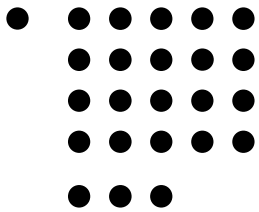

Fachhochschule Köln
University of Applied Sciences Cologne

07 Fakultät für Informations-, Medien- und Elektrotechnik

Studiengang Elektrotechnik

Institut für Nachrichtentechnik
Labor für Datennetze

Diplomarbeit

Thema:	**Entwicklung eines Messsystems zur Analyse von H.264-Videostreams unter Android**
Student :	**Marc Hüffmeyer**
Matrikelnummer:	**11049883**
Referent :	**Prof. Dr.-Ing. Andreas Grebe**
Korreferent :	**Prof. Dr. Carsten Vogt**
Abgabedatum :	**31. Juli 2009**

Hiermit versichere ich, daß ich die Diplomarbeit selbständig angefertigt und keine anderen als die angegebenen und bei Zitaten kenntlich gemachten Quellen und Hilfsmittel benutzt habe.

_____

Marc Hüffmeyer

Inhaltsverzeichnis

1 Einleitung

Moderne Mobiltelefone bieten heutzutage eine Vielzahl von Funktionen, die weit über die grundlegenden Sprachdienste hinausgehen. Sie können als Terminplaner oder auch als Foto- und Videokamera genutzt werden. Die Nutzung dieser Funktionen wurde durch die steigende Leistungsfähigkeit der Geräte möglich. Mit der Entwicklung von UMTS und insbesondere den Übertragungstechniken HSDPA und HSUPA erhält das „mobile Internet“ mehr und mehr Einzug auf mobilen Endgeräten. Aufgrund der hohen Leistungsfähigkeit und des Vorhandenseins der nötigen Infrastruktur können auch Videostreaming-Dienste genutzt werden. Um die Qualität einer Videoübertragung sicherzustellen, ist es notwendig, eine Software zu haben, die die Rahmenbedingung der Übertragung bestimmt und so Rückschlüsse auf die entscheidenen Parameter ermöglicht. In dieser Arbeit soll eine Software entwickelt werden, mit der es möglich ist, die Qualität einer Videoübertragung zu bewerten.
Um eine möglichst hohe Qualität der Videos zu ermöglichen, müssen sie zur Übertragung effizient komprimiert werden. Einer der neuesten Standards zur Komprimierung ist H.264. Dieser Standard bietet eine durchschnittlich dreimal höhere Kompressionsrate als MPEG-2, welches beispielsweise bei der Übertragung von digitalem Fernsehen (DVB) und der Speicherung von Videodaten auf einer DVD zum Einsatz kommt. H.264 ist Teil des MPEG-4 Standards, der sich im Bereich der mobilen Endgeräte schon jetzt durchgesetzt hat. Die Struktur von H.264 wird im zweiten Kapitel dargestellt, der Transport von H.264 über verschiedene Streaming-Protokolle im dritten Kapitel dieser Arbeit. Die zur Bewertung der Qualität verwendeten Parameter werden im vierten Kapitel zusammengefasst.
Die zu erstellende Software soll primär für die Android Plattform entwickelt werden. Jedoch soll bei der Entwicklung auch darauf geachtet werden, dass die Software leicht in eine andere Umgebung portierbar ist. Die Android Plattform ist ein komplettes Softwarepaket für mobile Endgeräte wie Smartphones und Netbooks. Neben einem auf Linux basierenden Betriebssystem liefert Android eine Middleware für die Entwicklung und den Betrieb der Anwendungen. Im fünften Kapitel wird der grundlegende Aufbau und die Entwicklung einer Android Anwendung beschrieben.
Nach der Betrachtung der Grundlagen zur H.264-Videokompression, der verwendeten Streaming-Protokolle und der Entwicklung einer Android Anwendung folgt das sechste Kapitel, welches sich mit dem Softwareentwurf und der Implementierung beschäftigt. Der plattformunabhängige Teil der Software steht hier im Fokus, während dem plattformabhängigen Teil lediglich ein kleiner Teil des Kapitels gewidmet ist.
Die Qualitätssicherung der Software wird im siebten Kapitel beschrieben. Hier erfolgt eine Betrachtung des Einflusses der Software auf die Übertragung. Ebenso werden Methoden zur Überprüfung der Korrektheit erläutert.
Das achte Kapitel zeigt die Auswertung der Messerergebnisse. Im neunten Kapitel wird die entworfene Software dargestellt. Es wird die Bedienung der Software erläutert, sowie die Darstellung der Messergebnisse präsentiert.

2 Der MPEG-4 Standard

2.1 Überblick über den MPEG-4 Standard

MPEG-4 ist ein Standard der Verfahren zur Kompression/Dekompression von Video- und Audiodaten sowie deren Transport und Speicherung beschreibt. Der Standard wurde von der International Organization for Standardization (ISO) und der International Electrotechnical Commision (IEC) entwickelt und ist als ISO/IEC 14496 bekannt. Er wurde im Oktober 1998 fertig gestellt und Anfang 1999 veröffentlicht. Zu Beginn des Jahres 2000 erschien MPEG-4 Version 2, welche nur Erweiterungen und keine Veränderungen gegenüber Version 1 beinhaltet. Der Standard ist in mehrere Teile gegliedert, welche „Parts“ genannt werden. [MPEG4]
MPEG-4 Part 2 Visual thematisiert die Videocodierung. Eine entscheidende Weiterentwicklung gegenüber dem MPEG-2 Standard ist die Einteilung des Bildes in eine Szene. Eine Szene setzt sich aus mehreren Objekten zusammen. Diese Einteilung wurde vorgenommen, um die Komposition von natürlichen (pixelbasierten) und synthetischen (computergenerierten 2D- und 3D-Modellen) Videodaten zu ermöglichen. Setzt sich die Szene aus mehr als einem einzigen rechteckigen Objekt zusammen, ist es nötig, neben den Objekten auch eine Szenenbeschreibung zu codieren. Ein weiteres wichtiges Ziel bei der Entwicklung von MPEG-4 war es, Interaktion mit dem Benutzer zu ermöglichen.
Zeitgleich zur Entwicklung des MPEG-4 Standards wurde in Zusammenarbeit mit der International Telecommunications Union (ITU) ein weiterer Standard zur Videocodierung entwickelt. Dieser wurde in den MPEG-4 Standard eingeliedert als Part 10: Advanced Video Coding (AVC). Die ITU führt den Standard unter der Bezeichnung H.264. H.264 bietet nicht die Vorteile, die die Unterteilung des Bildes in eine Szene mit sich bringt und hat auch keine Möglichkeit zur Interaktion mit dem Benutzer. Der entscheidende Vorteil von H.264 ist die Kompressionsrate, welche bei gleicher Bildqualität deutlich höher ist als bei allen anderen bekannten Standards. Dies führt zu deutlich niedrigeren Bitraten bei der Übertragung und weniger Platzbedarf bei der Speicherung. Deshalb wird der Fokus dieser Arbeit auf H.264 liegen.

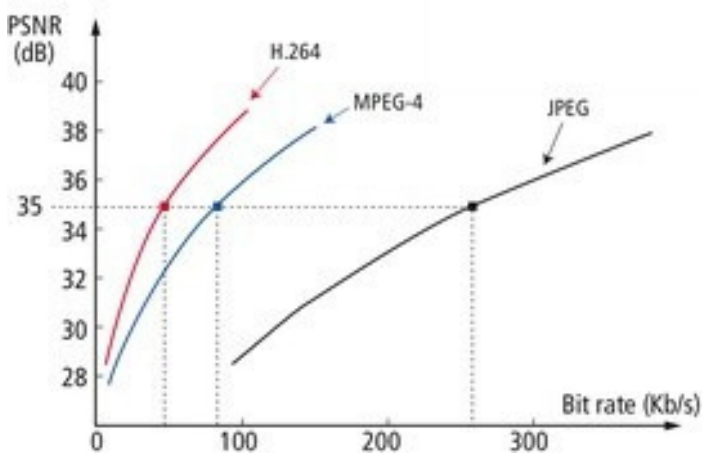

Abb. 1: Vergleich Kompressionsraten [www01]

In MPEG-4 Part 14 wird ein Containerformat für die Speicherung von audiovisuellem Inhalt definiert. Wie dieses Format aufgebaut ist wird in Kapitel 2.3 beschrieben.

2.2 Der H.264 Standard

2.2.1 Kompressionstechniken

Um Videodaten zu komprimieren wird eine Reihe von Verfahren durchlaufen. Die Verfahren sollen an dieser Stelle kurz vorgestellt werden, um einen Überblick über die entscheidenen Parameter bei der Auswertung der Videodaten zu bekommen

2.2.1.1 Inter- und Intra-Prediction

Eine elementare Idee bei der Videokompression ist das Entfernen redundanter Informationen aus einzelnen Bildern und einer ganzen Bildfolge. In H.264 wird dazu zunächst ein Bild in ein oder mehrere Slices unterteilt. Jeder Slice wird dann Inter- oder Intra-codiert.
Bei der Intra-Prediction wird ein einzelner Slice unabhängig von vorherigen oder zukünftigen Bildern komprimiert. Der Slice wird in Blöcke eingeteilt. Diese Blöcke werden in Abhängigkeit von ihrer Umgebung codiert. Das Resultat wird als I-Slice (Intracoded Slice) bezeichnet. Ein I-Slice beinhaltet sämtliche Infomationen, die zum Dekodieren dieses Slices nötig sind.
Bei der Inter-Prediction wird ein Differenzbild aus einem Referenzbild und dem aktuellen Bild gebildet. Dieses Differenzbild beinhaltet deutlich weniger Information als ein Vollbild und kann deshalb auch deutlich stärker komprimiert werden. Allerdings fehlen bei Verlust des Referenzbildes die zum Dekodieren des aktuellen Bildes nötigen Informationen, das Differenzbild ist praktisch also unbrauchbar. Ein Vorteil von H.264 gegenüber anderen Videostandards ist die Möglichkeit, dass ein Differenzbild mehrere Referenzbilder besitzen kann. Dies hat den Vorteil, dass der Verlust eines Referenzbildes nicht so schwer wiegt.

Abb. 2: Intracodiertes Bild [www02]

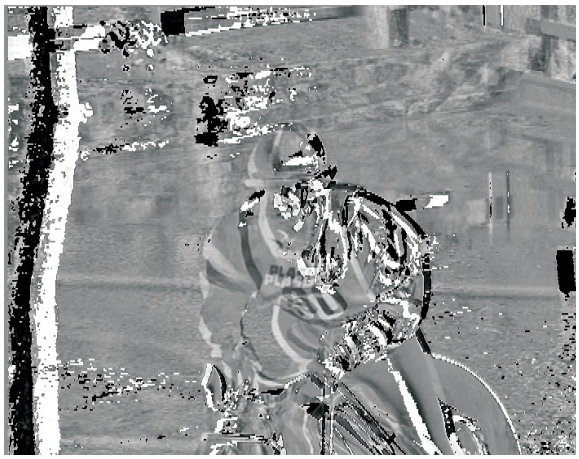

Abb. 3: Intercodiertes Bild [www02]

Slices aus Differenzbildern, welche ausschließlich auf vorherige Bilder referenzieren, werden P-Slices (Predicted Slice) genannt. Referenziert das aktuelle Bild auch auf zukünftige Bilder werden die Slices B-Slices (Bidirectional Predicted Slice) genannt. Damit der Dekoder B-Slices dekodieren kann, muss in diesem Fall das zukünftige Bild vor dem aktuellen Bild übertragen werden, d.h. die Dekodierreihenfolge entspricht nicht der Anzeigereihenfolge.
In anderen Videostandards wird keine Aufteilung in Slices vorgenommen. Entsprechend wird dort unterteilt in I-, P-, B-Frames. Zudem dienen ausschließlich I- und P-Frames als Referenz. H.264 bietet die Option, auch B-Slices als Referenz von anderen B-Slices zu nutzen.
Eine weitere Neuerung in H.264 ist die variable Einteilung der Bildsequenz. In anderen Standards ist die Abfolge von I-, P- und B-Frames durch ein Muster festgelegt, die Group of Pictures (GOP). Eine

GOP beinhaltet immer exakt einen I-Frame, welcher der erste Frame in der Gruppe ist. Danach folgen, in einer durch das Muster festgelegten Reihenfolge, P- und B-Frames. Mit dem nächsten I-Frame startet die nächste GOP.

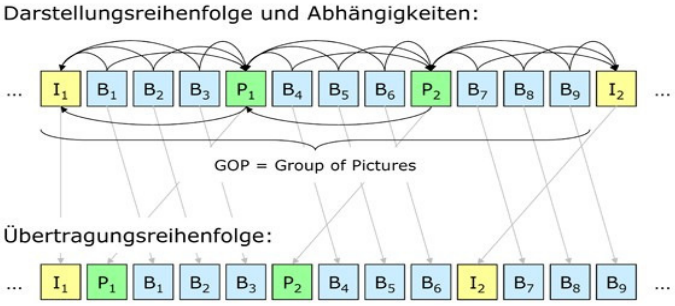

Abb. 4: Group of Pictures [www03]

In H.264 ist die Einteilung der Bildsequenz in eine feste GOP optional, standardmäßig wird sie nicht verwendet. An welcher Stelle ein I-Slice auftritt, ist vom Encoder frei wählbar. Der Vorteil dieses Mechanismus ist die effektivere Nutzung von I-Slices, beispielweise ist es sinnvoll, anstelle jedes 25. Bildes bei jedem Szenenwechsel I-Slices zu verwenden. So können deutlich längere Sequenzen ohne intracodiertes Bild auftreten, was die Kompressionrate erhöht. Der Nachteil von solch langen Sequenzen ohne I-Slice ist die maximale Dauer, die der Decoder benötigt, um sich nach dem Verlust eines Slices wieder zu synchronisieren.
Zudem wurden in H.264 SI- und SP-Slices eingeführt. Diese ermöglichen es, effektiv zwischen verschiedenen Streams zu switchen, ohne eine erhöhte Rate von I-Slices in Kauf nehmen zu müssen. So ist es beispielsweise möglich, dass ein Empfänger bei einem Einbruch der Bandbreite auf einen Stream mit niedrigerer Bitrate umschaltet. Wenn die Bandbreite wieder zur Verfügung steht, kann der Empfänger wieder zurückschalten. SI- und SP-Slices sind jedoch nur im Extended Profile (s. Kapitel 2.2.4) von H.264 verfügbar. Deshalb werden im Folgenden nur I-, P- und B-Slices näher betrachtet.

2.2.1.2 Farbraumtransformation, Luminanz, Chrominanz

Bei der Farbraumtransformation macht man sich eine Eigenschaft des menschlichen Auge zu Nutze. Das Auge ist deutlich empfindlicher für kleine Helligkeitsunterschiede als kleine Farbunterschiede. Diese Eigenschaft lässt sich im RGB-Farbraum nicht nutzen. Deshalb wird der RGB-Farbwert transformiert in einen Helligkeits-(Luminanz) und zwei Farbanteile(Chrominanz).

Abb. 5: YCbCr codiertes Bild [www04]

Abb. 6: Luminanzanteil Y [www04]

Abb. 8: Chrominanzanteil Cr [www04]

Abb. 7: Chrominanzanteil Cb [www04]

Dieses Farbmodell wird YCrCb genannt. Der Y-Wert beschreibt dabei die Helligkeit des Pixels. Cr beschreibt die Abweichung des Farbtons von Grau zu Rot bis Türkis. Cb beschreibt die Abweichung von Grau zu Blau bis Gelb. Durch die höhere Empfindlichkeit für Luminanzunterschiede ist es so möglich, anstelle jedes Farbwerts nur jeden zweiten oder vierten Wert abzutasten ohne spürbare Qualitätsverluste in Kauf zu nehmen. Das Muster, nach dem abgetastet wird, nennt man Chroma Format. In H.264 sind vier Chroma Formate definiert, wobei eines für Schwarz-Weiß Video dient, also keine Chrominanzanteile beinhaltet.

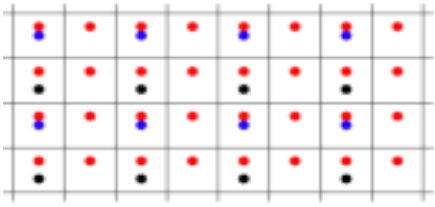

Abb. 9: Chroma Format 4:2:0 [www05]

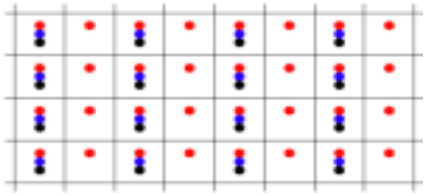

Abb. 10: Chroma Format 4:2:2 [www05]

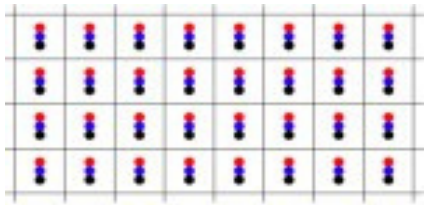

Abb. 11: Chroma Format 4:4:4 [www05]

2.2.1.3 Cosinustransformation/Integertransformation und Quantisierung

Wie ein eindimensionales Signal lässt sich auch ein Bild über seine Frequenzanteile beschreiben. Es wird dazu in Blöcke bestimmter Größe zerlegt. Diese Blöcke werden transformiert. In früheren Standards wurde dazu die Cosinustransformation genutzt, in H.264 wird eine Integertransformation verwendet. Diese ist eine leichte Abwandlung der Cosinustransformation und bietet den Vorteil, dass sie vollständig durch Addition und Shifts implementiert werden kann. Resultat der Transformation ist ein Block derselben Größe wie der Eingangsblock. Die Koeffizienten dieses Blocks beschreiben die im Eingangsblock vorkommenden Frequenzanteile. Durch Überlagerung von Basisvektoren, gewichtet mit dem zugehörigen Koeffizienten, lässt sich der Eingangsblock wieder rekonstruieren.

