere Betrachtung der einzelnen Klassen ist im nachfolgenden Kapitel zu finden. Dort wird insbesondere der Vorgang des dynamischen Ladens näher erläutert.

6 Implementation

Der im Kapitel 5.3 vorgestellte Entwurf des Multimedia-Gateways für mobile Clients wurde im Rahmen dieser Arbeit auch implementiert. Diese Implementation soll nun in den folgenden Abschnitten anhand der erstellten Klassen dokumentiert und vorgestellt werden. Eine übersichtliche Auflistung aller implementierten und bearbeiteten Klassen ist zusätzlich im Anhang C zu finden. Der Source-Code der Implementation ist auf der beigefügten CD enthalten, über dessen Verzeichnisstruktur der Anhang D Aufschluss gibt.

Für die Implementation des Gateways wurde aus Performancegründen und aufgrund der nötigen Integration in die benutzte Basisimplementation *Komssys* die Programmiersprache C++ genutzt. Soweit es möglich war wurden ausschließlich objektorientierte Techniken bei der Programmierung eingesetzt.

6.1 Konfiguration

Für das Einlesen der Konfigurationsdatei wird in *Komssys* der XML-Parser `expat` [46] genutzt. Dieser Parser ist jedoch nur für die XML-Syntax zuständig. Für die Inhalte der *Tags* existieren Klassen, die analog zu den XML-*Tags* eine hierarchische Struktur bilden. Für jeden Block innerhalb der Konfigurationsdatei existiert eine Klasse, die für alle *Tags* innerhalb dieses Blocks ein eigenes Objekt enthält, das das jeweilige *Tag* interpretieren kann. In *Komssys* setzen sich die Namen dieser Klassen aus dem Präfix `EConf` und dem Namen des Blocks zusammen, für den diese Klasse zuständig ist. So ist z. B. die Klasse `EConfSystem` für die System-Blöcke innerhalb der Konfigurationsdatei zuständig. Zusätzlich existieren weitere Klassen, die für *Tags* eines bestimmten Typs zuständig sind. Die Klasse `EConfBool` z. B. dient dazu, solche *Tags* einzulesen, die nur die Werte wahr oder falsch annehmen können. Das in Abschnitt 5.3.6 eingeführte *Tag* `<use-transcoding>`, das innerhalb eines Konfigurationsblocks, der für das Transcoding zuständig ist, verwendet wird, ist ein Beispiel für ein derartiges *Tag*.

Insgesamt wurden für das Einlesen der Transcoding-Konfiguration vier neue Klassen implementiert, die in Abb. 6.1 grau unterlegt dargestellt sind. Die Klasse `EConfTranscoding` ist dafür zuständig, den durch `<transcoding>` eingeleiteten

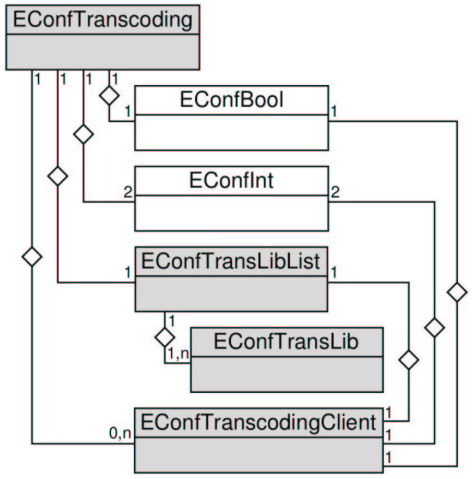


Abbildung 6.1: Aggregationsgraph der implementierten Klassen für die Konfigurationsdatei

Block einzulesen und die Werte in entsprechenden Variablen abzulegen. Für das Einlesen des *Tags* `<use-transcoding>` wird die Klasse `EConfBool` genutzt. Die Einstellungen für die Größe des Displays der Zielsysteme werden, da es sich um ganzzahlige Werte handelt, von der Klasse `EConfInt` eingelesen. Für das Einlesen einer Liste von Bibliotheken, die für das Transcoding genutzt werden sollen ist die Klasse `EConfTransLibList` zuständig. Diese nutzt die Klasse `EConfTransLib` für das Einlesen der einzelnen Bibliotheken. Nach diesen Standardeinstellungen können beliebig viele Blöcke für spezielle Clients folgen, deren Struktur jeweils der Struktur der Standardeinstellungen entspricht. Daher wird für das Einlesen jedes Client-Blocks eine Klasse `EConfTranscodingClient` genutzt die wiederum die gleichen Klassen benutzt, wie `EConfTranscoding`. Die Aggregations-Relation dieser genannten Klassen ist in Abb. 6.1 dargestellt. In dieser Abbildung gilt, wie bereits im vorherigen Kapitel, dass es sich bei den grau unterlegten Klassen um neu implementierte und bei den nicht unterlegten um vorhandene Klassen des *Komssys*-Projekts handelt.

Für den Zugriff auf die Einstellungen bietet `EConfTranscoding` vier Methoden, die anhand eines übergebenen Clientnamens die entsprechende Einstellung zurückliefern:

`bool getUseTranscoding(MNString clientName)` Diese Methode liefert in Abhängigkeit der Einstellung für den angegebenen Clientnamen `true` oder `false` zurück und gibt somit an, ob Transcoding für diesen Client genutzt werden soll.

`int getScreenWidth(MNString clientName)`,
`int getScreenHeight(MNString clientName)` Diese Methoden liefern die für den angegebenen Client konfigurierte Auflösung des Displays zurück, die für das Transcoding genutzt werden soll.

`MNString getTransLib(MNString clientName, PayloadType payloadType)`
: Diese Methode liefert den Namen der Bibliothek zurück, die für den angegebenen Client und das angegebene Medienformat in der Konfigurationsdatei enthalten ist. Ist für diesen Client bzw. für dieses Format keine Bibliothek konfiguriert, so wird ein leerer String zurückgegeben.

Intern nutzen alle vier Methoden die Methode `getTranscodingClient`, die versucht, die Einstellungen für den angegebenen Clientnamen zu finden. Der übergebene Name kann hierbei auch eine IP-Adresse oder ein Aliasname des Clients sein. Handelt es sich um eine IP-Adresse und wird diese in den Einstellungen nicht gefunden, so wird sie in einen oder mehrere Hostnamen aufgelöst, nach denen anschließend in den Einstellungen gesucht wird. Das bedeutet, dass innerhalb der Konfigurationsdatei nicht notwendigerweise der vollständige Name eines Clients angegeben sein muss, sondern stattdessen auch die IP-Adresse oder ein Aliasname (z. B. der Rechnername ohne Domain) des Clients in dieser Datei eingetragen werden kann.

6.2 SubGraphSH

Die Klasse `SH::SubGraphSH` dient als Basisklasse für die dynamisch zu ladenden Subpfade des `TranscodingGM`. Es handelt sich um eine abstrakte Klasse, in der aus diesem Grund nur wenige Methoden implementiert wurden. Wie bereits im Entwurf beschrieben wurde, wurde die Klasse von `SH::Base`, der Basisklasse aller *Stream-Handler* und `SHGraphManager`, der Basisklasse aller *Graph-Manager* abgeleitet. Von den ererbten Methoden wurde nur `pushReport` implementiert, da diese Methode für diese Klasse normalerweise nicht benötigt wird. Sie enthält in ihrem Rumpf daher nur eine Warnmeldung, die darauf hinweist, dass sie normalerweise nicht aufgerufen werden sollte. In einem normalen *SH* dient diese Methode zum Austausch von Reports zwischen den *SH* untereinander. Da der restliche Datenpfad jedoch mit den internen *Stream-Handlern* direkt verbunden ist, wird diese Methode im `SH::SubGraphSH` nicht benötigt. Sie wurde jedoch nicht als `private` deklariert, um es einer abgeleiteten Klasse zu ermöglichen, diese Methode zu überladen, wenn der Transcodingvorgang z. B. durch einen einzelnen *SH* (dem `SH::SubGraphSH` selbst) implementiert wird.

Zusätzlich enthält diese Klasse zwei weitere Methoden, die nicht von einer der Basisklassen ererbt wurden. Es handelt sich dabei um `setInAnchorSH(Base *sh)` und `setOutAnchorSH(Base *sh)`. Diese Methoden dienen dazu, einen Senken- bzw. Quell-Endpunkt des übergebenen *SH* als den eigenen zu übernehmen. Dies ist notwendig, damit der *Graph-Manager* in der Lage ist, den `SH::SubGraphSH` mit dem restlichen Datenpfad zu verbinden. Hierzu wählen diese Methoden jeweils den ersten verfügbaren Endpunkt des übergebenen *SH* aus und weisen diesen ei-

ner internen Variable zu, so dass ein Verbinden eines anderen *Stream-Handlers* mit dem `SH::SubGraphSH` mit Hilfe der Methode `SH::Base::match` möglich ist. Eine abgeleitete Klasse muss diese Methoden innerhalb des Konstruktors aufrufen, da sonst kein Verbinden durch den `TranscodingGM` möglich ist. Damit diese Methoden jedoch Zugriff auf die Endpunkte des übergebenen *SH* erhalten, war es notwendig, die Methoden `getSinkSpec` und `getSourceSpec` der Klasse `SH::Status`[1] als `public` zu deklarieren. Ursprünglich waren diese Methoden als `protected` deklariert, was den Zugriff auf diese Methoden eines anderen Objekts unmöglich machte.

6.3 SubGraphFactory

Die Klasse `SubGraphFactory` dient dazu, ein Objekt einer von `SH::SubGraphSH` abgeleiteten Klasse aus einer Bibliothek zu laden oder wenn dies nicht möglich ist, ein Objekt der Klasse `SH::NopSubGraphSH` (siehe Abschnitt 6.4) zurückzuliefern. Die verwendete Programmiersprache C++ bietet allerdings – anders, als z. B. Java – keinen Mechanismus, zum dynamischen Laden einer Klasse zur Laufzeit eines Programms. Daher musste bei der Realisierung der Klasse `SubGraphFactory` auf einen, ursprünglich für die Programmiersprache C entworfenen Mechanismus zurückgegriffen werden. Bei diesem Mechanismus kann eine Bibliothek anhand ihres Namens mit Hilfe der Funktion `dlopen` zur Laufzeit geladen werden. Auf die Symbole[2] innerhalb dieser Bibliothek kann anschließend mit Hilfe der Funktion `dlsym` zugegriffen werden. Unter C handelt es sich bei den Symbolen um die Namen der Funktionen. Da C++ jedoch das Überladen von Methoden verschiedener Klassen unterstützt, ist das Symbol für eine bestimmte Methode nicht bekannt. Somit ist es auch nicht möglich, über `dlsym` auf ein derartiges Symbol zuzugreifen. Stattdessen muss innerhalb der zu ladenden Bibliothek eine C-Funktion existieren, die einen Zugriff auf die enthaltenen Klassen ermöglicht.

Realisiert wurde das dynamische Laden eines `SH::SubGraphSH` innerhalb der Methode `SubGraphFactory::getSubGraphSH`. Dieser werden neben dem Namen des Clients und dem Medientyp drei weitere Parameter übergeben, die für den Aufruf des Konstruktors von `SH::SubGraphSH` benötigt werden. Zunächst wird mit Hilfe von `dlopen` eine Bibliothek geladen, deren Namen aus der Konfigurationsdatei ermittelt wurde (siehe Abschnitt 6.1). Mit Hilfe der Funktion `dlsym` wird anschließend innerhalb der Bibliothek nach einem Symbol mit dem Namen `create` gesucht. Wird dieses Symbol gefunden, so wird die zugehörige Funktion mit den drei Parametern des Konstruktors von `SH::SubGraphSH` aufgerufen. Als Ergebnis dieses Aufrufs sollte ein Objekt der Klasse `SH::SubGraphSH` zurückgeliefert

[1] Die Klasse `SH::Status` ist eine Basisklasse von `SH::Base` und dient dazu, Statusinformationen über einen *SH* zur Verfügung zu stellen.

[2] Alle nicht statischen Funktionen eines C- und C++-Programms werden innerhalb der binären Datei (z. B. einer Bibliothek) durch eindeutige Symbole angesprochen

werden. Dieses Objekt wird dann an die aufrufende Methode zurückgegeben und kann von ihr benutzt werden. Falls beim Laden der Bibliothek oder beim Suchen des Symbols ein Fehler auftritt oder für den übergebenen Medientyp keine Bibliothek konfiguriert ist, wird ein Objekt der Klasse `SH::NopSubGraphSH` zurückgeliefert. Diese Klasse stellt einen einfachen Subpfad zur Verfügung, der den Medienstrom lediglich weiterleitet (siehe auch Abschnitt 6.4). Diese Verhalten wurde implementiert, damit immer ein gültiger Subpfad zurückgeliefert werden kann.

Für die aufrufende Methode ist das Verhalten von `getSubGraphSH` vollkommen transparent, da sie nicht feststellen kann, ob es sich bei dem zurückgelieferten Objekt um ein aus einer Bibliothek geladenes oder ein normales Objekt handelt. Da die Klasse, die `SubGraphFactory` benutzt, keine Kenntnis über die Klasse eines dynamisch geladenen Objekts besitzt, sondern es als ein Objekt der Klasse `SH::SubGraphSH` behandelt, sollte diese das Objekt nicht direkt zerstören. Dies könnte dazu führen, dass ein falscher Destruktor für das Objekt aufgerufen wird und nicht der gesamte belegte Speicher freigegeben wird. Stattdessen sollte für eine Zerstörung des Objekts dieses mit Hilfe der Methode `destroy` an `SubGraphFactory` zurückgegeben werden, damit der richtige Destruktor aus der Bibliothek geladen und aufgerufen werden kann. Zusätzlich führt diese Methode auch zum Schließen der zugehörigen Bibliothek. Zum Laden des Destruktors aus der Bibliothek ist es, wie beim Laden des Objekts selbst, notwendig, dass die Bibliothek ein bekanntes Symbol (`destroy`) enthält, über das das Objekt zerstört werden kann.

Damit ein Objekt aus einer Bibliothek geladen und später wieder zerstört werden kann, muss diese also zwei bekannte Symbole – `create` und `destroy` – enthalten, die einen Zugriff auf die enthaltene Klasse erlauben. Erreicht werden kann dies durch die Möglichkeit in C++, eine Funktion mit C-Bindung zu deklarieren. Dies geschieht durch Angabe von `extern "C"` bei der Deklaration der Funktion. Hier ein Beispiel aus der Klasse `SH::TestSubGraphSH`, die zum Testen des dynamischen Ladens implementiert wurde:

```
extern "C"{
    SH::SubGraphSH *create(SHGraphManager* mgr,
                           u_int32 id,
                           MNSelector& sel){
        return new SH::TestSubGraphSH(mgr, id, sel);
    }

    void destroy(SH::SubGraphSH *sh){
        delete sh;
    }
}
```

Im Normalfall hat jedoch die Klasse innerhalb der Bibliothek keinen Zugriff auf Symbole innerhalb des Hauptprogramms. Das bedeutet, dass alle innerhalb der Bibliothek benutzten Funktionen und Methoden in der Bibliothek enthalten sein müssen. Dies würde jedoch zu sehr großen Bibliotheken und wahrscheinlich auch zu Fehlfunktionen führen. Damit das aus der Bibliothek geladene Objekt Zugriff auf die Symbole des Hauptprogramms erhält, muss die Bibliothek mit der Option `-export-dynamic`[3] compiliert werden. Dies führt dazu, dass alle Symbole des Hauptprogramms der Tabelle dynamischer Symbole hinzugefügt und somit für die Klasse innerhalb der Bibliothek verfügbar werden. Beim Compilieren der Bibliothek selbst ist es notwendig, die Option `-fno-rtti`[3] zu verwenden, die das Erstellen von Laufzeit-Typ-Informationen unterbindet. Andernfalls würden diese, da *Komssys* standardmäßig ohne Erzeugung dieser Informationen compiliert wird, zu nicht auflösbaren Symbolen führen, die ein Laden der Bibliothek unmöglich machen.

`SubGraphFactory` unterstützt darüber hinaus das Laden mehrerer Objekte aus verschiedenen Bibliotheken. Dazu werden intern alle geladenen Bibliotheken und Objekte jeweils in einem `vector` abgelegt. Dieser erlaubt es, innerhalb der Methode `destroy` anhand des übergebenen Objekts die zugehörige Bibliothek und somit auch den entsprechenden Destruktor zu ermitteln. Diese Funktionalität ist für die vorliegende Implementation des Gateway-Entwurfs zwar nicht erforderlich, aber sie bietet für zukünftige Weiterentwicklungen die Möglichkeit, den Datenpfad weiter zu unterteilen und diesen aus mehreren Subpfaden zusammenzusetzen, die dann jeweils durch `SubGraphFactory` geladen werden können.

6.4 NopSubGraphSH

Wie bereits im vorherigen Abschnitt zur Klasse `SubGraphFactory` erwähnt, handelt es sich bei `SH::NopSubGraphSH` um die Implementation eines Subpfades, der lediglich die Daten vom `SH::PushPullSH` vor dem Subpfad abholt und diese an denjenigen nach dem Subpfad weiterleitet. Einerseits ist diese Klasse ein Beispiel für die Funktionsweise des `SH::SubGraphSH`, andererseits wurde sie implementiert, um sicherzustellen, dass die Methode `SubGraphFactory::getSubGraphSH` immer einen funktionsfähigen Subpfad liefert.

Intern benutzt diese Klasse den `SH::ActiveSH`, einen aktiven *Stream-Handler* aus dem *Komssys*-Projekt, der so konfiguriert wurde, dass er jede Millisekunde ein Datenpaket aus dem vorherigen `SH::PushPullSH` abholt und an den nachfolgenden übergibt. Die ererbten Methoden aus `SH::Base` wurden so implementiert, dass deren Aufruf jeweils an die entsprechenden Methoden des `SH::ActiveSH` weitergeleitet werden.