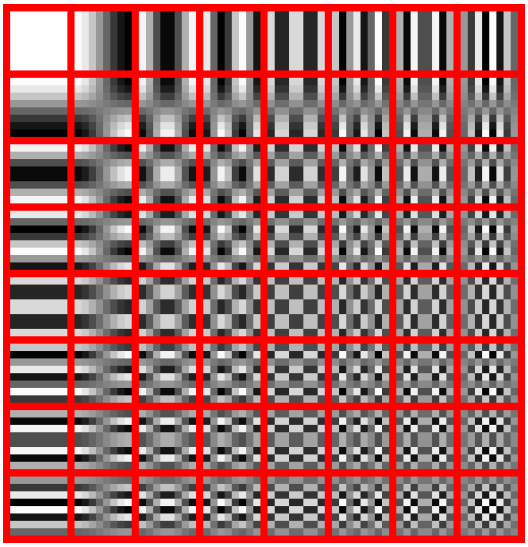

Abb. 12: Basisvektoren einer 8x8 Transformation [www06]

Der entscheidende Vorteil des transformierten Blocks ist die Größe der Koeffizienten. Häufig ist es so, dass viele Koeffizienten gleich Null oder sehr klein sind. Durch Quantisieren der Koeffizienten werden die sehr kleinen Anteile gleich Null gesetzt, ohne bei der Rekonstruktion spürbare Qualitätsverluste in Kauf zu nehmen. Typischerweise stehen die Koeffizienten gleich Null unten rechts im transformierten Block. Die Koeffizienten werden deshalb in einem Zick-Zack-Muster eingelesen, so dass eine lange Reihe von Null-Koeffizienten entsteht, was eine starke Kompression ermöglicht.

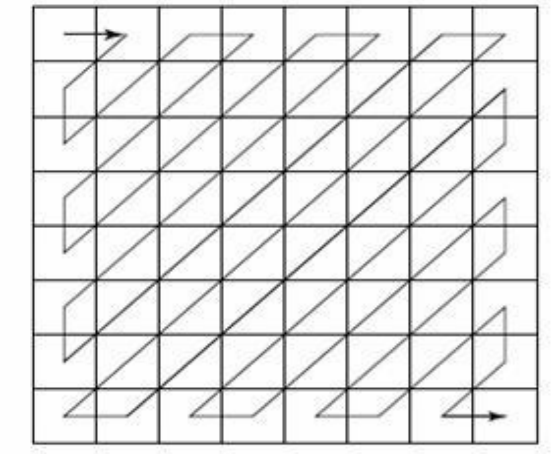

Abb. 13: Zig-Zag-Scan [www07]

2.2.1.4 Entropiecodierung

Um möglichst viele Bits zu sparen, werden Entropiecodierer eingesetzt. Ein Entropiecodierer steigert die Informationsdichte im Bitstrom. Wie dies geschieht, ist abhängig vom eingesetzten Mechanismus des Entropiecodierers. In H.264 werden drei verschiedene Entropiecodierer eingesetzt: Exp-Golomb-Coding, Context-Adaptive-Variable-Length-Coding (CAVLC) und Context-Adaptive-Binary-Arithmetic-Coding (CABAC). Die Exp-Golomb-Codierung wird für alle Daten außer den transformierten und quantisierten Koeffizienten eingesetzt, während CAVLC und CABAC für die Codierung eben dieser eingesetzt werden, also für den Großteil der Datenmenge. CABAC ist effektiver als CAVLC, benötigt aber auch mehr Rechenleistung. Da Mobilfunkgeräte jedoch nur begrenzte Rechenleistung zur Verfügung haben, wird CABAC nicht von Android unterstützt. Im Folgenden sollen deshalb ausschließlich die Exp-Golomb-Codierung und CAVLC betrachtet werden.

2.2.1.4.1 Exp-Golomb-Coding

Die Exp-Golomb-Codierung unterteilt sich wiederum in vier Varianten. Unsigned-Exp-Golomb-Coding (UE) für vorzeichenlose Werte, Signed-Exp-Golomb-Coding (SE) für vorzeichenbehaftete Werte, Truncated-Exp-Golomb-Coding (TE) als eine verkürzte Variante für die Werte 0 und 1 und Mapped-Exp-Golomb-Coding (ME), bei der der errechnete Wert noch einmal über eine Tabelle übersetzt werden muss. Die letzten drei Varianten leiten sich aber sehr einfach von der UE ab.
Die Exp-Golomb-Codierung setzt sich aus zwei Teilen zusammen, Prefix und Suffix. Der Prefix ist eine Folge von Nullen, welche durch genau eine Eins beendet wird. Durch die Anzahl an Nullen (LeadingZeroBits) wird festgelegt, welche Länge der Suffix hat, der binär codiert ist. Der Unsigned-Exp-Golomb-Codierte Wert u berechnet sich dann als:

$$u = 2^{LeadingZeroBits} - 1 + Suffix$$

Bit string form	Range of codeNum
1	0
0 1 x_0	1-2
0 0 1 x_1 x_0	3-6
0 0 0 1 x_2 x_1 x_0	7-14
0 0 0 0 1 x_3 x_2 x_1 x_0	15-30
0 0 0 0 0 1 x_4 x_3 x_2 x_1 x_0	31-62
...	...

Abb. 14: Unsigned-Exp-Golomb-Codierung [H264]

codeNum	syntax element value
0	0
1	1
2	–1
3	2
4	–2
5	3
6	–3
k	$(-1)^{k+1}$ Ceil(k÷2)

Abb. 15: Signed Exp-Golomb-Codierung [H264]

Um einen Signed-Exp-Golomb codierten Wert zu bestimmen ist ein zusätzlicher Schritt nötig. Zunächst wird ein Unsigned-Exp-Golomb-Codierter Wert *u* errechnet. Der Signed-Exp-Golomb codierte Wert *s* bestimmt sich daraus als:

$$s = (-1)^{u+1} ceil(u/2)$$

wobei *ceil(x)* definiert ist als ganzzahliger Wert, der größer oder gleich *x* ist.

Für die Bestimmung eines Truncated-Exp-Golomb codierten Wertes ist die Angabe eines weiteren Parameters notwendig, des Wertebereichs. Ist der Wertebereich gleich Eins, so beschreibt das nächste Bit den invertierten Wert. Ist also das nächste Bit gleich Null, so ist der Wert gleich Eins und

umgekehrt. Ist der Wertebereich größer als Eins, so errechnet sich der Wert analog zur Bestimmung eines Unsigned-Exp-Golomb codierten Wertes.
Für die Bestimmung eines Mapped-Exp-Golomb codierten Wertes wird wiederum zunächst der Unsigned-Exp-Golomb codierte Wert eingelesen. Dieser wird anhand einer Tabelle übersetzt ist einen anderen Wert.

2.2.1.4.2 Context-Adaptive-Variable-Length-Coding

Bei der CAVLC werden Bitfolgen abhängig vom Kontext über verschiedene Tabellen übersetzt. Die Tabellen wurden anhand von Erfahrungswerten erstellt, d.h. es wurden einige Testvideos codiert und die Häufigkeit des Auftretens bestimmter Werte eines Parameters gemessen.
Jeder CAVLC codierte Block ist nach einem bestimmten Muster aufgebaut. Zunächst wird eine Variable nC berechnet. Der Wert von nC ergibt sich durch die Verfügbarkeit der nach oben und nach links benachbarten Blöcke sowie deren Gesamtzahl an Koeffizienten.
Danach wird ein sogenanntes „coeff_token“ aus dem Bitstrom eingelesen. Dieses wird über eine Tabelle in zwei Variablen übersetzt, „TotalCoeff“ und „TrailingOnes“. „TotalCoeff“ gibt die Gesamtzahl an Koeffizienten ungleich Null des Blocks an, „TrailingOnes“ beschreibt die Anzahl an Koeffizienten mit dem Wert 1 oder -1. Es können maximal drei Koeffizienten mit dem Betrag 1 am Ende des Zig-Zag-Scans codiert werden. Anschließend werden entsprechend des Wertes von „TrailingOnes“ 0 bis 3 Vorzeichenbits eingelesen. Die Vorzeichenbits werden im Standard als „trailing_ones_sign_flag“ bezeichnet. Nach den Vorzeichenbits wird die Größe der restlichen Koeffizienten bestimmt. Für jeden Koeffizienten wird dazu zunächst der „level_prefix“ eingelesen. Der „level_prefix“ besteht aus einer Folge von Nullen, die mit einer Eins beendet wird. Die Anzahl von Nullen bestimmt die Länge des „level_suffix“. Durch den „level_suffix“ wird die Größe des Koeffizienten festgelegt. Nachdem die Größe aller Koeffizienten bestimmt wurde, wird der Parameter „total_zeros“ bestimmt. Dieser Parameter gibt die Anzahl an Koeffizienten gleich Null vor dem in Zig-Zag-Scan-Reihenfolge letzten Koeffizienten ungleich Null an. Die Variable „zerosLeft“ wird gleich dem Wert von „total_zeros“ gesetzt. Anschließend wird in einer Schleife die Anzahl an Koeffizienten gleich Null vor einem Koeffizienten ungleich Null eingelesen. Die Anzahl an Koeffizienten gleich Null wird als „run_before“ bezeichnet. Mit jedem Schleifendurchlauf wird der Wert von „zerosLeft“ um den Wert von „run_before“ verringert.
Da die Codierung sehr komplex ist, soll an dieser Stelle ein Beispiel zur Veranschaulichung dienen. Das Beispsiel wurde [www08] entnommen, musste jedoch leicht korrigiert werden. Der gegebene Block soll die Größe 4x4 haben und wird dargestellt durch die Codierung:

00000101 0 1 00001 1111 001 10 0101 00

Zunächst wird angenommen, dass basierend auf der Anzahl an Koeffizienten der benachbarten Blöcke die Variable nC als 1 errechnet wird, d.h. die möglichen Codeworte sind in Anhang C in Abbildung C.1 aufgelistet. Ein Vergleich der gegebenen Bitfolge mit den möglichen Codeworten ergibt, dass der Parameter „coeff_token“ durch die Bitfolge **00000101** repräsentiert wird. Aus Spalte 1 ergibt sich die Variable „TrailingOnes“ zu 2, aus Spalte 2 ergibt sich „TotalCoeff“ zu 4. Insgesamt sind also 4 Koeffizienten ungleich 0, von denen die letzten beiden den absoluten Wert 1 haben. Nun wird das Vorzeichen der letzten beiden Koeffizienten eingelesen. Eine 1 wird als „-“ interpretiert, eine 0 als „+“. Die beiden Koeffizienten haben also entsprechend den Bits **0** und **1** die Werte 1 und -1.
Es folgt der „level_prefix“ eines der verbleibenden Koeffizienten. Das entsprechende Codewort wird gemäß Abbildung C.2 durch die Bits **00001** repräsentiert, „level_prefix“ hat den Wert 4. Der „level_suffix“ wird also durch 4 weitere Bits repräsentiert, **1111**. Der Koeffizient hat den Wert 15, da der „level_suffix“ binär codiert ist. „level_prefix“ und „level_suffix“ werden nach dem gleichen Prinzip für den noch verbleibenden Koeffizient eingelesen. Aus den Bits **001** und **10** ergibt sich der Wert 2.
Als Zwischenergebnis kann festgehalten werden, dass die Koeffizienten die Werte 1, -1, 15, 2 haben. Diese sind in inverser Scan-Reihenfolge angeordnet. Es ist noch zu prüfen, ob zwischen den Koeffizienten ungleich Null Koeffizienten gleich Null liegen. Dies wird angegeben durch den Parameter „total_zeros“. Zur Erinnerung, die Variable „TotalCoeff“ hat den Wert 4, d.h. in Abbildung C.3 stehen die möglichen Codeworte in Spalte 5. Durch Vergleich mit dem Bitstrom folgt, dass „total_zeros“ repräsentiert wird durch die Bitfolge **0101** und somit den Wert 2 hat.

Als letzter Schritt wird nun bestimmt an welcher Stelle die beiden Koeffizienten gleich 0 auftreten. Die Variable „zerosLeft“ hat wie „total_zeros“ zunächst den Wert 2, es gilt also zunächst Spalte 3 in C.4. Im ersten Schleifendurchlauf werden die Bits **00** eingelesen, „run_before“ hat also den Wert 2. „zerosLeft“ wird um 2 verringert, hat nun den Wert 0. Damit sind keine weiteren Schleifendurchläufe nötig, da die Gesamtzahl an Koeffizienten gleich Null erreicht ist. Genau wie die Koeffizienten ungleich Null werden auch die Koeffizienten gleich Null in inverser Scan-Reihenfolge codiert. Die Koeffizienten gleich Null treten also zwischen den Koeffizienten mit Wert 1 und –1 auf. Die Koeffizienten in inverser Scan-Reihenfolge lauten also 1, 0, 0 , -1 ,15 ,2 . In Scan-Reihenfolge 2, 15, -1, 0, 0, 1.

2	15	1	0
-1	0	0	0
0	0	0	0
0	0	0	0

Abb. 16: Transformierter 4x4 Block [www08]

Um zu zeigen wie effektiv der Entropiecodierer arbeitet soll angenommen werden, dass alternativ jeder Koeffizient des Blocks mit einem Byte binär codiert wird. Die Annahme ist bereits sehr klein gewählt, da nur Werte im Bereich von -128 bis 127 für die Koeffizienten möglich wären. Es müssen aber auch deutlich größere Werte für die Koeffizienten codiert werden können.

00000010 00011111 11111111 00000000 00000000 00000001 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

Trotz der stark vereinfachenden Annahme spart der Entropiecodierer in diesem Beispiel über 76% der Bits im Vergleich zur binären Codierung.

2.2.2 Die Datenstruktur

2.2.2.1 Slices, Slice Groups und Macroblöcke

Jedes Bild kann in ein oder mehrere Slice Groups unterteilt werden. Jede Slice Group kann wiederum ein oder mehrere Slices enthalten. Ein Slice ist eine Menge von Macroblöcken, die zur selben Slice Group gehören. Wie sich ein Macroblock zusammensetzt, ist abhängig vom Chroma Format. Er besteht immer aus 16 4x4 Blöcken für Luma Samples. Die Anzahl an 4x4 Blöcken für Chroma Samples ist 8 für Chroma Format 4:2:0, 16 für Chroma Format 4:2:2 und 32 für Chroma Format 4:4:4, wobei jeweils die Hälfte der Blöcke für die Cr Samples benötigt wird und die andere Hälfte für die Cb Samples. Innerhalb eines Slices werden die zugehörigen Macroblöcke in Scan Order angeordnet, also von links nach rechts und von oben nach unten. Durch die Einteilung in Slice Groups können Macroblöcke flexibel gruppiert werden.
Die Aufteilung des Bildes in verschiedene Slices und Slice Groups hat einen einfachen Grund. H.264 wurde für den Transport in verlustbehafteten Medien konzipiert. Jeder Slice wird in einem eigenen Paket transportiert. Geht nun ein Paket, also ein Slice, verloren, geht nur ein Teil des Bildes verloren, anstelle des ganzen Bildes. Verloren gegangene Macroblöcke lassen sich anhand ihrer umgebenden Macroblöcke rekonstruieren, wenn diese zu einem anderen Slice gehören.