[3] Diese Optionen beziehen sich auf den Compiler `gcc-2.95`. Die Option `-export-dynamic` ist in der man-Page ld(1) und `-fno-rtti` in der Dokumentation des `gcc` beschrieben.

6.5 TranscodingGM

Der *Graph-Manager*, der den Datenpfad zusammenfügt, wurde durch die Klasse `TranscodingGM` realisiert. Diese wurde von der Klasse `ServerGM`, der Basisklasse aller *Graph-Manager* des Proxys bzw. des Servers, abgeleitet. Im Konstruktor dieser Klasse werden zunächst die *Stream-Handler* der beiden äußeren Teile des Datenpfades (siehe Abschnitt 5.3.1) erzeugt. Der Transcoding-Subpfad kann erst geladen werden, wenn dem `TranscodingGM` der Name des Clients und der Typ des Datenstroms, also das Medienformat, mitgeteilt wurden. Dies ist der Fall, wenn die Methode `open_file`[4] durch `MNStreamer` aufgerufen wird. In dieser Methode wird das Medienformat an die *SH* des Datenpfades übergeben und mit Hilfe der Klasse `SubGraphFactory` (siehe Abschnitt 6.3) ein `SH::SubGraphSH` geladen. Anschließend wird die Methode `getDelay()` des geladenen `SubGraphSH` aufgerufen und deren Rückgabewert an den `RTPEncoderSH` übergeben, damit dieser erst nach dieser Verzögerung mit dem Senden beginnt. Ist die zurückgelieferte Verzögerung zu kurz, kann es jedoch zu Beginn der Datenübertragung zu einer Blockierung des `RTPEncoderSH` durch einen leeren `PushPullSH` kommen. Zur Vermeidung dieser möglichen Blockierung wird der `PushPullSH` vor dem `RTPEncoderSH` so konfiguriert, dass ein Abholen der Daten erst möglich ist, wenn dieser bereits 30 Pakete enthält. Nachdem alle *SH* entsprechend konfiguriert wurden, werden diese miteinander verbunden, so dass am Ende der Methode `open_file` der Datenpfad vollständig geladen wurde und die Datenübertragung beginnen kann.

Zur Vermeidung der im Abschnitt 5.3.3 angesprochenen Blockierung des Datenpfades, enthält der `TranscodingGM` zwei zusätzliche Threads. Der erste Teil des Datenpfades wird innerhalb des Hauptthreads gestartet, in dem auch der `TranscodingGM` gestartet wurde. Der Transcoding-Subpfad und der letzte Teil des Datenpfades werden jeweils in einem eigenen Thread gestartet. Durch diese Realisierung kann kein Teil des Datenpfades einen vorherigen Teil durch Abholung der Daten blockieren und somit wird eine Blockierung des gesamten Datenpfades unterbunden.

6.6 Veränderungen am Client

Der Client von *Komssys* ist leider nicht in der Lage einen verzögerten Medienstrom korrekt darzustellen. Da allerdings beim Transcoding eines Medienstroms zwangsläufig eine Startverzögerung auftritt, war es notwendig den Client entsprechend anzupassen. Das Problem bestand darin, dass der Client nach der Bestätigung des *RTSP*-Play-Requests sofort einen Timer mit der Laufzeit des Videos gestartet hat, der nach Ablauf der Videosequenz den Client beendet hat.

[4] Der Name dieser Methode stammt ursprünglich aus dem *Graph-Manager* des Servers, da dieser die Datei, die den Medienstrom enthält, öffnen muss.

Trifft der Medienstrom unverzögert beim Client ein, so stellt dies auch kein Problem dar. Erst wenn die Daten verzögert eintreffen, führt diese Vorgehensweise dazu, dass der Client frühzeitig beendet wird und daher die Videosequenz nicht vollständig angezeigt wird. Normalerweise darf dieser Timer erst dann gestartet werden, wenn das richtige *RTP*-Paket, dessen Sequenznummer und Zeitstempel in der Bestätigung des Servers des *RTSP*-Play-Requests enthalten sind, beim Client eintrifft.

Um ein korrektes Verhalten des Clients zu erhalten, mussten zahlreiche Änderungen am Source-Code von *Komssys* vorgenommen werden, die an dieser Stelle nur angedeutet werden sollen. Eine vollständige, detaillierte Auflistung der Änderungen ist auf der beigefügten CD in der Datei `/src/CHANGES` zu finden.

Zunächst wurde eine Verbindung zwischen den Programmteilen, für die *RTSP*- sowie für die *RTP*-Kommunikation hergestellt. Hierzu wurde das Konzept, wie es bereits im Server und Proxy benutzt wurde, für den Client erweitert, so dass der *Graph-Manager* des Clients (`PlayerGM`) Zugriff auf Methoden des `RtspPlayer` erhalten hat. Zusätzlich musste die Sequenznummer und der Zeitstempel der Bestätigung des *RTSP*-Play-Requests an den `PlayerGM` übergeben werden, der diese wiederum an den `RTPDecoderSH` übergibt. Sobald nun im `RTPDecoderSH` ein RTP-Paket mit dieser oder einer höheren Sequenznummer bzw. mit gleichem oder höherem Zeitstempel eintrifft, wird der `PlayerGM` darüber informiert. Dieser ruft daraufhin eine Methode des `RtspPlayer` auf, durch die der Timer gestartet wird.

6.7 Evaluation

Zur Evaluation des implementierten Gateways war es erforderlich einen Transcoding-Subpfad zu implementieren, der aus einer Bibliothek geladen werden kann. Hierfür wurde die Klasse `TestSubGraphSH` implementiert, die wie die bereits beschriebene Klasse `NopSubGraphSH` die Daten aus dem ersten `PushPullSH` an den zweiten überträgt. Im Gegensatz zum `NopSubGraphSH` werden die Daten jedoch langsamer übertragen. Der benutzte `ActiveSH` überträgt alle 7 Millisekunden ein Datenpaket vom ersten an den zweiten `PushPullSH`. Die durch `getDelay` zurüchgegebene Verzögerung des Subpfades ergibt sich aus 10% der Dauer des Videostroms. Die benutzten Werte wurden durch Tests ermittelt und führten bei keiner Videosequenz zu einer Störung der Darstellung auf dem Client.

6.8 Benutzung des Gateways

Für die Benutzung des Gateways ist es erforderlich, drei verschiedene Programme zu starten: das Gateway selbst, einen Medienserver sowie einen Client. Es bietet sich an, als Server und Client ebenfalls die Programme des *Komssys*-Projektes zu benutzen[5].

An dieser Stelle wird die Konfiguration und der Start der einzelnen Programme auf drei verschiedenen Hosts beschrieben. Es wird davon ausgegangen, dass sich das ausführbare Programm sowie die Konfigurationsdatei jeweils im aktuellen Verzeichnis befindet. Um dieses Beispiel nachvollziehen zu können, sind auf der beiliegenden CD im Verzeichnis `/bin` bereits compilierte Versionen der Programme, eine Konfigurationsdatei sowie einige Videosequenzen zu finden.

6.8.1 Voraussetzungen und Konfigurations

Für den Start des Servers sowie des Gateways ist ein aktuelles Linuxsystem erforderlich, auf dem die XML-Bibliothek `expat` installiert ist. Getestet und entwickelt wurde das Gateway auf einem Linuxsystem mit dem Kernel 2.4.18 (Debian 3.0 - Woody). Für den Start des Clients ist zusätzlich eine Installation der QT-Bibliothek 2.2 sowie der MPEG-Bibliothek `mpeglib` erforderlich. Diese letzten beiden Voraussetzungen können beispielsweise durch eine Installation der grafischen Benutzeroberfläche KDE-2 mit dem zugehörigen Paket KDEMultimedia erfüllt werden.

Eine Installation ist nicht erforderlich. Für den Server und das Gateway ist es notwendig, das Programm `rtsp_server` sowie die Konfigurationsdatei `medianode.xml` auf den entsprechenden Host zu kopieren. Zusätzlich müssen dem Server Mediendateien mit zugehörigen Beschreibungsdateien zur Verfügung gestellt werden. Beispiele hierfür sind ebenfalls auf der beiligenden CD zu finden (siehe auch Anhand D). Für das Gateway sollte zusätzlich die Bibliothek `libtest.so`, die die Klasse `TestSubGraphSH` enthält, in das gleiche Verzeichnis wie das Programm kopiert werden. Für den Client muss das Programm `center` zusammen mit der Konfigurationsdatei auf den entsprechenden Host kopiert werden. Die Konfigurationsdatei sollte sich jeweils in dem Verzeichnis befinden, aus dem heraus das jeweilige Programm gestartet wird. Es bietet sich also an, sowohl das Programm als auch die Datei `medianode.xml` in ein Verzeichnis auf den jeweiligen Host zu kopieren und vor dem Start des Programms in dieses zu wechseln.

[5]Der Proxyserver sollte zwar auch mit fremden Servern und Clients zusammenarbeiten, jedoch konnte dies im Rahmen dieser Arbeit nicht näher getestet werden.

6.8.2 Start des Servers und des Gateways

Für die Ausführung des Servers und des Gateways ist jeweils das Programm `rtsp_server` zu starten. Für den Server wird dieses ohne Angabe von Parametern gestartet, für das Gateway mit dem Parameter `-p`:

Start des Servers:	`./rtsp_server`
Start des Gateways:	`./rtsp_server -p`

Durch die Angabe des Paramters `-p` wird der Server von *Komssys* im Proxy-Modus gestartet und bei aktiviertem Transcoding der `TranscodingGM` geladen. Nach dem Start des jeweiligen Programms wird die benutzte Konfiguration angezeigt. Wurden in der Konfigurationsdatei die Debug-Ausgaben aktiviert, so erscheinen diese im weiteren Verlauf des Programms auf der Standard-Fehler-Ausgabe. Für eine störungsfreie Übertragung der Daten ist es allerdings erforderlich, die Debug-Ausgaben in eine Datei umzuleiten oder besser, diese durch die Angabe von `<no-info-print/>` innerhalb der Konfigurationsdatei abzuschalten. Wurden die Debug-Ausgaben abgeschaltet, sollten innerhalb des Blocks `<do-info>` keine Dateien angegeben sein oder der Block sollte komplett entfernt werden, da andernfalls für die angegebenen Dateien dennoch Debug-Ausgaben erfolgen.

6.8.3 Start des Clients

Bei dem Client von *Komssys* handelt es sich um ein grafisches Programm. Es ist allerdings auch möglich den Client ohne grafische Oberfläche zu starten. Letzteres wird durch den Aufruf

```
./center --url rtsp://<servername>:<port>/<mediendatei>
```

erreicht. Als `<port>` muss der Port des Servers angegeben werden, auf dem dieser Anfragen von Clients erwartet. Wird die auf der CD enthaltene Konfigurationsdatei benutzt, so handelt es sich um den Port 9070. Die Mediendatei muss beim Aufruf des Clients so angegeben werden, wie diese auf dem Server vorliegt.

6.8.4 Beispiel

In der folgenden Auflistung sind die zu kopierenden Dateien sowie der für den Start des Programms auszuführende Befehl noch einmal zusammengefasst. Es wird hierbei davon ausgegangen, dass die Programme auf den verschiedenen Hosts jeweils in das Verzeichnis `/home/komssys` kopiert wurden.

Server:

Host:	welt
Inhalt von `~komssys`:	`rtsp_server`
	`medianode.xml`
	`video/pertplus.mpg`
	`video/pertplus.desc`
Befehl:	`cd /home/komssys`
	`./rtsp_server`

Gateway:

Host:	mephisto
Inhalt von `~komssys`:	`rtsp_server`
	`libtest.so`
	`medianode.xml`
Befehl:	`cd /home/komssys`
	`./rtsp_server -p`

Client:

Host:	faust
Inhalt von `~komssys`:	`center`
	`medianode.xml`
Befehl:	`cd /home/komssys`
	`./center --url rtsp://welt:9070/pertplus.mpg`

Für ein Testen des Gateways sollte auch der Inhalt der Datei `README` im Verzeichnis `/bin` auf der beiliegeden CD beachtet werden.

7 Zusammenfassung und Ausblick

7.1 Zusammenfassung

Durch die zunehmende drahtlose Integration multimediafähiger tragbarer Geräte in das Internet entsteht die Notwendigkeit, audiovisuelle Datenströme an die Bedürfnisse dieser Geräte anzupassen. Häufig besitzen mobile Geräte nur eine relativ geringe Hardwareausstattung, die eine störungsfreie Wiedergabe hochauflösender Videosequenzen unmöglich macht. Aus dieser Motivation heraus wurden viele verschiedene Techniken entwickelt, die ein Transcoding der Daten in angemessener Zeit ermöglichen. Es existiert allerdings keine Universallösung für die Umwandlung der Datenströme. Die Frage, ob die Umwandlung der Daten in codierter Form erfolgen soll oder ob eine teilweise Decodierung notwendig ist, scheint die Wissenschaftler, die sich mit diesem Thema beschäftigen, in zwei Lager zu spalten. Einerseits bietet die Bearbeitung der Daten ohne vorherige Decodierung den Vorteil, dass auf die rechenintensive Decodierung verzichtet werden kann und somit mehr Rechenzeit für ein qualitativ hochwertiges Transcoding genutzt werden kann. Andererseits ist die Bearbeitung codierter Daten weitaus rechenintensiver als eine äquivalente Bearbeitung decodierter Daten. Zwar bestehen für das Transcoding ohne Decodierung durch Ausnutzung mathematischer Eigenschaften der durchzuführenden Berechnungen gewisse Einsparpotentiale, jedoch ist bis heute nicht geklärt, ob diese in allen Fällen ausreichen, um den Vorteil der Bearbeitung decodierter Daten aufzuheben.

Im Kapitel 4 dieser Arbeit wurde ein tiefgreifender Einblick in verschiedene Techniken des Transcodings gegeben. In der Literatur zum Thema Transcoding ist solch ein umfassender Überblick nach Kenntnis des Autors nicht zu finden. Es wurden sowohl Techniken ohne vorherige Decodierung als auch mit teilweiser Decodierung betrachtet. Welches Verfahren für das Transcoding eines bestimmten Medienstroms am sinnvollsten ist, lässt sich jedoch nicht mit Bestimmtheit sagen. Es muss sicherlich im Einzelfall entschieden werden, welche Technik das beste Ergebnis bei möglichst geringer Komplexität liefert. Aus diesem Grund wurden die vorgestellten Verfahren auch nicht bewertet.

Die Vielzahl verschiedener Transcodingtechniken sowie die Komplexität der verschiedenen Videocodierungen ergibt also nahezu die Unmöglichkeit der Implementation eines universellen Transcoders. Der in Kapitel 5 vorgestellte Entwurf des Multimedia-Gateways nimmt diese Tatsache auf, indem sehr viel Wert auf die Flexibilität des Gateways gelegt wurde.

Die Frage, ob ein Transcoding überhaupt in Realzeit möglich ist, lässt sich ebenfalls nicht eindeutig beantworten. In den verschiedenen Arbeiten zu den vorgestellten Techniken wurden kaum Angaben darüber gemacht, ob der Transcodingvorgang mit heutiger Technologie in Realzeit erfolgen kann. Stattdessen wurde viel Wert auf die erreichbare Qualität der transcodierten Daten gelegt. Dies ist sicherlich legitim, jedoch ist es für die Entwicklung und die Akzeptanz durch die Nutzer eines Multimedia-Gateways auch wichtig, dass durch die Umwandlung des Medienstroms keine Verzögerungen bei der Darstellung auf dem Client entstehen. Um dieser Notwendigkeit Sorge zu tragen, wurde beim Entwurf insbesondere auch darauf geachtet, dass keine Einschränkungen für die Implementation des Transcodingprozesses eingeführt wurden. Durch das dynamische Laden des Transcoding-Subpfades ist es z. B. auch möglich, diesen durch eine fremde Bibliothek oder in einer anderen Programmiersprache als der benutzten zu realisieren. Die einzige Voraussetzung zur Nutzung solch eines 'fremden' Subpfades ist lediglich, dass innerhalb der Bibliothek die Symbole `create` und `destroy` zum Laden bzw. Zerstören des Objekts zur Verfügung stehen.

Durch die Implementation, die im Kapitel 6 vorgestellt wurde, konnte die Realisierbarkeit des entwickelten Entwurfs des Gateways gezeigt werden. Durch die starke Integration in die Basisimplementation *Komssys* sowie durch die Verwendung objektorientierter Programmiertechniken wurde auch eine einfache Weiterentwicklung des Gateways sichergestellt. Dass der Entwurf des Gateways in die richtige Richtung weist, zeigt sich auch daran, dass einer der Entwickler von *Komssys* – Carsten Griwodz – die Möglichkeit, einen Subpfad zur Laufzeit des Gateways zu laden, sehr begrüßt hat [47].

7.2 Diskussion

Aufgrund der Komplexität des Themas Transcoding audiovisueller Daten, kann der hier vorgestellte Entwurf eines Multimedia-Gateways nur ein erster Ansatz sein. Einerseits existieren Aspekte, die in dieser Arbeit gar nicht oder nur teilweise betrachtet werden konnten. Andererseits wurden auch nicht alle selbst gestellten Designziele des Gateways erfüllt. Letztere werden an dieser Stelle noch einmal aufgegriffen und diskutiert. Die nicht oder nur teilweise betrachteten Aspekte werden anschließend im Kapitel 7.3 aufgegriffen.