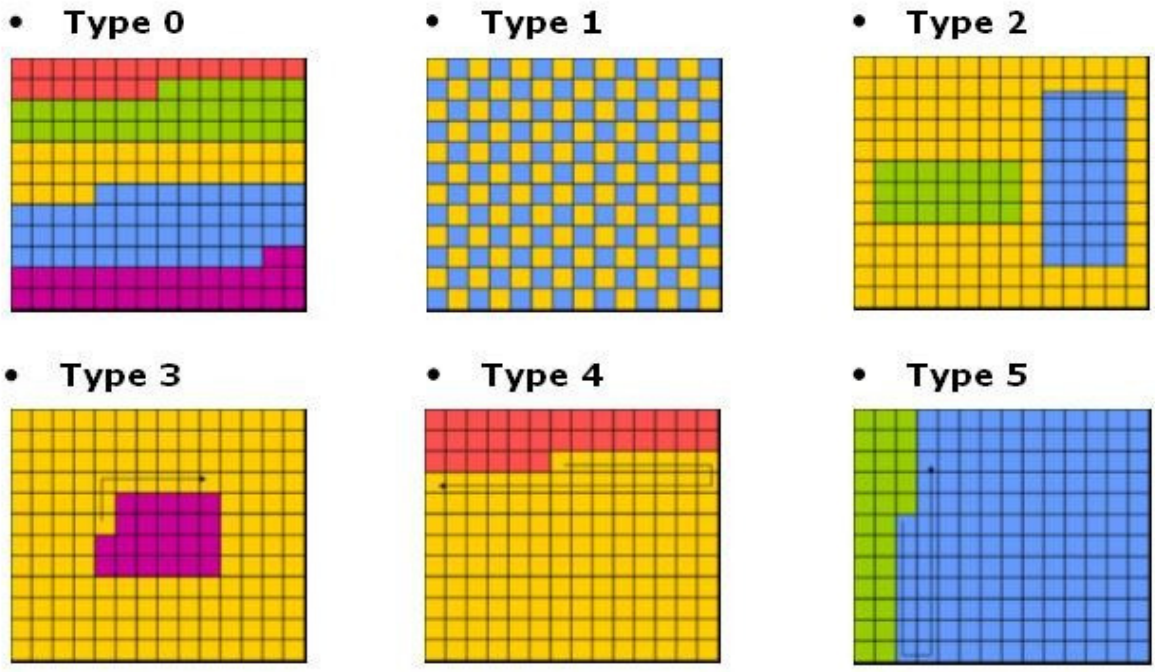

Abb. 17: Slice Group Typen [www09]

Abbildung 17 zeigt die möglichen Slice Group Typen. Die Typen 3-5 sind dynamisch, die Slice Groups wachsen und schrumpfen mit jedem Bild in Richtung der Pfeile. Zudem gibt es noch einen siebten Typen. Dieser ist vollkommen frei definierbar.
Wie schon in Kapitel 2.2.1.1 erwähnt, gibt es verschiedene Typen von Slices, die I-, P-, B-, SI- und SP-Slices. Ebenso gibt es entsprechende Typen von Macroblöcken. Der Slice Typ wird bestimmt durch die Typen von Macroblöcken, die er enthält. Ein I-Slice enthält nur I-Macroblock Typen, ein P-Slice enthält nur I- und P-Macroblock Typen, ein B-Slice nur I- und B-Macroblock-Typen, ein SI-Slice nur SI-Macroblöcke und ein SP-Slice nur SP-Macroblöcke.
Es existieren drei Arten von I-Macroblöcken, I_NxN, I_16x16 und I_PCM. Bei I_NxN wird unterschieden, welche Größe die transformatierten Blöcke haben. Standardmäßig werden in H.264 4x4 Blöcke transformiert, es besteht aber auch die Möglichkeit 8x8 Blöcke zu transformieren. I_NxN Blöcke sind also abhängig vom Transformationsmodus I_4x4 Blöcke oder I_8x8 Blöcke. Da die Transformation in Scan Order vorgenommen wird, stehen zur Berechnung des aktuellen NxN Blocks die NxN Blöcke der Macroblöcke über und links vom aktuellen Macroblock sowie die NxN Blöcke des aktuellen Macroblocks mit niedrigeren Indizes zur Verfügung.

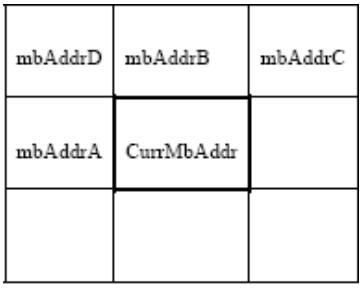

Abb. 18: Scan Order [H264]

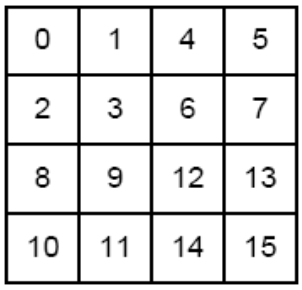

Abb. 19: Macroblockindizes 4x4 [H264]

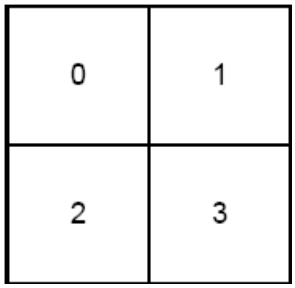

Abb. 20: Macroblockindizes 8x8 [H264]

Abbildung 18 zeigt den aktuellen Macroblock sowie die nach Scan Order bereits codierten Macroblöcke A, B, C und D. Die alphabetische Reihenfolge entspricht dabei nicht der Reihenfolge nach der codiert wird, sondern dient nur als Bezeichnung. Es wird von links nach rechts codiert. Ist das Zeilenende erreicht, wird am Beginn der darunter liegenden Zeile fortgefahren. Abbildung 19 und Abbildung 20 zeigen die 4x4 bzw. 8x8 Blockindizes eines Macroblocks.

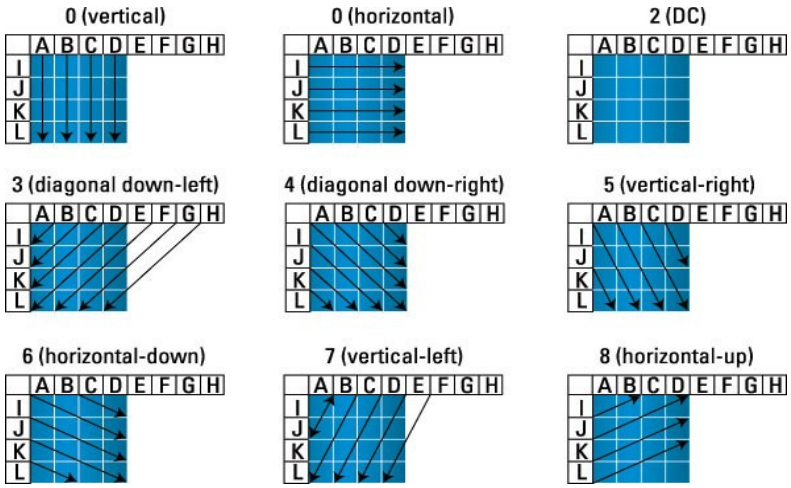

Abb. 21: Intra Prediction Modes 4x4 [www10]

Daraus ergibt sich eine Reihe von Möglichkeiten wie sich die NxN Blöcke berechnen lassen. In Abbildung 21 sind die Möglichkeiten für 4x4 Blöcke dargestellt. Bei der Berechnung von Typ 2 (DC) dient der Mittelwert der Samples A-D und I-L als Referenz für den kompletten NxN Block. Die Abbildungen 22a - 22d zeigen anhand von 8x8 Blöcken beispielhaft, wie sich der aktuelle NxN Block für die Typen 0-3 berechnet.

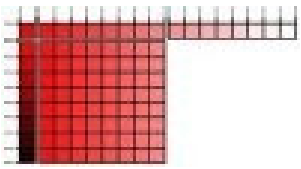

Abb. 22a: Intra Prediction Typ 0 [www11]

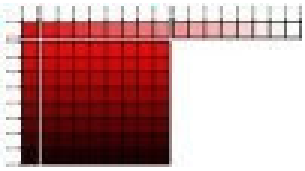

Abb. 22b: Intra Prediction Typ 1 [www11]

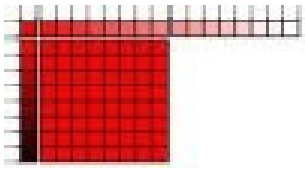

Abb. 22c: Intra Prediction Typ 2 [www11]

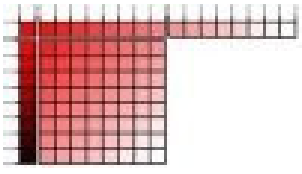

Abb. 22d: Intra Prediction Typ 3 [www11]

Bei der Kodierung eines I_16x16 Blocks werden die unteren 16 Samples des Macroblocks über dem aktuellen Macroblock und/oder die rechten 16 Samples des Macroblocks links des aktuellen

Macroblocks zur Berechnung benutzt. Abbildung 23 zeigt die möglichen Berechnungen. Ebenso wie bei Typ 2 der NxN Berechnung wird auch bei Typ 2 der 16x16 Berechnung der Mittelwert gebildet, allerdings aus 2*16 Samples anstelle von 2*N Samples. Bei Typ 3 wird eine planare Funktion aus den Samples in H und V gebildet.

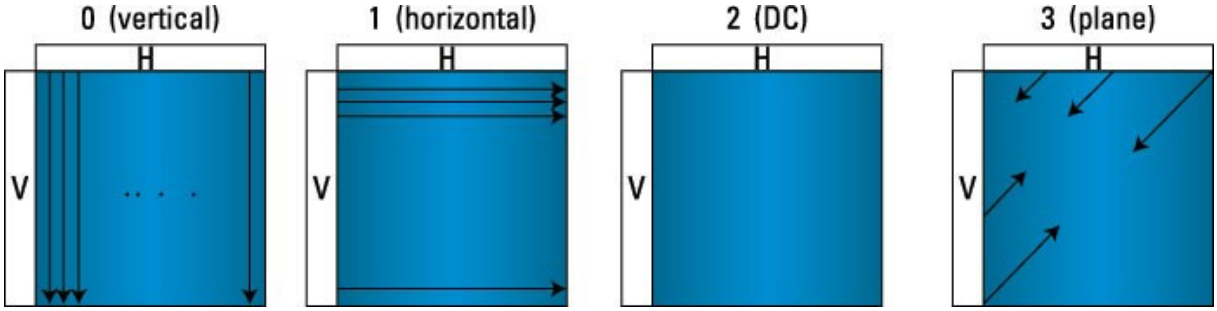

Abb. 23: Intra Prediction Modes 16x16 [www10]

Die dritte Möglichkeit der Macroblock Intra Prediction, I_PCM, erlaubt es dem Encoder, Samplewerte direkt ohne Prediction und Transformation zu codieren.
Bei der Inter-Prediction, also den P- und B-Macroblocktypen, kann die Macroblockebene noch einmal unterteilt werden in Sub-Macroblöcke. Auf Macroblockebene wird zunächst für Luma Samples unterschieden zwischen 16x16-, 16x8-, 8x16- und 8x8-Macroblöcken. Ein Macroblock besteht demnach aus einer 16x16-Partition, zwei 16x8- bzw. 8x16-Partitionen oder vier 8x8-Partitionen. Die 8x8-Partitionen können auf Sub-Macroblöcke der Größe 8x8, 8x4, 4x8 und 4x4 aufgeteilt werden. Eine 8x8 Partition kann wiederum aus einem 8x8-, zwei 8x4- bzw. 4x8-Sub-Macroblöcken oder vier 4x4-Sub-Macroblöcken bestehen. Die Größe der Blöcke für Chroma Samples ergibt sich durch das Chroma Format, für Format 4:4:4 gelten die gleichen Größen wie für die Luma Blöcke, für Format 4:2:2 halbiert sich die Größe in Y-Richtung und für Format 4:2:0 halbieren sich sowohl die Größe in X-Richtung als auch die Größe in Y-Richtung.
Jede Partition und jeder Sub-Macroblock referenziert auf einen Block gleicher Größe und benötigt einen eigenen Bewegungsvektor und einen Referenzindex. Bewegungsvektoren können bis zu 1/4 genau für Luma Samples und 1/8 genau für Chroma Samples angegeben werden. Zeigt der Bewegungsvektor nicht exakt auf einen Abtastwert, wird zwischen den umgebenden Abtastwerten interpoliert. Zusätzlich wurde in H.264 noch ein weiterer Inter Prediction Macroblocktyp eingeführt, der Skip-Macroblock. Ist ein Macroblock von diesem Typ, werden keine weiteren Daten für ihn codiert. Stattdessen errechnet der Decoder aus den Bewegungsvektoren der umgebenden Macroblöcke einen mittleren Bewegungsvektor für den aktuellen Macroblock. Dieses Konzept findet in zwei Situationen Anwendung. Zum einen werden Skip Macroblöcke eingesetzt, wenn sich die Information des Macroblock praktisch nicht ändert, zum anderen werden Sie eingesetzt für globale Bewegungskompensation. Verändert sich die Kameraposition, ist es nicht nötig für jeden Macroblock einen Bewegungsvektor anzugeben, sondern nur für einen einzigen, an dem sich die Anderen orientieren.
Abbildung 24 zeigt ein Bild, welches aus genau einem Slice, also auch einer Slice Group besteht. Der Slice Type ist P. Es können I-Macroblöcke, in der Abbildung rosa markiert, und P-Macroblöcke, grün markiert, vorkommen. Die Größe der Transformationsblöcke ist 4x4, d.h. I_NxN-Blöcke entsprechen I_4x4-Blöcken. Im Bereich der vier Figuren auf der rechten Seite des Bildes kann man erkennen, dass einige Partitionen von P_8x8-Macroblöcken weiter unterteilt wurden in Sub-Macroblöcke. Das Bild zeigt, dass für Regionen mit vielen Details und viel Bewegung kleinere Partionen bzw. Sub-Macroblöcke nötig sind. Macroblöcke und Sub-Macroblöcke geben also einen Aufschluss über den Informationsgehalt eines Bildes.

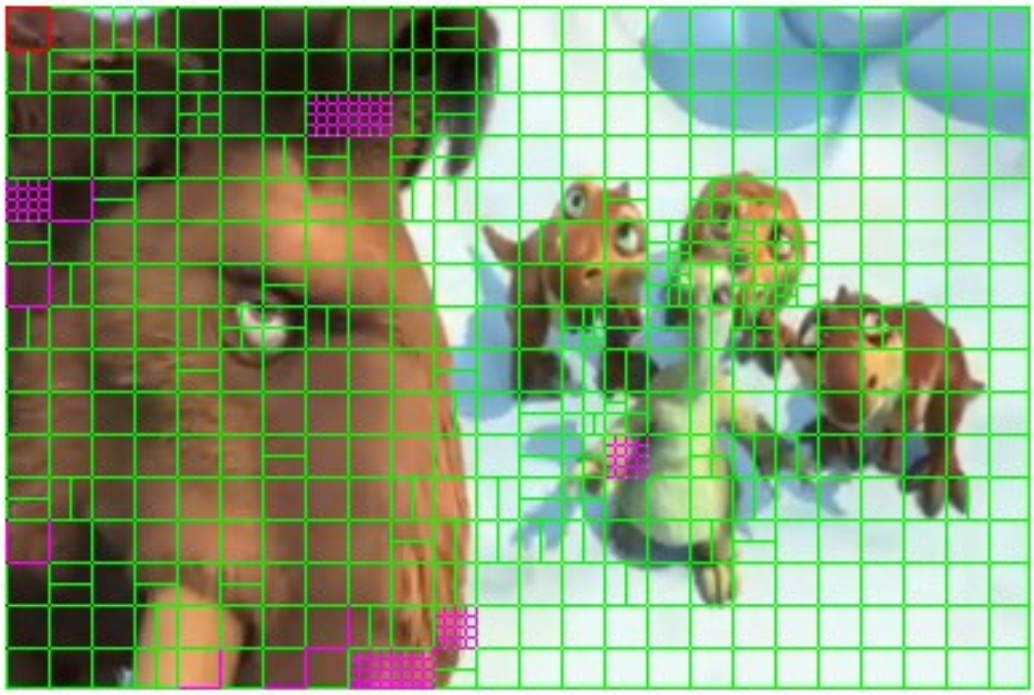

Abb. 24: Macroblocktypen

Abbildung 25 zeigt die Bewegungsvektoren der zugehörigen P-Macroblöcke. Hier kann man gut erkennen, dass für jede Partition und jeden Sub-Macroblock ein eigener Bewegungsvektor nötig ist. Die weißen Macroblöcke unten rechts im Bild sind zum Großteil Macroblöcke vom Typ P-Skip. Die nötigen Informationen über Bewegungsvektoren und Referenzindizes werden aus den umgebenden Macroblöcken entnommen, daher sind die Bewegungsvektoren in diesem Bereich identisch.