7.2.1 Caching

Ein Designziel, das im Abschnitt 5.1 erwähnt, aber im Entwurf selbst noch nicht betrachtet wurde, ist die Integration einer Cachingfunktionalität in das Gateway. Wie im Abschnitt 5.2.6 dargestellt wurde, wird das Caching innerhalb des Proxys durch einige zusätzliche *Stream-Handler* erreicht, die in den Datenpfad eingebunden werden. Mit Hilfe eines `MultiplierSH` wird der Datenstrom dupliziert und an einen `RTPDecoderSH` übergeben. Dieser entfernt die *RTP*-Header der Daten und gibt diese weiter an einen `FileSinkSH`, der die Daten in eine Datei schreibt. Da der Transcoding-Datenpfad bereits einen `RTPDecoderSH` enthält, wäre es für das Schreiben der Daten in eine Datei lediglich notwendig, den Datenpfad durch einen `MultiplierSH` und einen `FileSinkSH` zu erweitern. Es stellt sich jedoch die Frage, ob es sinnvoller ist, die Originaldaten oder die umgewandelten Daten in der Datei abzulegen. Abhängig von dieser Entscheidung müsste der `MultiplierSH` entweder vor dem ersten oder nach dem zweiten `PushPullSH` in den Datenpfad integriert werden. Des Weiteren müssten Strategien entwickelt werden, anhand derer entschieden werden könnte, ob und wie viel eines Medienstroms zwischengespeichert werden soll. Hierfür bieten die in *Komssys* bereits enthaltenen Strategien eine gute Vorlage.

7.2.2 Transparenz

Eine weitere Anforderung an das Gateway ist dessen Transparenz gegenüber dem Client. Diese Forderung geht jedoch über den Entwurf des Gateways an sich hinaus, da hierzu auch gewisse Rahmenbedingung im Netzwerk erfüllt sein müssen. Eine teilweise Transparenz ist bereits dadurch gegeben, dass sich das Gateway gegenüber dem Client wie ein normaler Medienserver verhält. Für die Nutzung ist es jedoch erforderlich, dass das Gateway auf dem Client als Proxy eingestellt ist. Ebenso muss die Konfiguration des Gateways so an die Bedürfnisse des Clients angepasst werden, dass die Medienströme in geeigneter Weise transcodiert werden. Genau genommen sind es also zwei Punkte, die eine vollkommene Transparenz des Gateways verhindern:

1. Das Gateway muss auf dem Client als Proxy konfiguriert werden.
2. Das Gateway muss für die Bedürfnisse des Clients konfiguriert werden.

Der erste Punkt könnte dadurch relativiert werden, dass innerhalb des Netzwerkes, in dem sich der Client befindet, alle *RTSP*-Anfragen auf das Gateway umgeleitet werden. Dies würde jedoch dazu führen, dass eine Medienübertragung von Servern innerhalb des Clientnetzwerkes, abgesehen vom Gateway selbst, nicht mehr möglich wäre. Um dieses dennoch zu ermöglichen, müsste das Gateway so

im Netzwerk platziert werden, dass sämtlicher Datenverkehr der Clients dieses passieren würde. Hierzu könnte der Gateway-Host als Router konfiguriert werden und Anfragen auf den Standard-*RTSP*-Ports mit Hilfe eines Paketfilters[1] an lokale Ports umgeleitet werden, so dass sie vom Gateway-Prozess entgegengenommen und bearbeitet werden können.

Eine Lösung des zweiten Punktes ist jedoch etwas aufwendiger und benötigt weitere Forschungsarbeiten. Ein Ansatz besteht darin, dass *RTSP* die Möglichkeit bietet, ähnlich wie *HTTP*, einen Identifikations-Angabe der Clientsoftware in jede Anfrage an einen Server zu integrieren. Diese Angabe könnte durch das Gateway ausgewertet werden, um gewisse Charakteristiken des Clients aus einer Datenbank zu lesen. Diese Angabe ist jedoch weder zwingend noch ist deren Inhalt eindeutig bestimmt. Es wäre unter Umständen also auch eine Anpassung der Clientsoftware notwendig.

7.3 Ausblick

Wie anhand dieser Arbeit bereits zu erkennen ist, handelt es sich bei dem Thema des Transcodings audiovisueller Daten um einen sehr komplexen Forschungsbereich. Somit bietet dieser Bereich viele Möglichkeiten für weitere Forschungsarbeiten. Im letzten Abschnitt wurden bereits zwei Themen genannt, die als Grundlage weiterer Arbeiten dienen könnten. Im Folgenden werden weitere Möglichkeiten, die in einem engen Zusammenhang mit dieser Arbeit stehen, vorgestellt.

7.3.1 Weitere Formate

Diese Arbeit beschränkt sich auf vier Videoformate, die häufig für die Videocodierung eingesetzt werden. Es existiert jedoch eine Vielzahl weiterer Formate die im Rahmen dieser Arbeit nicht betrachtet werden konnten. Einerseits wurden im Bereich der Videocodierung Fortschritte erzielt was zu neuen Formaten, wie z. B. MPEG-4 und H.264, führte. Andererseits existieren weit verbreitete proprietäre Medienformate, die im Wesentlichen die Techniken der hier vorgestellten Formate einsetzen oder diese erweitern. Eine Ausweitung dieser Arbeit auf andere Formate wäre also sicherlich sinnvoll.

Bei den jüngsten Medienformaten wie MPEG-4 oder H.264 stehen nach Kenntnis des Autors jedoch noch kaum Ansätze für ein Transcoding zur Verfügung. Hier könnte untersucht werden, inwiefern die in dieser Arbeit vorgestellten Techniken auch auf diese Formate angewandt werden können oder dafür erweitert werden müssen.

[1] Unter Linux könnte z. B. der im Kernel enthaltene Paketfilter `ipchains` genutzt werden.

7.3.2 Audio-Transcoding

Wie bereits zu Beginn dieser Arbeit erwähnt wurde, ist der Bereich des Audio-Transcoding relativ unerforscht. Die Fähigkeit, Audiosignale umzuwandeln, um geringere Datenraten zu erhalten, würde jedoch sicherlich einige Vorteile für die Medienwiedergabe auf mobilen Geräten bieten. Für Benutzer solcher Geräte, die keine HiFi-Qualität bei der Audio-Wiedergabe erwarten, könnten höhere Kompressionsraten oder eine Reduktion auf einen Kanal eingesetzt werden.

7.3.3 Weitere Implementation

Der Entwurf des Multimedia-Gateways umfasst lediglich den Rahmen für das Multimedia-Gateway, nicht aber das Transcoding selbst. Da der Transcodingvorgang keine triviale Aufgabe ist, konnte dieser in dieser Arbeit nicht implementiert werden. Eine gute Implementation, die mehrere Techniken umfasst und dabei effiziente Algorithmen einsetzt, wäre ein interessanter Inhalt weiterer Arbeiten. Insbesondere könnte die Konvertierung von interlaced MPEG-2-Strömen in progressive Ströme untersucht werden. Auch eine effiziente und intelligente Kontrolle der Datenrate würde in diesen Bereich fallen.

A Beispiel einer Konfigurationsdatei von Komssys

```
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE komssysconf>
<komssysconf>
    <system type="client">
        <no-tfrc-reply />
        <gui>
            <autostart />
            <decoder />
            <no-qt/>
        </gui>
        <datapump>
            <type value="MNSTREAMER" />
            <sendblock/>
        </datapump>
        <rtsp>
            <destination value="yes"/>
            <eol value="crlf" />
            <default-server-port value="9070" />
            <use-proxy/>
            <proxy>
                <host value="norge" />
                <port value="9072" />
            </proxy>
        </rtsp>
        <sdp>
            <eol value="crlf" />
        </sdp>
    </system>

    <system type="server">
        <no-tfrc-reply />
        <no-patching/>
        <datapump>
            <type value="MNSTREAMER" />
        </datapump>
        <rootdir value="/opt/komssys/video" />
```

```
        <rtsp>
            <default-server-port value="9070" />
        </rtsp>
        <debug>
            <info-print />
            <warn-print />
            <error-print />
            <info-dont>
                <file value="MNSelector" />
            </info-dont>
        </debug>
    </system>

    <system type="proxy">
        <no-tfrc-reply />
        <no-patching/>
        <caching/>
        <caching-strategy value="FIFO"/>
        <cachedir value="/tmp" />
        <cachesize value="1" />
        <datapump>
            <type value="MNSTREAMER" />
        </datapump>
        <rtsp>
            <default-server-port value="9072" />
        </rtsp>
        <transcoding>
            <use-transcoding value="yes"/>
            <libraries>
                <transcoding-lib pt="MPV" lib="./libmpeg.so"/>
                <transcoding-lib pt="H261" lib="./libh261.so"/>
            </libraries>
            <client name="faust">
                <screenwidth value="640"/>
                <screenheight value="200"/>
                <libraries>
                    <transcoding-lib pt="MPV" lib="./libmpeg2h263.so"/>
                </libraries>
            </client>
        </transcoding>
        <debug>
            <no-info-print />
            <warn-print />
            <error-print />
            <info-do>
                <file value="PushPullSH"/>
            </info-do>
        </debug>
    </system>
</komssysconf>
```

B Medienformate in der Konfigurationsdatei

In der Einstellungsdatei werden für die Angabe einer Bibliothek für ein bestimmtes Medienformat die folgenden Bezeichnungen für die Medienformate benutzt. Die Bezeichnungen entsprechen denen, die intern in *Komssys* benutzt werden und stammen mehrheitlich aus dem RFC 1890 [45]. Von *Komssys* unterstützt werden nur diejenigen, bei denen eine Bemerkung mit einer verständlicheren Bezeichnung vorhanden ist.