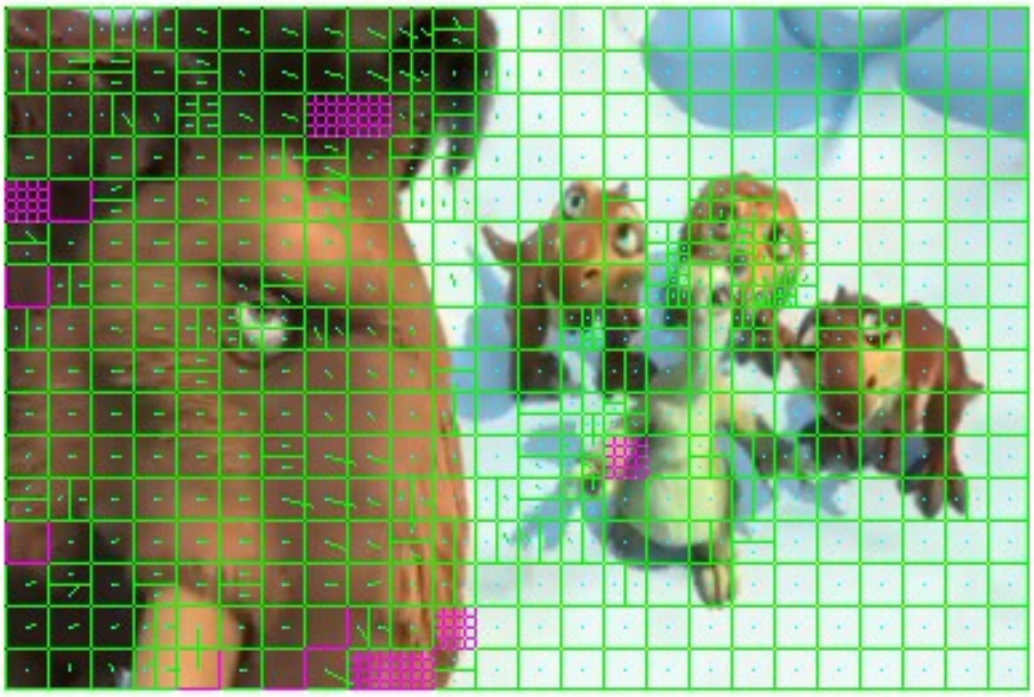

Abb. 25: Bewegungsvektoren

2.2.2.2 Picture Management

In H.264 werden Referenzbilder in zwei Listen, L0 und L1 gespeichert. P-Macroblöcke können nur auf Bilder aus L0 referenzieren, während B-Macroblöcke auf L0 und L1 referenzieren können. Referenziert ein B-Macroblock oder Submacroblock auf beide Listen, so müssen auch zwei Bewegungsvektoren für ihn codiert werden. Die Größe der Referenzlisten wird durch den Parameter „max_num_ref_frames“ bestimmt. Dieser Parameter wird vor der Übertragung des ersten Bildes dem Decoder bekannt gemacht. Wie dies geschieht wird in Kapitel 2.2.2.3 beschrieben. Beide Listen haben die gleiche maximale Größe und „max_num_ref_frames“ gibt die Summe der Größen an. Hat der Parameter beispielweise den Wert 10, können beide Listen jeweils maximal 5 Bilder beinhalten. Das aktuelle Bild referenziert auf die Bilder der Listen über einen Index.
Es gibt zwei Arten von Referenzbildern, Short-Term-Reference-Pictures und Long-Term-Reference-Pictures. Wird im Decoder ein Referenzbild dekodiert wird dieses entsprechend der Dekodierreihenfolge in eine der Listen eingefügt. Dabei wird das älteste Short-Term-Reference-Picture, also das Short-Term-Reference-Picture, welches am längsten in der Liste gespeichert ist, verdrängt. Long-Term-Reference-Pictures können nicht von anderen Referenzbildern aus den Listen L0 und L1 verdrängt werden. Die einzige Möglichkeit ein Long-Term-Reference-Picture aus den Listen zu löschen ist ein Instantaneous Decoding Refresh (IDR). Ein IDR löscht alle Bilder aus den Referenzlisten, d.h. auf einen IDR muss zwingend ein Bild folgen welches ausschließlich aus I-Slices besteht, da keine Bilder als Referenz zur Verfügung stehen. Die Summe aus Short-Term-Reference-Pictures und Long-Term-Reference-Pictures in einer Liste ist immer kleiner oder gleich dem halben Wert von „max_num_ref_frames“. Dieses Verfahren nennt man Sliding Window. Ein Short-Term-Reference-Picture wird in den Listen über seine „frame_num“ identifiziert. Die „frame_num“ ist ein Parameter des Slice Headers, sie erhöht sich mit jedem Referenzbild. Bilder die nicht als Referenz verwendet werden, haben die gleiche „frame_num“ wie das vorige Referenzbild. Ein Long-Term-Reference-Picture wird in den Listen über die „long_term_pic_num“ identifiziert. Auch diese ist ein Parameter des Slice Headers.

2.2.2.3 Parameter Set Konzept

Die Idee des Parameter Set Konzepts ist es, alle Informationen die für mehrere oder gar alle Bilder benötigt werden von den Bildern zu trennen und den Bildern selbst nur eine Referenz auf diese Informationen zu geben. So können die Header-Informationen pro Bild sehr klein gehalten werden. Es gibt zwei Arten von Parameter Sets, Sequence Parameter Sets (SPS) und Picture Parameter Sets (PPS). Picture Parameter Sets beinhalten Informationen, die für ein einzelnes Bild nötig sind, beispielsweise den Slice Group Typ und die Anzahl an Slice Groups oder die Information, ob zur Transformation 4x4-Blöcke oder 8x8-Blöcke verwendet wurden. Jedes PPS hat eine ID. Die Slices referenzieren über diese ID auf das von Ihnen verwendete PPS.

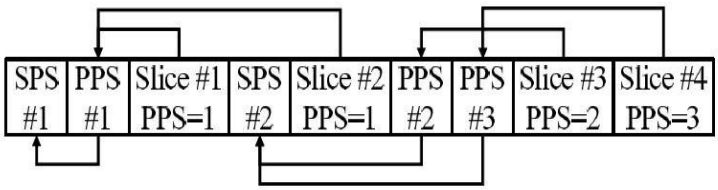

Abb. 26: Parameter Set Konzept

Sequence Parameter Sets beinhalten Informationen die für eine Bildsequenz nötig sind. Hier ist beispielweise angegeben, welches Chroma Format verwendet wird, wie groß die maximale Anzahl an Referenzbildern ist oder in welcher Auflösung die Bilder codiert sind. Auch die SPS haben eine ID. Auf diese wird von den PPS referenziert.
Ohne Parameter Set kann eine Bildsequenz nicht dekodiert werden, d.h. vor einem Slice müssen die

Parameter Sets im Decoder vorliegen. Wenn möglich sollten die Parameter Sets über eine zuverlässige Verbindung ausgetauscht werden.

2.2.3 Die Bitstromstruktur

2.2.3.1 NAL Units

H.264 läßt sich in zwei Ebenen gliedern, den Video Coding Layer (VCL) und den Network Abstraction Layer (NAL). Im VCL werden die Videodaten durch die in Kapitel 2.2.1 und Kapitel 2.2.2 beschriebenen Verfahren codiert. Der daraus resultierende Bitstrom wird an den NAL weiter gereicht, welcher den Bitstrom in NAL Units aufteilt. Jede NAL Unit besteht aus einem Header und der Raw Byte Sequence Payload (RBSP).
Der Header ist ein Byte groß und setzt sich aus drei Parametern zusammmen:

- forbidden_zero_bit, 1 Bit
- nal_ref_idc, 2 Bit
- nal_unit_type, 5 Bit

Das „forbidden_zero_bit“ ist ein Bit zum Füllen des Headers und muss zwingend den Wert Null haben. Der zweite Parameter „nal_ref_idc“ steht für NAL Refererence Indicator. Er gibt an, ob die aktuelle NAL Unit als Referenz benötigt wird oder nicht. Der Wert Null gibt dabei an, dass es sich nicht um eine Referenz handelt, ein Wert ungleich Null gibt an, dass es sich um eine Referenz handelt, die Werte Eins, Zwei und Drei unterscheiden sich dabei nicht weiter. Die übrigen 5 Bit beschreiben den NAL Unit Type. Die RBSP ist abhängig vom NAL Unit Type.
Die möglichen NAL Unit Types zeigt Abbildung 28. Die NAL Unit Types 1-5 werden als VCL NAL Units bezeichnet, die übrigen NAL Units als Non VCL NAL Units. Diese Bezeichnung wurde gewählt, da die Typen 1-5 den vom VCL erzeugten Bitstrom enthalten, also die komprimierte Bildsequenz, während die restlichen Typen Steuerinformationen enthalten.

- NAL Unit Type 1 beinhaltet genau einen Slice eines codierten Bildes, d.h. sowohl den Slice Header als auch die Slice Daten.
- Die NAL Unit Types 2-4 werden für Daten Partitionierung eingesetzt. Daten Partitionierung kann im Extended Profile eingesetzt werden um große Bilder auf verschiedene NAL Units zu verteilen, welche separat transportiert werden. Die Daten Partitionen werden gegliedert in A, B und C. Dem entsprechen die NAL Unit Types 2, 3 und 4. Partition A wird für den Transport des Slice Headers und der Header-Daten der Macroblöcke eingesetzt. Die entsprechenden Parameter sind im H.264-Standard in Kapitel 7 als Category 2 markiert. Partition B wird für den Transport von Intra-Codierten Macroblöcken eingesetzt, Partition C für den Transport von Inter-Codierten Macroblöcken. Die zugehörigen Parameter sind im Standard als Category 3 bzw. Category 4 markiert. Abbildung 27 zeigt einen Ausschnitt eines Syntax-Elements. In der ersten Spalte steht der Name des Parameters oder einer Funktion. Eine Funktion besteht wiederum aus Parametern und Bedingungen. Die zweite Spalte beinhaltet die Category des Parameters. Die dritte Spalte gibt den eingesetzten Entropiecodierer an.

mb_qp_delta	2	se(v) | ae(v)
residual( )	3 | 4	

Abb. 27: Ausschnitt eines Syntax-Elements des H.264-Standards [H264]

- NAL Unit Type 5 beinhaltet einen Slice eines IDR-Bildes. Zur Erinnerung, ein IDR-Bild leert den Speicher für Referenzbilder, den Decoded Picture Buffer (DPB). IDR-Bilder bestehen also vollständig aus I-Slices. Daten Partitionierung kann für IDR-Bilder nicht eingesetzt werden.

nal_unit_type	Content of NAL unit and RBSP syntax structure	C
0	Unspecified	
1	Coded slice of a non-IDR picture slice_layer_without_partitioning_rbsp( )	2, 3, 4
2	Coded slice data partition A slice_data_partition_a_layer_rbsp( )	2
3	Coded slice data partition B slice_data_partition_b_layer_rbsp( )	3
4	Coded slice data partition C slice_data_partition_c_layer_rbsp( )	4
5	Coded slice of an IDR picture slice_layer_without_partitioning_rbsp( )	2, 3
6	Supplemental enhancement information (SEI) sei_rbsp( )	5
7	Sequence parameter set seq_parameter_set_rbsp( )	0
8	Picture parameter set pic_parameter_set_rbsp( )	1
9	Access unit delimiter access_unit_delimiter_rbsp( )	6
10	End of sequence end_of_seq_rbsp( )	7
11	End of stream end_of_stream_rbsp( )	8
12	Filler data filler_data_rbsp( )	9
13	Sequence parameter set extension seq_parameter_set_extension_rbsp( )	10
14..18	Reserved	
19	Coded slice of an auxiliary coded picture without partitioning slice_layer_without_partitioning_rbsp( )	2, 3, 4
20..23	Reserved	
24..31	Unspecified	

Abb. 28: NAL Unit Types [H264]

- Mit NAL Unit Type 6 werden Supplement Enhancement Information (SEI) Messages transportiert. SEI-Messages werden nicht benötigt um den Bitstrom zu dekodieren. Sie stellen Zusatzinformationen dar und können beispielsweise Timestamps oder Informationen über Sub-Sequences enthalten. In Annex D des H.264-Standards kann nachgelesen werden, welche Arten von SEI-Messages existieren.
- Parameter Sets werden in NAL Units vom Type 7 und vom Type 8 transportiert. Das Konzept der Parameter Sets wurde in Kapitel 2.2.2.3 erläutert.
- NAL Unit Type 9 wird als Access Unit Delimiter (AUD) bezeichnet. Im MPEG-4-Standard wird ein Access Unit als kleinste eigenständige Einheit definiert. Für die Videocodierung entspricht eine Access Unit einem Bild. H.264 Access Units bestehen aus genau einem „primary coded picture“, optional einem oder mehreren „redundant coded pictures“ sowie eventuell einem „auxiliary coded picture“.
- Um dem Decoder mitzuteilen, dass die Übertragung einer Bildsequenz bzw. des Streams beendet wird, werden die NAL Unit Types 10 und 11 verwendet.
- NAL Unit Type 12 wird als Filler Data bezeichnet. Als Payload dieses NAL Units dienen eine beliebige Anzahl an Bytes mit dem Wert 0xFF.
- In NAL Units vom Typ 13 werden Sequence-Parameter-Set-Extensions transportiert. Diese sind nicht zwingend notwendig, um die Sequenz zu dekodieren, sondern stellen

Zusatzinformatitionen des Parameter Sets dar.

- NAL Unit Type 19 beschreibt ein „auxiliary coded picture“. Dieses Hilfsbild kann z.B. für Alpha Composition, transparente Bilder mit anderen Bildern als Hintergrund, verwendet werden. Es kann genau wie ein „primary coded picture“ codiert werden.

Innerhalb einer Access Unit gibt es eine vorgeschriebene Reihenfolge in der die NAL Units auftauchen können. Am Anfang eines Access Units steht optional ein AUD. Dieser wird gefolgt von 0 bis N SEI-Messages. Danach muss das „primary coded picture“ folgen, welches in den VCL NAL Units transportiert wird, also den NAL Unit Types 1-5. Optional können mehrere „redundant coded pictures“ folgen, welche in den selben NAL Unit Types wie das „primary coded picture“ transportiert werden. Das letzte codierte Bild ist das „auxiliary coded picture“, wenn vorhanden. Nach den codierten Bildern kann das Ende der Sequenz oder das Ende des Streams signalisiert werden. Parameter Sets können an jeder Stelle auftauchen. Einzige Bedingung für die Position des Parameter Sets ist, dass sie auftauchen, bevor auf sie referenziert wird.

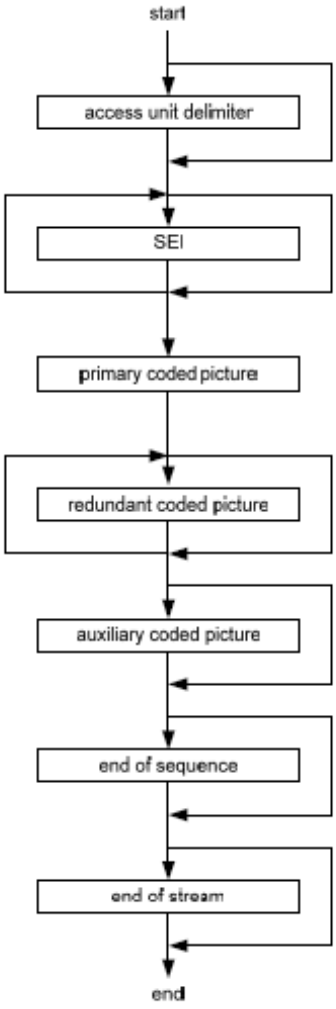

Abb. 29: Struktur einer Access Unit [H264]

2.2.3.2 Aufbau des Bitstroms

Abbildung 30 zeigt die wesentlichen Bestandteile des Bitstroms. Der Bitstrom ist eine Folge von NAL Units, diese können Parameter Sets, codierte Slices oder auch die anderen oben genannten NAL Unit Types beinhalten. Ein codierter Slice setzt sich aus einem Slice Header sowie den Slice Daten

zusammen. Der Slice Header beinhaltet Informationen wie z.B. den Slice-Type, die Bildnummer zu der er gehört und den Index des verwendeten Picture Parameter Sets. Im H.264-Standard kann in Kapitel 7.3.3 nachgelesen werden, wie die genaue Syntax des Slice Headers aussieht. Die Slice Daten beinhalten die Macroblöcke. Die Macroblöcke werden in Scan Order abgearbeitet. Ist der aktuelle Macroblock kein Skip-Macroblock, folgen die Daten der Macroblockebene. Die Syntax der Slice Daten ist im H.264-Standard in Kapitel 7.3.4 beschrieben. Die Macroblockebene beinhaltet Informationen wie den Macroblock-Type und die Macroblock-Prediction. Die Macroblock-Prediction besteht im Falle eines Intra-Codierten-Macroblocks aus den in Kapitel 2.2.2.1 Abbildung 21 und Abbildung 23 dargestellten Typen. Im Falle eines Inter-Codierten-Macroblocks werden hier die Indizes der Referenzbilder aus den in Kapitel 2.2.2.2 beschriebenen Listen L0 und L1 eingefügt. Zudem wird auf Macroblockebene noch ein sogenanntes „coded_block_pattern“ bestimmt. Dies wird entweder codiert oder über den Macroblock-Type bestimmt. Aus dem „coded_block_pattern“ werden zwei Variablen „CodedBlockPatternLuma“ und „CodedBlockPatternChroma“ bestimmt. Sie geben an, welche 8x8 bzw. 4x4 Blöcke transformierte Koeffizienten ungleich Null enthalten. Entsprechend der beiden Variablen sind dann natürlich noch die Koeffizienten selbst enthalten, also die transformierten und quantisierten Samples und Bewegungsvektoren.

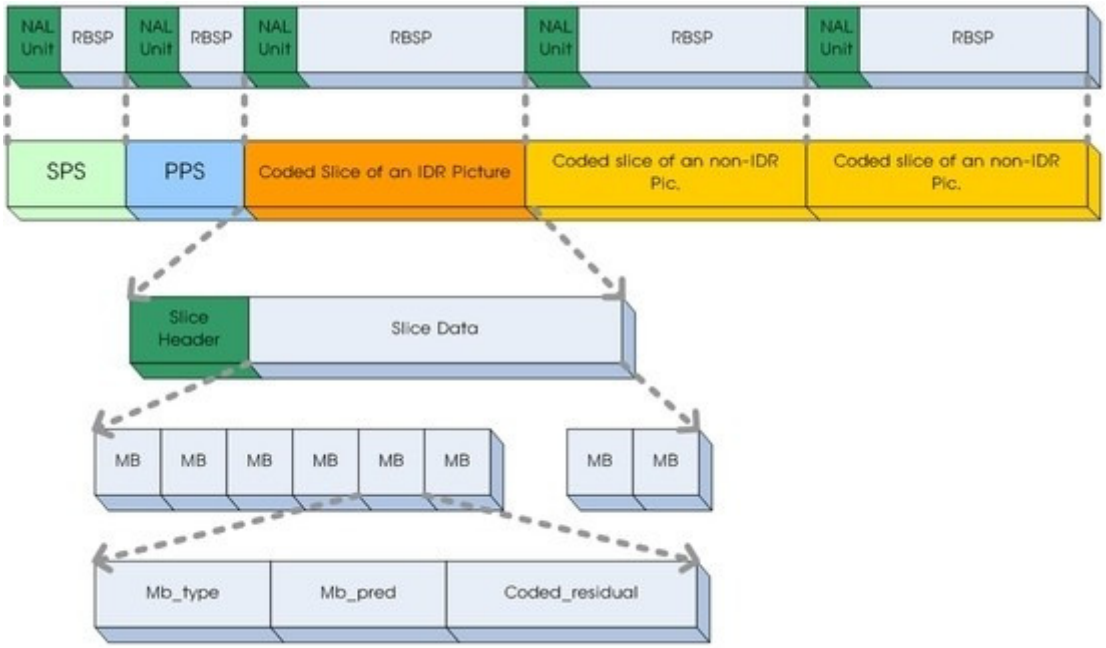

Abb. 30: Aufbau des Bitstroms [www12]

2.2.3.3 Start Code Emulation Prevention

Vor jeder NAL Unit steht im Bitstrom ein Start Code, um den Beginn einer neuen NAL Unit zu markieren. Dieser Start Code besteht aus 24 Bits, 23 Bits mit Wert 0 gefolgt von einem Bit mit dem Wert 1. In hexadezimaler Form also 0x000001. Um zu vermeiden, dass dieser Start Code innerhalb einer NAL-Unit emuliert wird, fügt der Encoder ein Emulation Prevention Byte ein. Dieses Byte hat den hexadezimalen Wert 0x03. Neben dem Start Code 0x000001 sind auch die Bytefolgen 0x000000 und 0x000002 innerhalb einer NAL Unit verboten. Das Emulation Prevention Byte wird vor dem letzten Byte der Bytefolge, welche die verbotenen Bytefolgen emulieren würde, eingefügt. Der Bytestrom könnte also die Bytefolge 0x00000301 enthalten. Um den Bytestrom korrekt zu dekodieren, muss der Decoder das Start Code Emulation Prevention Byte ignorieren.
Ein Beispiel:
Der Encoder erzeugt innerhalb einer NAL Unit die Bytefolge:

... 10110011 00000000 00000000 00000000 000000001 11011110 ...

Das vierte Byte in der dargestellten Bitfolge würde eine verbotene Bitfolge erzeugen, deshalb fügt der Encoder zwischen dem dritten und dem vierten Byte das Emulation Prevention Byte ein.

... 10110011 00000000 00000000 00000011 000000000 00000001 11011110 ...

Der Decoder entdeckt die Bytefolge 0x00000300 und verwirft das Emulation Prevention Byte.

2.2.4 Profile und Level

Der H.264-Standard definiert verschiedene Profile und Level. Einer Bildsequenz wird jeweils genau ein Profil und ein Level zugeordnet. Profile geben an welche Features eingesetzt werden können und werden im SPS durch den Parameter „profile_idc“ angegeben. Level definieren Einschränkungen für Parameter, damit die maximalen Prozessor- und Speicherkapazitäten im Decoder nicht überschritten werden. Der entsprechende Parameter im SPS ist der „level_idc“.

2.2.4.1 Profile

Es existieren sieben Profile, Baseline-, Main-, Extended-, High-, High 10-, High 4:2:2- und High 4:4:4-Profile. Generell kann man sagen, je höher das Profil, desto größer ist der Kompressionsfaktor, aber auch der zum Dekodieren nötige Rechenaufwand. Unter Android wird nur das Baseline-Profile unterstützt.

2.2.4.1.1 Baseline-Profile

Das Baseline-Profile ist optimiert für mobile Endgeräte mit geringen Kapazitäten. Folgende Techniken können deshalb nicht oder nur eingeschränkt genutzt werden:

- Es werden ausschließlich I- und P-Slices unterstützt.
- Daten Partitionierung wird nicht unterstützt, die NAL Unit Types 2 bis 4 dürfen nicht im Bitstrom vorkommen.
- Es werden nur Frames (Vollbilder) unterstützt. Fields (Halbbilder) können nicht verarbeitet werden.
- Es kann ausschließlich das Chroma Format 4:2:0 verwendet werden.
- Die Bittiefe für Luma und Chroma Samples ist 8.
- Weighted Prediction wird nicht unterstützt.
- Als Entropiecodierer wird CAVLC eingesetzt, CABAC wird nicht unterstützt.
- Die Anzahl an Slice Groups liegt zwischen 1 und 8.
- 8x8-Transformation kann nicht eingesetzt werden.
- Der „level_prefix“ darf nicht größer als 15 sein, d.h. die Transformationskoeffizienten können maximal den Wert 2^{15} haben.

Das Baseline-Profile entspricht dem „profile_idc“ 66.

2.2.4.1.2 Weitere Profile

In Annex A des H.264-Standards kann nachgelesen werden, welche Techniken in den weiteren Profilen verwendet werden können. Hier sollen nur die wichtigsten Merkmale reichen.

- B-Slices können in allen Profilen mit Ausnahme des Baseline-Profils eingesetzt werden.
- SI- und SP-Slices sind nur im Extended-Profile verfügbar.
- Die Chroma Formate 4:2:2 und 4:4:4 sind nur in den Profilen High 4:2:2 bzw. High 4:4:4 nutzbar.
- Daten Partionierung ist nur im Extended-Profile möglich.
- CABAC-Entropiecodierung kann in allen Profilen außer Baseline und Extended eingesetzt werden.
- Mehrere Slice Groups können nur im Baseline- und Extended-Profile gebildet werden.

- „redundant coded pictures“ dürfen im Baseline- und im Extended-Profile vorkommen.
- 8x8-Transformation wird nur in den vier High-Profilen unterstützt.

Der „profile_idc“ für das Main-Profile ist 77, für das Extended-Profile 88. Die vier High-Profile werden durch die Werte 100, 110, 122 und 144 repräsentiert.

2.2.4.2 Level

Level definieren Einschränkungen für die Anzahl an Macroblöcken pro Bild und die Anzahl an Macroblöcken pro Sekunde. Zudem wird vorgeschrieben, welche Größe der Decoded Picture Buffer (DPB) und der Coded Picture Buffer (CPB) haben dürfen. Außerdem wird eine maximale Bitrate vorgeschrieben. Diese ist sowohl für den Bitstrom vom VCL als auch für den Bitstrom vom NAL definiert.

Level number	Max macroblock processing rate MaxMBPS (MB/s)	Max frame size MaxFS (MBs)	Max decoded picture buffer size MaxDPB (1024 bytes for 4:2:0)	Max video bit rate MaxBR (1000 bits/s, 1200 bits/s, cpbBrVclFactor bits/s, or cpbBrNalFactor bits/s)	Max CPB size MaxCPB (1000 bits, 1200 bits, cpbBrVclFactor bits, or cpbBrNalFactor bits)	Vertical MV component range MaxVmvR (luma frame samples)	Min compression ratio MinCR	Max number of motion vectors per two consecutive MBs MaxMvsPer2Mb
1	1 485	99	148.5	64	175	[-64,+63.75]	2	-
1b	1 485	99	148.5	128	350	[-64,+63.75]	2	-
1.1	3 000	396	337.5	192	500	[-128,+127.75]	2	-
1.2	6 000	396	891.0	384	1 000	[-128,+127.75]	2	-
1.3	11 880	396	891.0	768	2 000	[-128,+127.75]	2	-
2	11 880	396	891.0	2 000	2 000	[-128,+127.75]	2	-
2.1	19 800	792	1 782.0	4 000	4 000	[-256,+255.75]	2	-
2.2	20 250	1 620	3 037.5	4 000	4 000	[-256,+255.75]	2	-
3	40 500	1 620	3 037.5	10 000	10 000	[-256,+255.75]	2	32
3.1	108 000	3 600	6 750.0	14 000	14 000	[-512,+511.75]	4	16
3.2	216 000	5 120	7 680.0	20 000	20 000	[-512,+511.75]	4	16
4	245 760	8 192	12 288.0	20 000	25 000	[-512,+511.75]	4	16
4.1	245 760	8 192	12 288.0	50 000	62 500	[-512,+511.75]	2	16
4.2	522 240	8 704	13 056.0	50 000	62 500	[-512,+511.75]	2	16
5	589 824	22 080	41 400.0	135 000	135 000	[-512,+511.75]	2	16
5.1	983 040	36 864	69 120.0	240 000	240 000	[-512,+511.75]	2	16

Abb. 31: Level Einschränkungen [H264]

2.3 Das MP4 File Format

Das MP4 File Format wird im MPEG-4 Standard in Part 12 und Part 14 definiert. Es ist ein Containerformat für MPEG-4-Inhalte.

2.3.1 Containerformate

Ein Containerformat beschreibt, wie verschiedene Inhalte innerhalb eines Behälters, des Containers,

gespeichert werden. In der Videocodierung werden in der Regel mindestens eine Videospur und eine Audiospur sowie Metadaten zu diesen Spuren in einem Container gespeichert. Hinzukommen können beispielsweise auch noch weitere Audiospuren, für verschiedene Sprachen, oder Spuren für Untertitel. Die Spuren werden als Tracks bezeichnet.

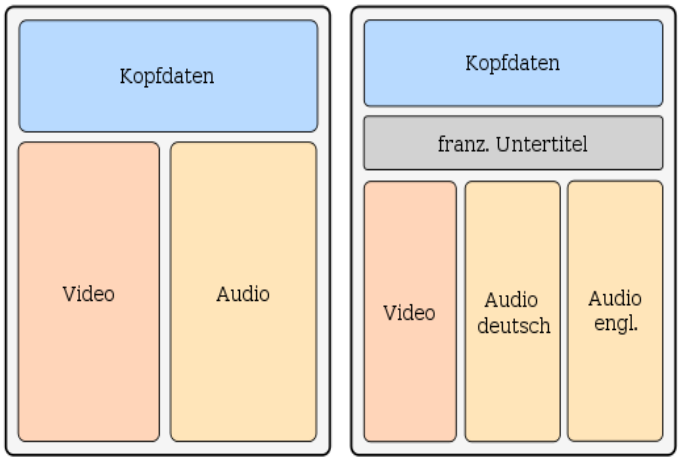

Abb. 32: Container Beispiele [www13]

Damit ein Video streamingfähig ist, wird pro Track ein Hint-Track benötig. Hint-Tracks dienen dazu, dem Streamingserver mitzuteilen, wie die Daten verpackt werden müssen. Für verschiedene Netzwerkprotokolle werden verschiedene Hint-Tracks genutzt.
Das Speichern von verschiedenen Tracks in einem Container nennt man Multiplexen. Dabei werden die Tracks unterteilt in sogenannte Chunks. Ein Chunk setzt sich aus mehreren Samples zusammen. Wie ein Sample definiert ist, hängt von der Art des Tracks ab. Für einen Videotrack entspricht ein Sample je nach Codec einem Frame oder einem Slice. Häufig verwendete Containerformate für H.264-Videotracks sind:

- MP4 (.mp4)
- 3GPP (.3gp)
- Quicktime (.qt)
- Matroska (.mkv)

MP4 stellt den Standard für die Speicherung von H.264 Videodaten dar, da sowohl das MP4 File Format als auch H.264 im MPEG-4 Standard definiert werden. 3GPP ist für mobile Endgeräte optimiert und stellt eine vereinfachte Version von MP4 dar. Quicktime ist ein von der Firma Apple entwickeltes Format, welches die Basis für die Entwicklung des MP4-Containers darstellt. Der Matroska-Container ist ein Open Source Projekt.

2.3.2 Der MP4 Container

Der MP4 Container ist ein hierarchisch aufgebauter Container. Die einzelnen Knoten in der Hierarchie werden Atome genannt, häufig findet man auch die Bezeichnung Boxen. In jedem Atom sind die ersten beiden Parameter der Name des Atoms sowie dessen Größe. An oberster Stelle der Hierarchie stehen das moov-Atom und das mdat-Atom. Im moov-Atom werden sämtliche Metadaten gespeichert, während im mdat-Atoom die Chunks gespeichert werden. Das moov-Atom beinhaltet für jeden Track ein trak-Atom, welches verschiedene Atome für die Metadaten des Tracks enthält. Verpflichtend muss das trak-Atom ein mdia-Atom enthalten, welches wiederum ein minf-Atom enthält. Im minf-Atom werden die Media Informationen gespeichert, d.h. hier ist beschrieben, um was für eine Art von Track es sich handelt. Desweiteren ist hier eine Sample Table enthalten, das stbl-Atom. In diesem Atom werden Informationen über die Chunks gespeichert, z.B. der Offset innerhalb der Datei, die Größe der

Samples oder der Timestamp für das erste Sample des Chunks.

Der in Abbildung 33 gezeigte MP4-Container beinhaltet einen Video-Track (trackId 1), einen Audio-Track (trackId 2) sowie deren Hint-Tracks (trackId 65536 und trackId 65537). Im Zweig des Hint-Tracks mit trackId 65536 findet sich als letztes Atom (letzte Box) das Hint Information Atom. Dieses beinhaltet die RTP Track SDP Hint Information. Hier wird das in Kapitel 3.2.4 beschriebene SDP-Protokoll gespeichert.

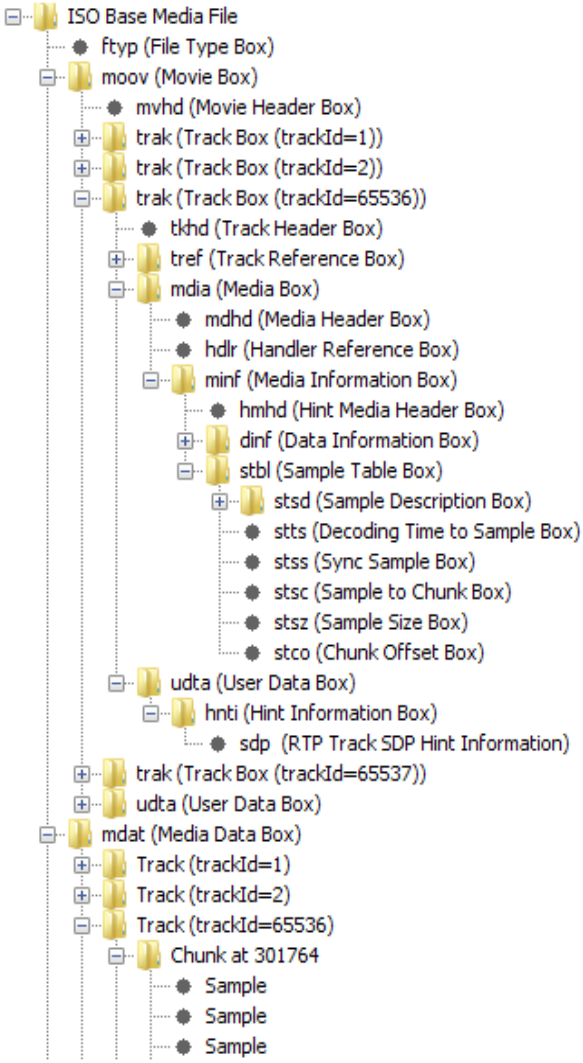

Abb. 33: Hierarchischer Aufbau des MP4-Containers

Die Samples des Hint-Tracks beinhalten Informationen darüber wie die Samples des Video-Tracks in RTP Pakete aufgeteilt werden. In den Samples des Hint-Tracks ist beispielsweise vorgeschrieben, auf welche Anzahl an RTP Paketen ein Sample des Video-Tracks aufgeteilt wird und wie einige Parameter des RTP Headers belegt werden müssen.
Abbildung 34 zeigt, wie der erste Slice des Video-Tracks transportiert wird. In der rechten Spalte kann man erkennen, dass vier RTP Pakete zum Transport des Slices eingesetzt werden. Zudem kann man erkennen, wie einige Parameter des RTP Headers vom ersten RTP Paket belegt werden. Das RTP Protokoll wird in Kapitel 3.2.2 beschrieben.