Bezeichnung	Audio/ Video	Frequenz in Hz	Kanäle (Audio)	Bemerkung
PCMU	A	8000	1	einfaches Audioformat
1016	A	8000	1	
G721	A	8000	1	
GSM	A	8000	1	
G723	A	8000	1	
DVI4	A	8000	1	
DVI42	A	16000	1	
LPC	A	8000	1	
PCMA	A	8000	1	
G722	A	8000	1	
L16	A	44100	2	
L162	A	44100	1	
QCELP	A	8000	1	
CN	A	8000	1	
MPA	A	8000	1	MP3
G728	A	8000	1	
DVI43	A	11025	1	
DVI44	A	22050	1	
G729	A	8000	1	
CELLB	V	90000		
JPEG	V	90000		
CUSM	V	90000		
NV	V	90000		
PICW	V	90000		
CPV	V	90000		

Bezeichnung	Audio/ Video	Frequenz in Hz	Kanäle (Audio)	Bemerkung
H261	V	90000		H.261 Video
MPV	V	90000		MPEG-1 Systems und MPEG-1/2 Video
MP2T	AV	90000		
H263	V	90000		
SPEG	V	90000		Scalable MPEG

C Auflistung aller implementierten und bearbeiteten Klassen

An dieser Stelle sind noch einmal alle Klassen aufgelistet, die vom Autor im Rahmen dieser Arbeit implementiert bzw. bearbeitet wurden.

C.1 Vom Autor implementierte Klassen

- `EConfTranscoding`
- `EConfTranscodingClient`
- `EConfTransLib`
- `EConfTransLibList`
- `SH::NopSubGraphSH`
- `SH::SubGraphSH`
- `SH::TestSubGraphSH`
- `SubGraphFactory`
- `TranscodingGM`

C.2 Vom Autor bearbeitete Klassen

- `EConfFront`
- `GMUser`
- `MNConfig`

- `MNRTSPClient`
- `MNSessionState`
- `MNStreamer`
- `RtspPlayer`
- `ServerGM`
- `SH::RTPEncoderSH`
- `SH::Status`
- `SHGraphManager`

D Inhalt der CD

Auf der beigefügten CD ist der gesamte Source-Code zu finden, der in dieser Arbeit benutzt und erweitert wurde. Im Hauptverzeichnis der CD sowie in einigen Unterverzeichnissen befinden sich zusätzlich Textdateien mit dem Namen `README`, die nähere Informationen und Hinweise zum Inhalt der einzelnen Verzeichnisse und der CD enthalten.

Verzeichnisstruktur

`/`	Im Hauptverzeichnis der CD befindet sich eine Textdatei mit dem Namen `README` die nähere Informationen zum Inhalt der CD enthält.
`doc/`	Dieses Verzeichnis enthält sämtliche Quellen, die im Laufe dieser Arbeit für die Dokumentation erstellt wurden. Zusätzlich sind hier zwei digitale Versionen dieser vorliegenden Arbeit zu finden, eine im PostScript- und eine im PDF-Format.
`html/`	Das Verzeichnis `html` enthält die HTML-Dateien, die zu Beginn dieser Arbeit als kurzer Überblick zum Thema erstellt wurden. Innerhalb dieser Dateien befinden sich auch einige Internetadressen, die weitere Informationen zu einzelnen Themen bieten.
`tex/`	Innerhalb dieses Verzeichnisses sind alle Quelldateien zu finden, die zum Erstellen dieses Dokuments benutzt wurden. Es handelt sich um reine Textdateien, die den Text im LaTeX-Format enthalten. Die Grafiken befinden sich in EPS-Dateien ebenfalls in diesem Verzeichnis.
`vortrag/`	In diesem Verzeichnis sind die Folien des Zwischenvortrages zu finden. Es handelt sich um HTML-Dateien, die jeweils eine Folie enthalten.

`src/` Dieses Verzeichnis enthält den Source-Code, der für diese Arbeit benutzt und z.T. neu erstellt wurde sowie eine automatisch generierte Dokumentation des Quellcodes. Für ein erfolgreiches Compilieren des Quellcodes ist es erforderlich, den Inhalt der Datei `README` in diesem Verzeichnis zu beachten. Zusätzlich befindet sich in diesem Verzeichnis noch die Datei `CHANGES`, die nähere Informationen zu den Änderungen enthält, die im Rahmen dieser Arbeit am Quellcode vorgenommen wurden. In welchen Unterverzeichnissen die neu implementierten Klassen zu finden sind, ist ebenfalls in dieser Datei enthalten.

- `doc/` In dem Verzeichnis `doc` ist eine Dokumentation im HTML-Format zu finden, die mit Hilfe des Programms `doxygen` automatisch aus dem Source-Code generiert worden sind.
- `komssys/` Dieses Verzeichnis enthält den kompletten Quellcode von *Komssys* sowie den Code aller neu implementierten Klassen.

`bin/` In diesem Verzeichnis sind binäre Versionen der einzelnen Programme von *Komssys* sowie eine Konfigurationsdatei `medianode.xml` enthalten. Diese wurden auf einem Linux-System (Debian 3.0 - Kernel 2.4.18) mir Hilfe des Compilers `gcc-2.95` erzeugt.

- `video/` Da es für den Server von *Komssys* notwendig ist, dass für jede Mediendateien eine zugehörige Beschreibungsdatei existiert, die die Daten für *SDP* enthält, befinden sich in diesem Verzeichnis zwei Beispielvideos im MPEG-1-Format mit zugehörigen Beschreibungsdateien.

E Glossar

AAC Advanced Audio Coding
Ein Audio-Kompressionsfomat, das von der *Moving Pictures Expert Group* 1997 zum MPEG-2-Standard hinzugefügt wurde und bessere Qualität gegenüber *MP3* sowie Unterstützung von fünf Kanälen bietet [24].

AC-Koeffizient
Als AC-Koeffizienten werden alle, außer dem $(0, 0)$-Koeffizienten eines DCT-Blocks bezeichnet, der bei der Anwendung der *DCT* auf einen Bildblock entsteht.

ADPCM Adaptive Differential Pulse Code Modulation
Eine Quellcodierung, bei der ein analoges Signal zu bestimmten Zeitpunkten abgetastet wird. Die Abtastzeitpunkte sowie die Dauer der einzelnen Abtastungen werden anhand der Signalcharakteristik bestimmt. Gespeichert werden die Differenzen aufeinander folgender Abtastwerte (siehe auch *PCM* und *DPCM*).

ADVS Activity Dominant Vector Selection
Eine Methode zur Aktualisierung der Bewegungsvektoren nicht verworfener Frames, die z.T. bessere Ergebnisse liefert, als die *FDVS* [10].

CIF Common Interchange Format
Eine Auflösung von 352x288 Pixel, die von H.261 und H.263 unterstützt wird.

CPDT Cascaded Pixel-Domain Transcoder
Ein nahe liegender Ansatz eines Transcoders, bei dem ein Decoder und ein Encoder hintereinander geschaltet werden. Der Datenstrom zunächst vollständig decodiert und anschließend mit anderen Parametern wieder codiert.

DC-Koeffizient
Mit DC-Koeffizient wird der $(0, 0)$-Koeffizient eines DCT-Blocks bezeichnet, der bei der Anwendung der *DCT* auf einen Bildblock entsteht.

DCT Diskrete Cosinus Transformation
Eine mathematische Abbildung, die bei der räumlichen Kompression von bewegten und unbewegten Bildern eingesetzt wird.

DDT DCT-Domain Transcoder
Eine Transcoder-Architektur, bei der sämtliche Berechnungen im Frequenzbereich stattfinden.

DPCM Differential Pulse Code Modulation
Ein Quellcodierverfahren, bei dem ein kontinuierliches Signal zu festen Zeitpunkten abgetastet wird und die so ermittelten Werte jeweils als Differenz zum vorherigen Wert abgespeichert werden.

Entropiecodierung
Eine Codierung, bei der die Länge der einzelnen Codewörter von der Entropie der Quelle abhängig ist. Der Entropiebegriff wurde von Claude E. Shannon 1948 eingeführt und gibt den Informationsgehalt einer Quelle an. Der Huffmancode ist wohl der berühmteste Vertreter dieser Codierungen.