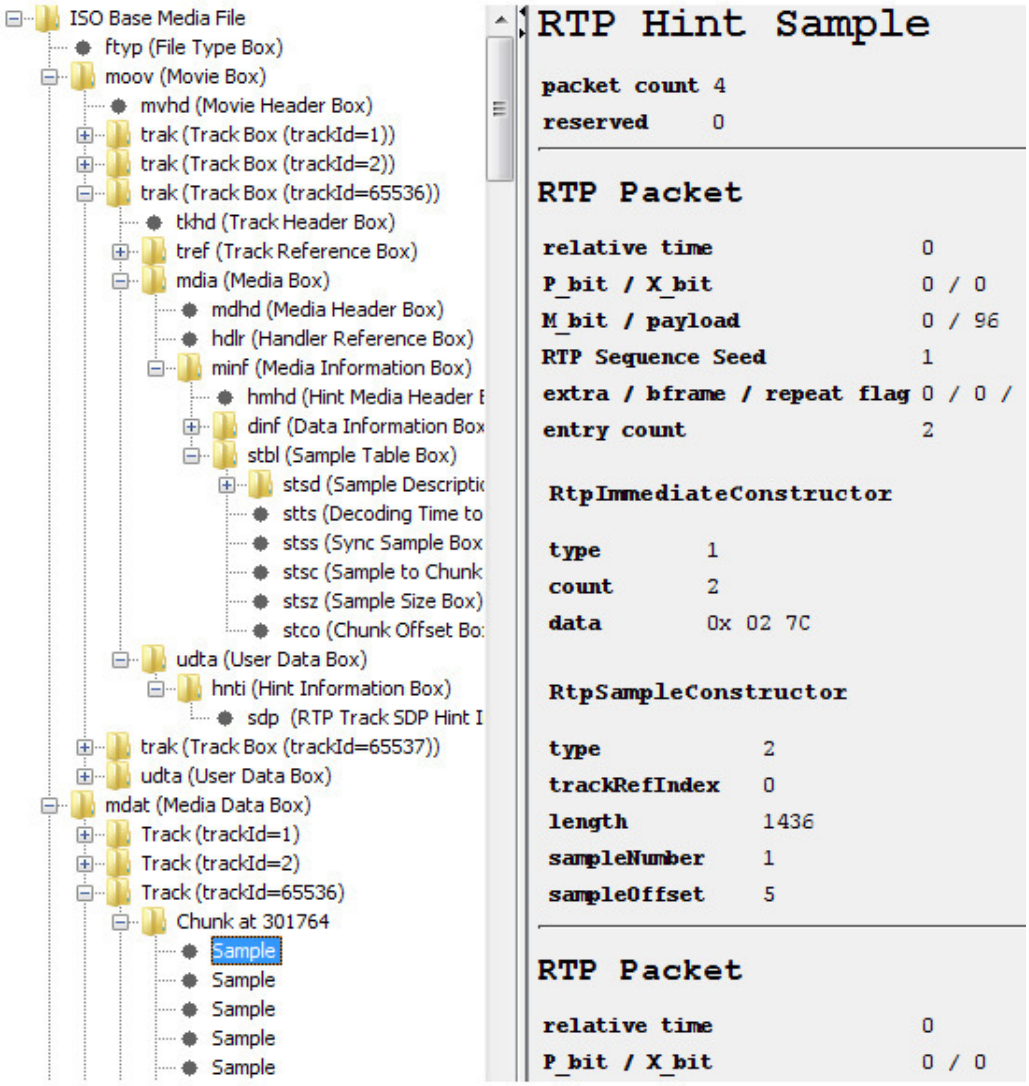

Abb. 34: Hint-Samples im MP4-Container

3 IP-Streaming Protokolle und der Transport von H.264

3.1 Überblick über Streaming und die verwendeten Protokolle

Beim Streamen von Audio- und Videodaten unterscheidet man zwischen Live-Streaming und On-Demand-Streaming. Beim Live-Streaming werden die Daten kontinuierlich als Broadcast versendet, während beim On-Demand-Streaming der Empfänger an den Sender eine Anfrage stellt. Dieser sendet die angefragten Daten dann an den Anfragesteller. Der Empfänger hat hier die Möglichkeit, das Video zu kontrollieren. Er kann also pausieren, vor- und zurückzuspulen oder neustarten.
Welcher Streaming-Mechanismus verwendet wird, ist beim Streamen von H.264-Codierten Videos entscheidend für die Übertragung der in Kapitel 2.2.2.3 beschriebenen Parameter Sets. Diese sollten, wenn möglich, über eine zuverlässige Verbindung übertragen werden. Jedoch wird beim Streaming in der Regel UDP als Transportprotokoll verwendet, welches ein unzuverlässiges Protokoll darstellt. Beim On-Demand-Streaming steht jedoch in der Regel eine TCP-Verbindung, also eine zuverlässige Verbindung, für die Steuerung des Streams zur Verfügung. Besteht eine solche Verbindung, können die Parameter Sets zu Beginn der Übertragung über diese Verbindung gesendet werden. Beim Live-Streaming werden die Parameter Sets periodisch im Bitstrom mitgesendet. Ein H.264-codierter Live-Stream kann also erst dekodiert werden, wenn eine neue Periode beginnt und die benötigten Parameter Sets empfangen wurden.
Gegenstand dieser Arbeit ist On-Demand-Streaming. Android unterstützt Streaming über HTTP und RTSP. Beim Streamen über RTSP werden verschiedene Protokolle eingesetzt. RTSP selbst wird zum Steuern des Streams verwendet. Für den Transport der Audio- und Videodaten wird RTP verwendet. Kontrolldaten können über RTCP versendet werden. Die Parameter Sets werden über SDP transportiert, welches wiederum über RTSP transportiert wird. RTSP, RTP und RTCP können im OSI-Model dem Session Layer zugeordnet werden.

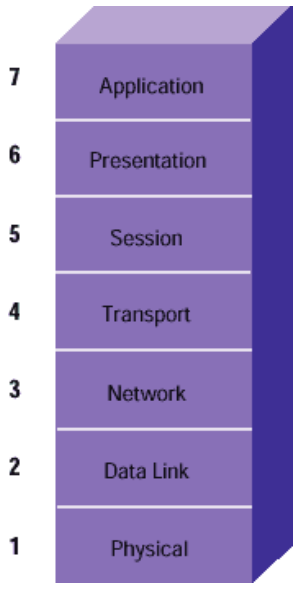

Abb. 35: OSI Referenzmodell [www14]

3.2 IP-Streaming Protokolle

3.2.1 Real Time Streaming Protocol

RTSP wurde 1998 von der Internet Engineering Task Force (IETF) entwickelt und ist in RFC 2326 dokumentiert. Es ist syntaktisch und funktionell sehr ähnlich zu HTTP 1.1, unterscheidet sich aber dennoch in einigen Punkten. So ist RTSP beispielsweise zustandsbehaftet, während HTTP 1.1 zustandslos ist. Ein weiterer wichtiger Unterschied ist der Umstand, dass die zu transportierenden Daten nicht über RTSP selbst, sondern ein anderes Protokoll übertragen werden. Zudem ist RTSP bidirektional, d.h. sowohl Client als auch Server können Anfragen senden, während dies bei HTTP 1.1 dem Client vorbehalten bleibt. Es ist ein textbasiertes Protokoll, dem in der Regel TCP als Transportprotokoll dient, es kann aber auch UDP verwendet werden. [RFC2326]
Die Kommunikation zwischen Client und Server findet über RTSP Messages statt. Dabei wird zwischen verpflichtenden, empfohlenen und optionalen Messages unterschieden. Welche Messages es gibt und in welche Richtung diese gesendet werden, zeigt Abbildung 36.

DESCRIBE	C->S	P,S	recommended
ANNOUNCE	C->S, S->C	P,S	optional
GET_PARAMETER	C->S, S->C	P,S	optional
OPTIONS	C->S, S->C	P,S	required (S->C: optional)
PAUSE	C->S	P,S	recommended
PLAY	C->S	P,S	required
RECORD	C->S	P,S	optional
REDIRECT	S->C	P,S	optional
SETUP	C->S	S	required
SET_PARAMETER	C->S, S->C	P,S	optional
TEARDOWN	C->S	P,S	required

Abb. 36: RTSP Message Types [RFC2326]

Jeder Request wird mit einem Reponse beantwortet. Dieser beinhaltet eine Codenummer und eine zugehörige Beschreibung. Damit der Message-Sender den Response dem abgesetzten Request zuordnen kann, beinhalten sowohl Response als auch Request eine Sequenznummer, die Canonical Sequence Number (CSeq). Diese stimmt in Response und Request überein. Eine einmal verwendete Sequenznummer darf während der Session nicht nocheinmal benutzt werden.

```
Request: OPTIONS rtsp://192.168.1.2/IceAge3_360kb.mp4 RTSP/1.0\r\n
CSeq: 7\r\n
User-Agent: VLC media player (LIVE555 Streaming Media v2008.11.13)\r\n
\r\n
```

Abb. 37: RTSP OPTIONS Request

Die Abbildungen 37 und 38 zeigen einen Request und den zugehörigen Response. Im Request ist der Message Type enthalten, in diesem Beispiel ist der Request vom Typ OPTIONS. Danach folgt die Request-URI und die RTSP Versionsnummer. In der zweiten Zeile steht die CSeq-Nummer. Jede Zeile wird durch die Zeichenfolge \r\n beendet. Die letzte Zeile der RTSP-Message enthält ausschließlich diese Zeichen. Das Ende des RTSP-Protokolls lässt sich also durch die Zeichenfolge \r\n\r\n auffinden.

```
Response: RTSP/1.0 200 OK\r\n
Server: DSS/5.5.5 (Build/489.16; Platform/Linux; Release/Darwin; state/beta; )\r\n
Cseq: 7\r\n
Public: DESCRIBE, SETUP, TEARDOWN, PLAY, PAUSE, OPTIONS, ANNOUNCE, RECORD\r\n
\r\n
```

Abb. 38: RTSP OPTIONS Response

Der Response beinhaltet zunächst die RTSP Versionsnummer. Gefolgt wird diese von der Codenummer und der zugehörigen Beschreibung, im Beispiel ist die Codenummer 200 und die Beschreibung OK.

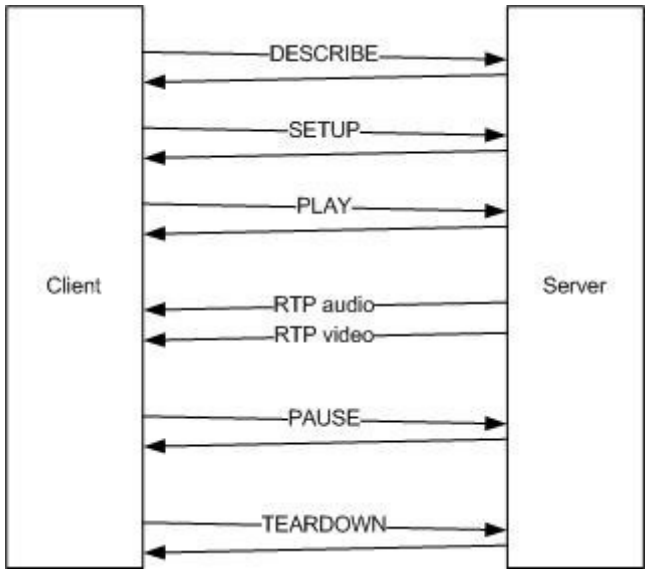

Abb. 39: RTSP Kommunikation [www15]

Abbildung 39 zeigt den typischen Verlauf einer RTSP Session. Zunächst sendet der Client einen OPTIONS-Request. Im zugehörigen Response antwortet der Server mit den zulässigen Message Types, d.h. den Requests, die der Server verarbeiten kann.
Nach dem OPTIONS-Request kann ein DESCRIBE-Request folgen. Der zugehörige Response beinhaltet eine Session Description in Form des SDP-Protokolls. Für den Transport von H.264-codierten Videos sollte der Server den DESCRIBE-Request verarbeiten können, da über die Session Description auch die Parameter Sets transportiert werden können und diese somit über eine zuverlässige Verbindung übertragen würden. SDP wird in Kapitel 3.2.4 beschrieben, der Transport der Parameter Sets über SDP in Kapitel 3.3.3. Abbildung 40 zeigt den DESCRIBE-Request, Abbildung 41 den DESCRIBE-Response.

```
Request: DESCRIBE rtsp://192.168.1.2/IceAge3_360kb.mp4 RTSP/1.0\r\n
CSeq: 8\r\n
Accept: application/sdp\r\n
User-Agent: VLC media player (LIVE555 Streaming Media v2008.11.13)\r\n
\r\n
```

Abb. 40: RTSP DESCRIBE Request

```
Response: RTSP/1.0 200 OK\r\n
Server: DSS/5.5.5 (Build/489.16; Platform/Linux; Release/Darwin; state/beta; )\r\n
Cseq: 8\r\n
Last-Modified: Thu, 07 May 2009 07:31:35 GMT\r\n
Cache-Control: must-revalidate\r\n
Content-length: 731
Date: Sun, 21 Jun 2009 14:08:49 GMT\r\n
Expires: Sun, 21 Jun 2009 14:08:49 GMT\r\n
Content-type: application/sdp
x-Accept-Retransmit: our-retransmit\r\n
x-Accept-Dynamic-Rate: 1\r\n
Content-Base: rtsp://192.168.1.2/IceAge3_360kb.mp4/\r\n
\r\n
Session Description Protocol
```

Abb. 41: RTSP DESCRIBE Response

Anhand der Session Description formuliert der Client pro Track einen SETUP-Request. Innerhalb dieses Request schlägt er zwei Portnummern vor. Die erste Nummer stellt dabei den Port dar, an den die Daten des Tracks gesendet werden. Der zweite Port dient für den Empfang von RTCP-Messages des Tracks. RTCP wird in Kapitel 3.2.3 beschrieben. Zudem schlägt der Client hier vor, welche Protokolle für den Transport eingesetzt werden. Der Server kann die Vorschläge vom Client bestätigen oder selbst neue Vorschläge für obige Parameter machen. Der Response beinhaltet zudem zwei Portnummern des Servers, welche für den Empfang von Daten und RTCP-Messages auf Server Seite dienen. Außerdem legt der Server eine Session ID fest. Diese ID dient der Steuerung des Streams. Sie ist notwendig, um ein Steuerkommando einem Stream zuzuordnen, wenn zwischen Client und Server mehrere Sessions bestehen. Die Session ID ist identisch für jeden Track einer Session.

```
Request: SETUP rtsp://192.168.1.2/IceAge3_360kb.mp4/trackID=65536 RTSP/1.0\r\n
CSeq: 9\r\n
Transport: RTP/AVP;unicast;client_port=65098-65099
User-Agent: VLC media player (LIVE555 Streaming Media v2008.11.13)\r\n
\r\n
```

Abb. 42: RTSP SETUP Request

```
Response: RTSP/1.0 200 OK\r\n
Server: DSS/5.5.5 (Build/489.16; Platform/Linux; Release/Darwin; state/beta; )\r\n
Cseq: 9\r\n
Last-Modified: Thu, 07 May 2009 07:31:35 GMT\r\n
Cache-Control: must-revalidate\r\n
Session: 8694269565801541411
Date: Sun, 21 Jun 2009 14:08:49 GMT\r\n
Expires: Sun, 21 Jun 2009 14:08:49 GMT\r\n
Transport: RTP/AVP;unicast;source=192.168.1.2;client_port=65098-65099;server_port=6970-6971;ssrc=5B44EF4D
\r\n
```

Abb. 43: RTSP SETUP Response

Nach den SETUPs folgt der PLAY-Request. Dieser startet die Übertragung. Der PLAY-Request enthält neben der Session ID ein Zeitintervall. Das Intervall markiert die Zeitpunkte innerhalb des Videos, zwischen denen gesendet werden soll. In der Regel dient die Länge des Videos als Intervall, was durch das Weglassen der oberen Intervallgrenze markiert werden kann. Der PLAY-Response enthält, wenn im SETUP RTP als Protokoll gewählt wurde für jeden angeforderten Track die erste RTP-

Sequenznummer und den ersten RTP-Timestamp. RTP wird in Kapitel 3.2.2 beschrieben.

```
Request: PLAY rtsp://192.168.1.2/IceAge3_360kb.mp4/ RTSP/1.0\r\n
CSeq: 11\r\n
Session: 8694269565801541411
Range: npt=0.000-\r\n
User-Agent: VLC media player (LIVE555 Streaming Media v2008.11.13)\r\n
\r\n
```

Abb. 44: RTSP PLAY Request

```
Response: RTSP/1.0 200 OK\r\n
Server: DSS/5.5.5 (Build/489.16; Platform/Linux; Release/Darwin; state/beta; )\r\n
Cseq: 11\r\n
Session: 8694269565801541411
Range: npt=0.00000-143.91400\r\n
RTP-Info: url=rtsp://192.168.1.2/IceAge3_360kb.mp4/trackID=65536;seq=49633;rtptime=1751189088
\r\n
```

Abb. 45: RTSP PLAY Response

Nach dem Austausch der PLAY-Messages startet die Übertragung der Daten nach dem in den SETUPs definierten Mechanismus. Die Übertragung kann durch einen PAUSE-Request unterbrochen oder einen TEARDOWN-Request beendet werden. Soll die Übertragung nach einem PAUSE-Request fortgesetzt werden, muss erneut ein PLAY-Request gesendet werden. PAUSE- und TEARDOWN-Request entsprechen dem PLAY-Request mit Ausnahme des Zeitintervalls, welches in diesen beiden Requests fehlt. Auch der PAUSE- und TEARDOWN-Response entsprechen dem PLAY-Response, mit Ausnahme des Zeitintervalls und der Information über die RTP-Sequenznummern und die RTP-Timestamps.

3.2.2 Real Time Transport Protocol

RTP wurde erstmals 1996 von der IETF standardisiert und in RFC 1889 dokumentiert. Im Jahr 2003 wurde der Standard von der IETF überarbeitet und in RFC 3550 festgehalten. Mit RFC 3550 wurde RFC 1889 obsolet. RTP dient dem Transport von Echtzeitdaten wie Video-, Audio- oder auch Simulationsdaten. [RFC3550]
Jedes RTP Paket setzt sich aus einem RTP Header und dem RTP Payload zusammen. Jeder RTP Header beinhaltet mindestens einen fixen Anteil, welcher 12 Byte groß ist. Der Header kann aber auch noch erweitert werden. Wie dies geschieht wird im Folgenden erläutert.