FDVS Forward Dominant Vector Selection
Eine Methode, die beim Frameskipping innerhalb eines Transcoders eingesetzt werden kann, um die Bewegungsvektoren nicht verworfener Frames zu aktualisieren [8].

FTP File Transfer Protocol
Bei *FTP* handelt es sich um ein Protokoll der Anwendungsschicht, das im Internet zur Übertragung von Dateien genutzt wird.

GM Graph-Manager
Bei einem Graph-Manager handelt es sich um eine Kontrolleinheit, die innerhalb von Komssys dazu genutzt wird, mehrere Stream-Handler zu einem Datenpfad zu verbinden (siehe auch *Stream-Handler (SH)*)

GOB Group of Blocks
Hiermit wird die Gruppierung einzelner Makroblöcke bei H.261/3 bezeichnet.

GOP Group of Pictures
Bei MPEG-1/2 werden die einzelnen Bilder zu Gruppen zusammengefasst. Jede Gruppe beginnt mit einem I-Frame und kann eine beliebige Anzahl Bilder enthalten.

HAVS Horizontal and Vertical Search
Ein Suchschema zur Bewegungsvektoranpassung. Hierbei wird innerhalb eines Suchfensters zunächst in horizontaler und anschließend in vertikaler Richtung ein Minimum der *SAD* gesucht.

HTTP Hyper Text Transfer Protocol
Ein Klartextprotokoll der Anwendungsschicht, das hauptsächlich zur Übertragung von Internetseiten eingesetzt wird.

Huffmancodierung
Eine Entropiecodierung, die von D.A. Huffman, einem Schüler Shannons, 1952 entwickelt wurde. Der durch dieses Verfahren entstehende Code ist optimal bzgl. der mittleren Codewortlänge, die minimal ist.

IEC International Electrotechnical Commission
Eine globale Organisation, die internationale Standards für elektrische und elektronische Technologien entwickelt und veröffentlicht.
http://www.iec.ch

ISDN Integrated Services Data Network
Bei *ISDN* handelt es sich um ein digitales Telefonnetz, das hauptsächlich in Deutschland eingesetzt wird. Ein *ISDN*-Anschluss bietet zwei Datenkanäle mit einer Bandbreite von je 64 kBits/s und einem Steuerkanal mit 16 kBits/s.

ISO International Organisation of Standardization
ISO ist eine 1946 gegründete freiwillige Organisation mit Sitz in Genf, zur Schaffung internationaler Standards. Mitglieder sind fast alle nationalen Standardisierungsinstitutionen.
http://www.iso.org

ITU International Telecommunication Union
Eine 1865 gegründete, internationale Organisation zur Standardisierung im Bereich der Telekommunikation.
http://www.itu.org

JPEG Joint Photographic Experts Group
Eine Kommission, die seit 1988 Verfahren entwickelt, um Bilder platzsparend abzuspeichern. Der JPEG-Standard ist ein weit verbreitete Kompressionsverfahren für Bilder, das sehr stark im Internet genutzt wird.
http://www.jpeg.org

Komssys KOM(S) Streaming System
Komssys ist eine frei verfügbare RTSP/RTP-Implementierung für den Forschungsbereich der Medienübertragung im Internet-Umfeld [38].
http://komssys.sourceforge.net

MP3 MPEG-1 Layer III
Mit MP3 wird ein sehr bekanntes und weit verbreitetes Audioformat bezeichnet, das im MPEG-1-Standard [23] enthalten ist. Der Standard defi-

niert drei verschiedene Layer, die sich anhand ihrer Komplexität und erreichbarer Kompression unterscheiden.

MPEG Moving Pictures Expert Group
Eine 1988 gegründete Arbeitsgruppe der *ISO* und der *IEC*. Ziel dieser Gruppe ist die Entwicklung von Standards zur Kompression und Codierung von bewegten Bildern.
http://www.cselt.it/mpeg/

PCM Pulse Code Modulation
Ein Quellcodierverfahren zur Digitalisierung analoger Signale. Das Signal wird hierbei zu festen Zeitpunkten abgetastet und der ermittelte Wert abgespeichert. Dieses Verfahren kommt u.a. bei der Speicherung von Audiosignalen auf CD's zum Einsatz.

PDA Personal Digital Assistent
PDA ist die Bezeichnung einer sehr kompakten Geräteklasse, die vornehmlich als Kalender und Notizbuch genutzt werden kann, aber auch einige vom Desktopcomputer bekannte Applikationen bietet. Die meisten dieser Geräte, die auch als Handheld bezeichnet werden, besitzen ein verhältnismäßig kleines Graustufen- oder Farbdisplay und nur sehr begrenzte Rechenkapazitäten.

QCIF Quater Common Interchange Format
Eine Auflösung von 176x144 Pixel, die von H.261 und H.263 unterstützt wird.

RFC Request for Comment
Internetstandards der Internet Engineering Task Force (IETF) werden in Form von *RFC*s veröffentlicht und sind als solche frei abrufbar.

RGB
Die Abkürzung *RGB* steht für die drei Farben Rot, Grün und Blau. Im Bereich der Bildbearbeitung kann mit den Werten dieser drei Farben ein Pixel eines digitalen Bildes beschrieben werden.

RTP Real-Time Transport Protocol
Ein Internetprotokoll der Anwendungsschicht, das zur Übertragung kontinuierlicher Echtzeitdaten dient [30].

RTCP Real Time Control Protocol
Dieses Internetprotokoll dient als Signalisierungs- und Steuerungsprotokoll innerhalb einer *RTP*-Übertragung [30].

RTSP Real Time Streaming Protocol

Ein Klartextprotokoll der Anwendungsschicht, das im Internet zur Signalisierung und Kontrolle von Echtzeitübertragungen (z. B. mittels *RTP*) dient [28].

SAD Summe absoluter Differenzen

Ein Wert, der für die Suche von Bewegungsvektoren eingesetzt wird, um die größte Übereinstimmung zu finden.

$$SAD(m,n) = \sum_{i}^{M} \sum_{j}^{N} |P(i,j) - R(i+m, j+n)|$$

Wobei $P(i,j)$ und $R(i,j)$ die Pixelwerte im aktuellen bzw. im Referenzframe sind.

SDP Session Description Protocol

Ein Protokoll, das zur Beschreibung von Sitzungsinhalten dient und im Zusammenhang mit *RTSP* und *RTP* zum Einsatz kommt [29].

SDRE Sum of Differential Reconstruction Error

Ein Kriterium, das es erlaubt zu entscheiden, ob ein Bewegungsvektor bei der Transcodierung angepasst werden sollte oder nicht.

$$SDRE(B_x, B_y) = \left| \left(\frac{q_1^c}{q_2^p} \right)^2 - 1 \right| \sum_{i} \sum_{j} |R(i+B_x, j+B_y) - P(i+B_x, j+B_y)|$$

$P(i,j)$ und $R(i,j)$ sind die Pixelwerte im aktuellen bzw. im Referenzframe, q_1^c ist der Quantisierungsfaktor des aktuellen Blocks und q_2^p der Quantisierungsfaktor des Blocks, auf den der Basisvektor B verweist, der bei der Quantisierung im Transcoder benutzt wird. [8]

SIF Source Input Format

Eine Auflösung von 352 x 240 Pixel bei 30 Bildern/s bzw. 352 x 288 Pixel bei 25 Bildern/s, das im MPEG-1-Standard eingeführt wurde.

SH Stream-Handler

Bei Stream-Handler handelt es sich um kleine Programmteile bzw. Objekte, die in der Lage sind einen Datenstrom zu verarbeiten. Innerhalb einer Applikation können diese von einer Kontrolleinheit zu einem Datenpfad verbunden werden, durch den der Datenstrom hindurchfließt und dabei von den Stream-Handlern manipuliert werden kann. Siehe auch *Graph-Manager (GM)*.

Tag XML-Tag
Als Tag wird bei XML, ähnlich wie auch bei HTML, eine Zeichenfolge bezeichnet, die durch spitze Klammern eingeschlossen ist. Zwischen diesen Klammern sind der Name des Tags sowie optionale Parameter angegeben. Auf ein einleitendes Tag muss in XML immer ein abschließendes Tag folgen, das durch einen Schrägstrich nach der ersten Klammer markiert ist. Zusätzlich existieren in XML sog. selbstabschließende Tags, die sowohl ein einleitendes als auch ein abschließendes Tag darstellen. Diese sind durch einen Schrägstrich vor der letzten Klammer gekennzeichnet. Beispiele für XML-Tags sind im Anhang A zu finden.

TCP Transmission Control Protocol
Eines der wichtigsten Protokolle des Internets, das den Transport der Daten innerhalb eines Netzwerks übernimmt. *TCP* ist ein verbindungsorientiertes Protokoll, d.h. dass vor jeder Übertragung von Daten eine TCP-Verbindung zum Empfänger aufgebaut wird. Das Protokoll stellt die korrekte Übertragung der Daten und deren Reihenfolge sicher.