Abbildung 46 zeigt den RTP Header. Die CRSC ist ein Teil der Erweiterung des Headers. Die folgenden Parameter gehören zum fixen Anteil des RTP Headers.

Version (V):
Dieser Parameter beschreibt die im RTP Paket verwendete Version. RFC 3550 entspricht Version 2, RFC 1889 entspricht Version 1.

Padding (P):
Das Padding Flag wird gesetzt, wenn ein oder mehrere Bytes am Ende des RTP Pakets zum Füllen genutzt werden. Das letzte Byte des Padding beschreibt, wie viele Bytes als Padding genutzt werden inklusive dem letzten Byte selbst.

Extension (X):
Das Extension Flag wird gesetzt, wenn neben dem fixen Anteil und den CRSCs noch zusätzliche Header Erweiterungen eingesetzt werden. Wie diese Erweiterungen aufgebaut sind, wird weiter unten beschrieben.

CRSC count (CC):
Der CC beinhaltet die Anzahl an CRSCs im Header. Da dieses Feld 4 Bit groß ist, können 0 bis 15 CRSCs im Header transportiert werden.

Marker (M):
Der Einsatz des Marker Bits wird durch Profile bestimmt. Profile sind abhängig vom Payload. So wird die Verwendung des Marker Bits für den Fall, das H.264 transportiert wird, in RFC 3984 – RTP Payload Format for H.264 Video definiert. In diesem Fall wird das Marker Bit nur im letzten RTP Paket einer Access Unit gesetzt.

Payload Type (PT):
Über das PT-Feld lässt sich der Inhalt des RTP Pakets identifizieren. Einige Payload Typen sind in RFC 3551 definiert. Für H.264 gibt es keinen festen PT. Der PT wird entweder dynamisch erzeugt oder im Hint-Track des MP4-Containers festgelegt. Für eine Session bleibt der Payload Typ also konstant.

Sequence Number (sequence number):
Die Sequenznummer wird mit jedem übertragenen RTP Paket erhöht. Die Verwendung der Sequenznummer ist periodisch, d.h. wenn der maximale Wert 65535 erreicht wird, enthält das folgende Paket die Sequenznummer 0. Als Startwert wird zufällig eine Nummer gewählt. Durch Lücken in der Sequenznummer lassen sich Paketverluste entdecken. Bei der Übertragung mehrerer Tracks wird für jeden Track eine eigene Serie von Sequenznummern benutzt.

Timestamp (timestamp):
Der Timestamp markiert den Zeitpunkt, an dem der transportierte Inhalt wiedergegeben werden muss. Er wird in Ticks gezählt und orientiert sich an einer Clock. Die Clock Rate ist abhängig von den transportierten Daten. Für H.264 wird in RFC 3984 eine 90 kHz Clock Rate vorgeschrieben. Der Startwert des Timestamp wird zufällig gewählt.

Synchronization Source Identifier (SSRC):
Der SSRC dient zur Identifikation der Quelle. Er wird zu Beginn der Übertragung zufällig gewählt und kann sich während der Übertragung ändern.

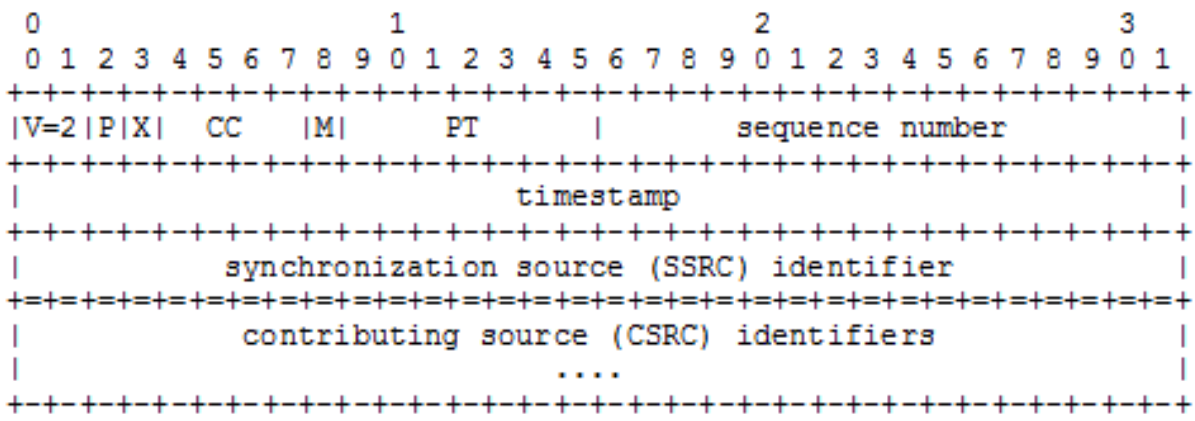

Abb. 46: RTP Header [RFC3550]

Die variablen Anteile des Headers sind zum einen CRSCs und zum anderen kann pro RTP Header

genau eine frei definierbare Extension angehängt werden. Die Extensions dienen dazu, formatabhängige Funktionalitäten zu implementieren, die nicht durch den fixen Anteil abbildbar sind.

Contribution Source Identifier (CRSC):
Wird ein RTP Paket aus mehreren Quellen zusammen gesetzt, können diese Quellen als CRSC im Paket transportiert werden. Mögliche Anwendungen sind Audio- oder Video-Konferenzen. Durch die CRSC ist es möglich, Audio- und Video-Samples einem Teilnehmer zuzuordnen.

Header-Extensions:
Einzige Vorgabe beim Aufbau der Header Extension sind die ersten vier Byte. Die ersten beiden Byte können durch ein Profil definiert werden. Die nächsten beiden Byte geben die Größe der Header Extension in 32-Bit-Worten an. Dabei sind die ersten vier Byte exklusive. Abbildung 47 zeigt die Header Extension.

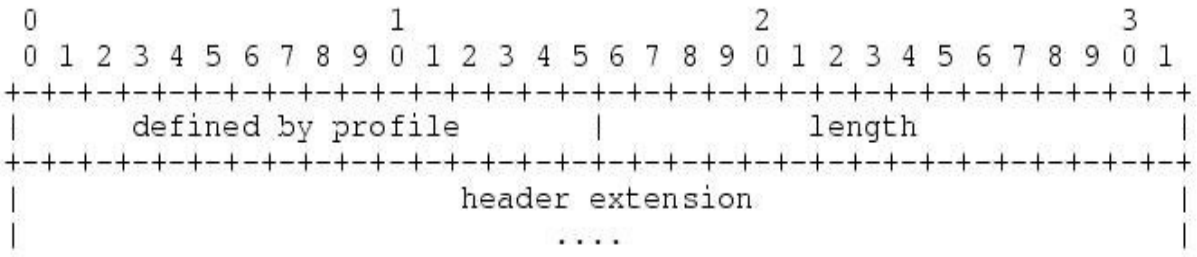

Abb. 47: RTP Header Extension [RFC3550]

3.2.3 RTP Control Protocol

RTCP ist Teil des RTP Standards, d.h. es wurde ebenso in RFC 3550 standardisiert wie RTP. Hauptfunktion von RTCP ist es, ein Feedback über die Qualität der Übertragung zu geben.
Dazu werden in RFC 3550 verschiedene Packet Types definiert, der Sender Report (SR), der Receiver Report (RR), die Source Description (SDES), das Goodbye (BYE) und Application Specific Functions (APP). Ein RTCP Paket muss mindestens zwei dieser Packet Types beinhalten, entweder einen Sender Report oder einen Receiver Report, sowie eine Source Description oder ein Goodbye.

3.2.3.1 Sender Reports und Receiver Reports

Sender Reports können in drei Blöcke unterteilt werden: Header, Sender Info und Report Blocks. Jeder Sender Report beinhaltet genau einen Header, eine Sender Info und optional ein oder mehrere Report Blocks. Ein Sender Report enthält Report Blocks, wenn der Sender gleichzeitig auch Empfänger ist, beispielsweise in einer Videokonferenz. Die Anzahl an Report Blocks entspricht dann der Anzahl an Sendern, von denen RTP Pakete empfangen wurden.
Receiver Reports enthalten Header und Report Blocks. In der Regel ist in einem Receiver Report mindestens ein Report Block. Es kann aber auch ein Receiver Report, der ausschließlich einen Header enthält, versendet werden, wenn eine Session besteht aber keine Daten empfangen werden.
Abbildung 48 zeigt einen Sender Report. Receiver Reports haben den gleichen Aufbau mit Ausnahme der fehlenden Sender Info. Die Felder Version (V) und Padding (P) entsprechen den Feldern des RTP Headers. Der Report Count (RC) gibt die Anzahl an enthaltenen Report Blocks an. Der Packet Type (PT) beschreibt den Typen, ein Sender Report wird durch den Wert 200 signalisiert, ein Receiver Report durch 201, eine Source Description durch 202, ein Goodbye durch 203 und Application Specific Functions durch 204.

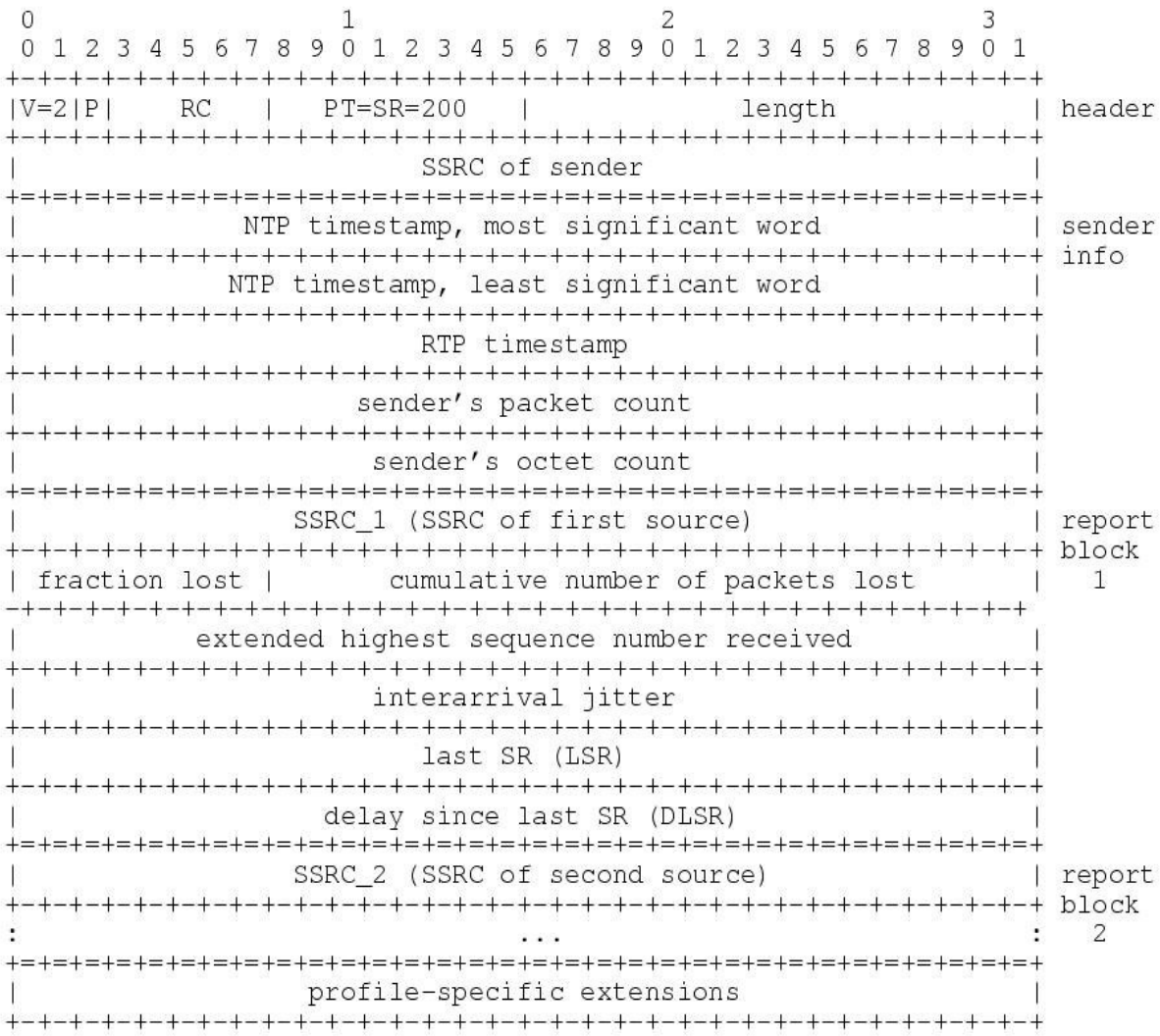

Abb. 48: RTCP Sender Report [RFC 3550]

3.2.3.2 Source Description

Eine Source Description beschreibt die Quellen der RTP Pakete. Sie besteht aus einem Header und optional einem oder mehreren Chunks. Der Header entspricht der ersten Zeile des Headers der Sender Reports mit Ausnahme des Report Count, welcher durch den Source Count ersetzt wird. Der Source Count gibt die Anzahl an enthaltenden Chunks an. Ein Chunk besteht aus einem SSRC/CRSC Identifier und einem von mehreren möglichen SDES-Items. Das wichtigste dieser Items ist der CNAME (Canonical Name). Der CNAME bleibt immer konstant, auch wenn sich der SSRC ändert, und kann so zur Identifikation verwendet werden. Weitere SDES-Items sind zum Beispiel EMAIL oder PHONE.

3.2.3.3 Goodbye

Das Goodbye Packet dient zum Mitteilen, dass eine oder mehrere Quellen nicht mehr zur Verfügung stehen. Neben dem Header, der dem Header der Source Description entspricht, ist in diesem Paket Type nur eine Liste der nicht mehr verfügbaren SSRC/CRSCs enthalten. Die Länge des Liste steht im Source Count. Optional kann noch ein Grund für das Beenden der Session angehängt werden.

3.2.3.4 Application Specific Functions

Die Application Specific Functions Pakete beinhalten wiederum die erste Zeile des Sender Report Headers. Der Report Count wird hier durch das Feld Subtype ersetzt, welches es ermöglichen soll verschiedene Untertypen zu definieren. Danach folgen der SSRC und die applikationsabhängigen Daten.

3.2.4 Session Description Protocol

Das Session Description Protocol wurde ebenfalls von der IETF entwickelt. Es wurde 2006 in RFC 4566 dokumentiert und dient dazu, Multimedia Session zu beschreiben. SDP ist ein textbasiertes Protokoll. [RFC4566]
Jede Zeile des SDP hat die Form:

<type>=<value>

Leerzeichen dürfen weder vor noch nach dem Gleichheitszeichen vorkommen. Die vorkommenden Typen lassen sich in drei Teile gliedern, Session Description, Time Description und Media Description. Es gibt verpflichtende und optionale Typen. Abbildung 49 zeigt die möglichen Typen, die Optionalen sind mit einem * markiert.

```
Session description
   v=  (protocol version)
   o=  (originator and session identifier)
   s=  (session name)
   i=* (session information)
   u=* (URI of description)
   e=* (email address)
   p=* (phone number)
   c=* (connection information -- not required if included in
        all media)
   b=* (zero or more bandwidth information lines)
   One or more time descriptions ("t=" and "r=" lines; see below)
   z=* (time zone adjustments)
   k=* (encryption key)
   a=* (zero or more session attribute lines)
   Zero or more media descriptions

Time description
   t=  (time the session is active)
   r=* (zero or more repeat times)

Media description, if present
   m=  (media name and transport address)
   i=* (media title)
   c=* (connection information -- optional if included at
        session level)
   b=* (zero or more bandwidth information lines)
   k=* (encryption key)
   a=* (zero or more media attribute lines)
```

Abb. 49: SDP Typen [RFC4566]

Man kann erkennen, dass einige Typen sowohl in der Session Description als auch in der Media Description existieren, beispielsweise die Information über die benötigte Bandbreite (b). Treten diese Typen auf, bevor der erste Media Type (m) auftritt, werden sie der Session Description zugeordnet. Nach dem Auftreten des Media Type werden sie der Media Description zugeordnet. Analog wird verfahren, wenn mehrere Media Descriptions im SDP-Protokoll vorkommen.