UDP User Datagram Protocol
UDP ist das verbindungslose Gegenstück zu *TCP*, das im Internet zur Datenübertragung eingesetzt wird. Für eine korrekte Übertragung und Reihenfolge der Daten ist die Applikation verantwortlich. Aufgrund des geringeren Protokoll-Overheads bietet sich *UDP* für schnelle Datenübertragungen an.

VLC Value-Length-Codierung
Bei dieser Codierung wird eine Folge von gleichen Werten als Paar der Länge und des Wertes codiert.

Literaturverzeichnis

[1] A. Vetro and H.Sun, "Media conversion to support mobile users," in *IEEE Canadian Conference on Electronic and Computer Engineering*, May 2001.

[2] Z. M. Mao, H.-S. W. So, and B. Kang, "Network support for mobile multimedia using a self-adaptive distributed proxy," in *11th International workshop on on Network and Operating Systems support for digital audio and video*, pp. 107–116, 2001.

[3] S.-F. Chang and D. G. Messerschmitt, "Manipulation and compositing of MC-DCT compressed video," *IEEE Journal of Selected Areas in Communications*, vol. 13, pp. 1–11, Jan. 1995.

[4] N. Merhav and V. Bhaskaran, "Fast algorithms for DCT-domain image downsampling and for inverse motion compensation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 7, pp. 468–476, June 1997.

[5] H. Li and H. Shi, "A fast algorithm for reconstructing motion-compensated blocks in compressed domain," *Journal of Visual Languages and Computing*, vol. 10, pp. 607–623, Dec. 1999.

[6] R. Dugad and N. Ahuja, "A fast scheme for image size change in the compressed domain," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, pp. 461–474, Apr. 2001.

[7] C.-W. Lin and Y.-R. Lee, "Fast algorithms for DCT-domain video transcoding," in *IEEE International Conference on Image Processing*, pp. 421–424, Oct. 2001.

[8] J. Youn, M.-T. Sun, and C.-W. Lin, "Motion vector refinement for high performance transcoding," *IEEE Transactions on Multimedia*, vol. 1, pp. 30–40, May 1999.

[9] M.-J. Chen, M.-C. Chu, and S.-Y. Lo, "Motion vector composition algorithm for spatial scalability in compressed video," *IEEE Transactions on Consumer Electronics*, vol. 47, pp. 319–325, Aug. 2001.

[10] M.-J. Chen, M.-C. Chu, and C.-W. Pan, "Efficient motion-estimation algorithm for reduced frame-rate video transcoder," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, pp. 269–275, Apr. 2002.

[11] P. A. Assuncao and M. Ghanbari, "A frequency-domain video transcoder for dynamic bit-rate reduction of mpeg-2 nit streams," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, pp. 953–967, Dec. 1998.

[12] B. Shen, I. K. Sethi, and V. Bhaskaran, "Adaptive motion-vector resampling for compressed video downscaling," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 9, pp. 929–936, Sept. 1999.

[13] P. Yin, A. Vetro, H. Sun, and B. Liu, "Drift compensation architectures and techniques for reduced resolution transcoding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, pp. 1009–1020, Nov. 2002.

[14] Z. Lei and N. D. Georganas, "H.236 video transcoding for spatial resolution downscaling," in *IEEE International Conference on Information Technology: Coding ans Computing*, Apr. 2002.

[15] K.-T. Fung, Y.-L. Chan, and W.-C. Siu, "Dynamic frame-skipping for high-performance transcoding," in *IEEE International Conference on Image Processing*, Oct. 2001.

[16] K.-T. Fung, Y.-L. Chan, and W.-C. Siu, "New architecture for dynamic frame-skipping transcoder," *IEEE Transactions on Image Processing*, vol. 11, pp. 886–900, Aug. 2002.

[17] N. Feamster and S. J. Wee, "An MPEG-2 to H.263 transcoder," in *SPIE International Symposium on Voice, Video and Data Communications*, Sept. 1999.

[18] T. Shanableh and M. Ghanbari, "Heterogeneous video transcoding to lower spatio-temporal resolutions and different encoding formats," *IEEE Transactions on Multimedia*, vol. 2, pp. 101–110, June 2000.

[19] ITU-T, "Video codec for audiovisual services at px64 kbit/s." International Telecommunications Union, Telecommunications Standardisation Sector, ITU-T Recommendation H.261, Mar. 1993.

[20] ITU-T, "Video coding for low bit rate communication." International Telecommunications Union, Telecommunications Standardisation Sector, ITU-T Recommendation H.263, Mar. 1996.

[21] ISO/IEC 11172-2, "Information technology – coding of moving pictures and associated audio for digital storage media at up to about 1,5 mbit/s – part 2: Video." MPEG-1, 1993.

[22] ISO/IEC DIS 13818-2, "Information technology – generic coding of moving pictures and associated audio information: Video." MPEG-2, 1994.

[23] ISO/IEC 11172-3, "Information technology – coding of moving pictures and associated audio for digital storage media at up to about 1,5 mbit/s – part 3: Audio." MPEG-1, 1993.

[24] ISO/IEC DIS 13818-3, "Information technology – generic coding of moving pictures and associated audio information: Audio." MPEG-2, 1994.

[25] ISO/IEC DIS 13818-7, "Information technology – generic coding of moving pictures and associated audio information: Advanced audio coding." MPEG-2, 1997.

[26] ISO/IEC 11172-1, "Information technology – coding of moving pictures and associated audio for digital storage media at up to about 1,5 mbit/s – part 1: Systems." MPEG-1, 1993.

[27] ISO/IEC DIS 13818-1, "Information technology – generic coding of moving pictures and associated audio information: Systems." MPEG-2, 1994.

[28] H. Schulzrinne, A. Rao, and R. Lanphier, "Real time streaming protocol (RTSP)." RFC 2326, IETF, Apr. 1998.

[29] M. Handley and V. Jacobson, "Session description protocol (SDP)." RFC 2327, IETF, Apr. 1998.

[30] H.Schulzrinne, S. Casner, R. Frederick, and V. Jacobsen, "RTP: A transport protocol for real-time applications." RFC 1889, IETF, Jan. 1996.

[31] Y.-R. Lee, C.-W. Lin, and C.-C. Kao, "A DCT-domain video transcoder for spatial resolution downconversion," *Lecture Notes in Computer Science*, no. 2314, pp. 207–218, 2002.

[32] P. Yin, M. Wu, and B. Liu, "Video transcoding by reducing spatial resolution," in *IEEE International Conference on Image Processing*, Sept. 2000.

[33] J.-N. Hwang, T.-D. Wu, and C.-W. Lin, "Dynamic frame skipping in video transcoding," in *IEEE Workshop Multimedia Signal*, Dec. 1998.

[34] B. Shen and S. Roy, "A very fast video spatial resolution reduction transcoder," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2002.

[35] Z. Lei and N. D. Georganas, "Accurate bit allocation and rate control for DCT domain video transcoding," in *IEEE Canadian Conference on Electrical & Computer Engineering*, May 2002.

[36] K.-D. Seo, S.-H. Lee, J.-K. Kim, and J.-S. Koh, "Rate control algorithm for fast bit-rate conversion transcoding," *IEEE Transactions on Consumer Electronics*, vol. 46, pp. 1128–1136, Nov. 2000.

[37] H. Sorial, W. E. Lynch, and A. Vincent, "Selective requantisation for transcoding of MPEG compressed video," in *IEEE International Conference on Multimedia and Expo*, July 2000.

[38] "KOM(S) Streaming System: An RTSP/RTP implementation for multimedia systems researchers who want to investigate protocols and operating systems issues." http://komssys.sourceforge.net, 1997–2003.

[39] RealNetworks, "Rtsp proxy kit 2.0." http://www.rtsp.org/2001/proxy/.

[40] Apple Computer, Inc., "Darwin." http://developer.apple.com/darwin/.

[41] Live.com, "Streaming media." http://www.live.com/liveMedia/.

[42] C. Griwodz and M. Zink, "Dynamic data path reconfiguration," in *International Workshop on Multimedia Middleware 2001 at ACM Multimedia*, pp. 72–75, Oct. 2001.

[43] H.-J. Boehm, R. Atkinson, and M. Plass, "Ropes: an alternative to strings," *Software Practice & Experience*, vol. 25, pp. 1315–1330, Dec. 1995.

[44] S. Theiss, "Konzeption und Implementierung eines Multiformat-fähigen Proxycaches für die KOM-VoD-Umgebung." Studienarbeit. Fachbereich Elektrotechnik und Informationstechnik, Technische Universität Darmstadt, June 2001.

[45] H. Schulzrinne, "RTP profile for audio and video conferences with minimal control." RFC 1890, IETF, Jan. 1996.

[46] "The expat xml parser." http://expat.sourceforge.net/.

[47] "KOM(S) Streaming System: Mailing list archive: komssys-develop." http://sf.net/mailarchive/forum.php?thread_id=1825124&forum_id=155, Mar. 2003.

Index