Für die Beschreibung der Session und der Medien werden Attribut Typen (a) eingesetzt. Attribute können in zwei Formen auftreten:

a=<attribute>
a=<attribute>:<value>

Die erste Form wird genutzt für den Fall, dass das Attribut eine Eigenschaft ist. Bei der zweiten Form kann noch zusätzlich ein Wert zugewiesen werden.
Von besonderem Interesse sind an dieser Stelle der Media Type (m) und die nachfolgenden Attribute (a). Der Media Type beschreibt u.a. um welche Art von Medien es sich handelt, also audio, video, text, etc., und welches Protokoll für den Transport verwendet wird. Im Falle von RTP wird hier auch noch der Wert des Feldes „Payload Type“ des RTP Headers angegeben. Dieser Wert wird in nachfolgenden Attributen zum Mappen von Informationen genutzt. Das Attribut „rtpmap“ ordnet dem Payload Type einen Codec und eine Clock Rate zu. Die Clock Rate entspricht dabei der Clock Rate, die bei RTP für die Berechnung der Timestamps dient. Im Attribut „fmtp“ können für diesen Codec spezifische Informationen übermittelt werden. Ist der Codec H.264, werden hier u.a. Parameter Sets transportiert.

```
Session Description Protocol Version (v): 0
Owner/Creator, Session Id (o): StreamingServer 3454729539 1241681495000 IN I
Session Name (s): /IceAge3_360kb.mp4
URI of Description (u): http:///
E-mail Address (e): admin@
Connection Information (c): IN IP4 0.0.0.0
Bandwidth Information (b): AS:354
Time Description, active time (t): 0 0
Session Attribute (a): control:*
Session Attribute (a): x-copyright: MP4/3GP File hinted with GPAC 0.4.2 (C)2
Session Attribute (a): range:npt=0- 143.91400
Media Description, name and address (m): video 0 RTP/AVP 96
Bandwidth Information (b): AS:290
Media Attribute (a): rtpmap:96 H264/90000
Media Attribute (a): control:trackID=65536
Media Attribute (a): fmtp:96 profile-level-id=42C01E; packetization-mode=1;
Media Attribute (a): framesize:96 384-256
Media Description, name and address (m): audio 0 RTP/AVP 97
Bandwidth Information (b): AS:64
Media Attribute (a): rtpmap:97 mpeg4-generic/48000/2
Media Attribute (a): control:trackID=65537
Media Attribute (a): fmtp:97 profile-level-id=41; config=1190; streamType=5;
```

Abb. 50: SDP Beispiel

Abbildung 50 zeigt ein SDP mit zwei Media Descriptions, die erste für einen Video Track, die zweite für einen Audio Track. Für den Video Track wird in RTP Payload Type 96 verwendet, der Codec ist H.264 und die RTP Clock Rate ist 90kHz. Auf die Darstellung der Parameter Sets wurde hier aus Gründen der Übersichtlichkeit verzichtet.

3.3 Transport von H.264 Video über RTP

Der Transport von H.264 Video über RTP wurde von der IETF standardisiert und in RFC 3984 – RTP Payload Format for H.264 Video im Jahr 2005 dokumentiert. RFC 3984 beschreibt die Fragmentierung und Aggregation von NAL Units, sowie deren Kapselung in RTP Pakete. [RFC3984]

3.3.1 Packetization Modes

In RFC 3984 werden verschiedene Pakettypen für die Fragmentierung, Aggregation und den Transport von einzelnen NAL Units definiert. Welche Pakettypen zum Einsatz kommen wird im Packetization Mode festgelegt. Es existieren insgesamt drei Packetization Modes: Single Unit Mode, Non-Interleaved Mode und Interleaved Mode.
Der Single Unit Mode wird eingesetzt in Konversationssystemen, die dem H.241 Standard der ITU entsprechen. Primäre Funktion sind Video Telephony Systeme, die H.264 als Codec verwenden möchten. Dieser Modus muss von jedem Empfänger unterstützt werden. Der Non-Interleaved Mode wird eingesetzt für Systeme, die nicht dem H.241 Standard entsprechen. Die Unterstützung dieses Modus durch den Empfänger wird von der IETF empfohlen. Die ersten beiden Modi werden genutzt in Systemen mit geringer Ende-zu-Ende-Verzögerung, während der Interleaved Mode in Systemen, die keine geringe Ende-zu-Ende-Verzögerung benötigen, eingesetzt werden kann. Die Unterstützung des Interleaved Mode durch den Empfänger ist optional.
Im Single Unit Mode und im Non-Interleaved Mode müssen Pakete in Decoding Order versendet werden, d.h. die RTP-Sequenznummer beschreibt die Decoding Order. Im Interleaved Mode können Pakete auch in einer Reihenfolge, die nicht der Decoding Order entspricht, versendet werden. Damit der Empfänger die Decoding Order wieder rekonstruieren kann, wird in den im Interleaved Mode erlaubten Pakettypen eine Decoding Order Number (DON) mitgesendet.

Type	Packet	Single NAL Unit Mode	Non-Interleaved Mode	Interleaved Mode
0	undefined	ig	ig	ig
1-23	NAL unit	yes	yes	no
24	STAP-A	no	yes	no
25	STAP-B	no	no	yes
26	MTAP16	no	no	yes
27	MTAP24	no	no	yes
28	FU-A	no	yes	yes
29	FU-B	no	no	yes
30-31	undefined	ig	ig	ig

Abb. 51: Packetization Modes und unterstützte Pakettypen [RFC3984]

Abbildung 51 zeigt, welche Pakettypen im jeweiligen Packetization Mode unterstützt werden. Die Kennzeichnung des Pakettypen geschieht über den Type, welcher in einem für den Pakettypen spezifischen Header steht. Die Header sind dem NAL Unit Header sehr ähnlich.
Der verwendete Packetization Mode kann als Codec spezifischer Parameter im fmtp Attribut von SDP übertragen werden (s. Abb. 50).

3.3.2 RTP Payload Format

Der vom VCL erzeugte Bytestrom enthält Start Codes, um die einzelnen NAL Units voneinander zu trennen. Die Start Codes werden beim Transport verworfen, da sie durch das RTP Payload Format obsolet sind.

3.3.2.1 Single NAL Unit Packet

Ein Single NAL Unit Packet enthält genau eine NAL Unit. Der Type des Single NAL Unit Packets kann die Werte von 1 bis 23 annehmen und entspricht dem NAL Unit Type der transportierten NAL Unit. Neben der NAL Unit werden keine weiteren Daten zum Transport benötigt, da zur Identifikation des Pakettypen der NAL Unit Header dient, welcher den NAL Unit Type enthält.

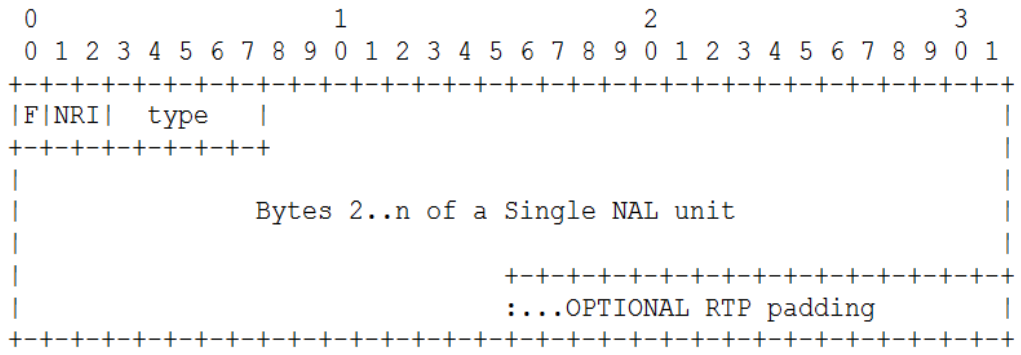

Abb. 52: Singe NAL Unit Packet [RFC3984]

3.3.2.2 Single Time Aggregation Packets

Single Time Aggregation Packets (STAP) können genutzt werden, um NAL Units mit der gleichen NAL Unit Zeit, also dem gleichen Decoding Timestamp, in einem RTP Paket zu transportieren. Dies können beispielsweise die NAL Units einer Access Units sein. Es existieren zwei Arten von STAPs: STAP-A und STAP-B. STAP-A wird im Non-Interleaved Mode eingesetzt, STAP-B im Interleaved Mode.

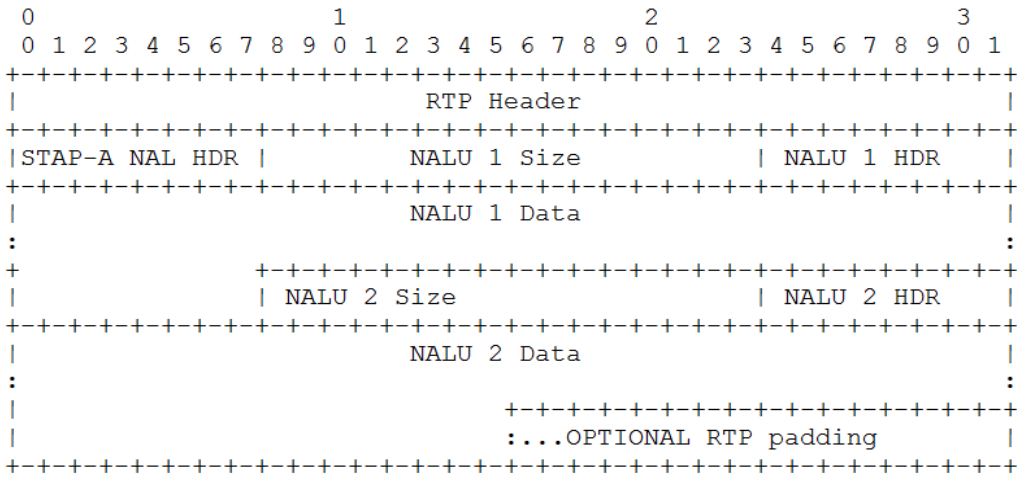

Abb. 53: Single Time Aggregation Packet A [RFC3984]

Abbildung 53 zeigt ein STAP-A Paket mit zwei NAL Units. Nach dem RTP Header folgt ein STAP-A Header. Dieser entspricht vom Aufbau dem NAL Unit Header. Das Forbidden Bit (F) muss auch im STAP-A Header immer den Wert 0 haben. Der NAL Reference Indicator (NRI) des STAP-A Headers entspricht dem höchsten vorkommenden NAL Reference Indicator der im Paket transportierten NAL Units. Der Type des STAP-A Headers ist 23. Die Angabe der NAL Unit Size ist nötig, da beim Transport keine Start Codes verwendet werden.
STAP-B Pakete und Header sind analog aufgebaut. Einziger Unterschied ist die Angabe einer Decoding Order Number pro RTP Paket, da STAB-B für den Transport im Interleaved Mode dient. Die DON wird hinter dem STAP-B Header eingefügt. Sowohl beim Transport von STAP-A Paketen als auch beim Transport von STAP-B Paketen werden die NAL Units innerhalb des Pakets in Decoding Order angeordnet, deshalb reicht die Angabe einer einzelnen DON pro STAP-B Paket.

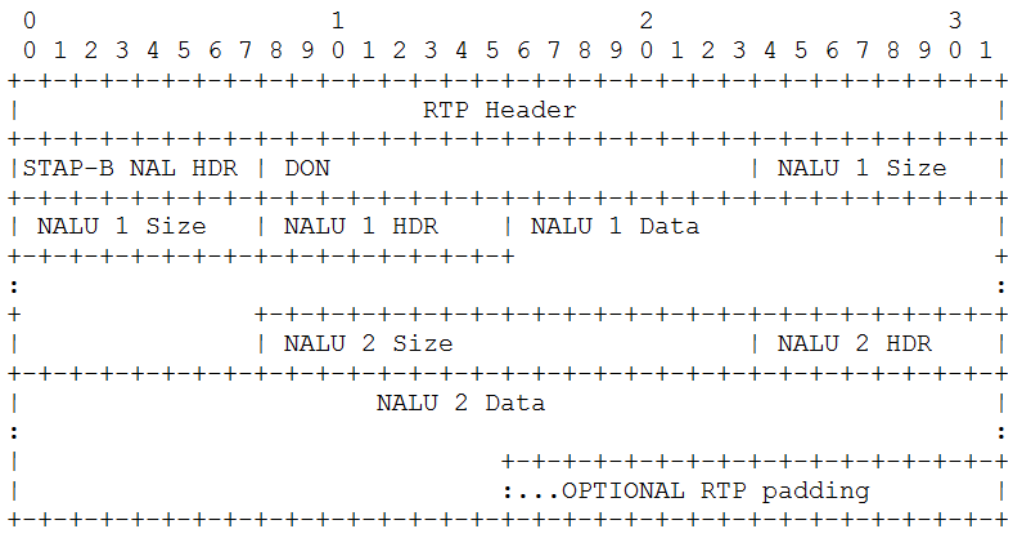

Abb. 54: Single Time Aggregation Packet B [RFC3984]

3.3.2.3 Multi Time Aggregation Packets

In Multi Time Aggregation Packets (MTAP) können Pakete mit verschiedenen Dekodierzeiten transportiert werden. Der Timestamp des RTP Pakets entspricht dem Timestamp der frühesten NAL Unit. Jede NAL Unit wird mit einem Offset zu diesem versehen. Zudem müssen die NAL Units innerhalb eines MTAP nicht in Decoding Order sortiert sein. Deshalb wird eine Decoding Order Number Base (DONB) im RTP Payload eingefügt und jede NAL Unit mit einer Decoding Order Number Difference (DOND) versehen.

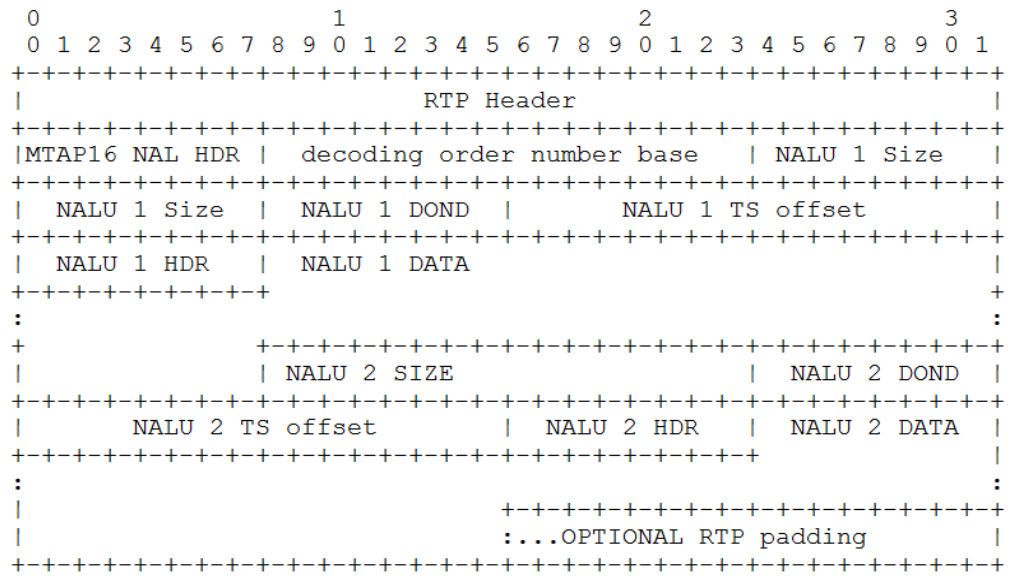

Abb. 55: Multi Time Aggregation Packet 16 [RFC3984]

Auch MTAP Pakete existieren in zwei Varianten: MTAP16 und MTAP24. Der Unterschied zwischen den beiden Varianten ist die Größe des Feldes für den Timestamp Offset, welches 16 Bit bzw. 24 Bit groß ist. MTAP16 und MTAP24 Header entsprechen dem Header für STAP-A und STAP-B.

3.3.2.4 Fragmentation Units

Große NAL Units können zum Transport fragmentiert und auf verschiedene RTP Pakete verteilt werden. Genau wie bei den Single Time Aggregation Packets existieren auch für Fragmentation Units (FU) zwei Varianten für den Transport im Non-Interleaved Mode bzw. dem Interleaved Mode: FU-A und FU-B. Diese unterscheiden sich wiederum nur durch die DON. Werden FU-B Pakete zum Transport eingesetzt, muss die DON für jedes Fragment den gleichen Wert haben.

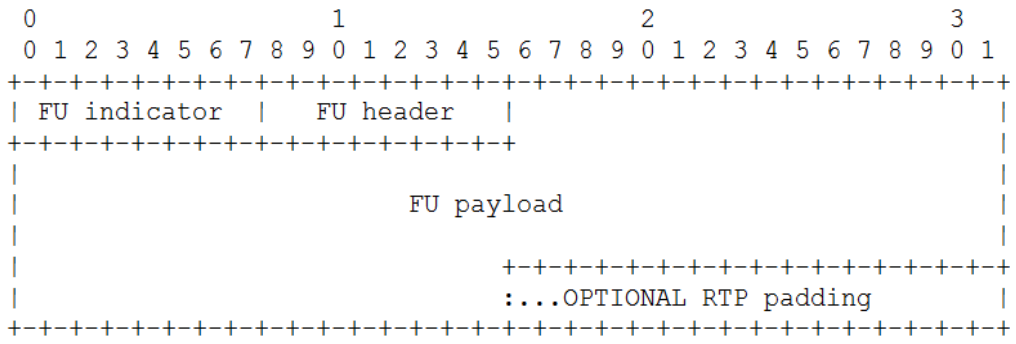

Abb. 56: Fragmentation Unit A [RFC3984]

Der Overhead eines FU-A besteht aus zwei Teilen: dem FU Indicator und dem FU Header. Der NAL Unit Header der fragmentierten NAL Unit wird auf diese beiden Einheiten aufgeteilt. So stehen im FU Indicator das Forbidden Bit (F) und der NAL Reference Indicator (NRI). Die übrigen 5 Bit des FU Indicator dienen zur Identifikation des Pakettypen. Nach Abbildung 51 sind für Fragmentation Units die Werte 28 bzw. 29 erlaubt. Der FU Header besteht aus drei Flags sowie dem NAL Unit Type. Die Flags werden als Start- (S), End- (E) und Reserved-Flag (R) bezeichnet. Start- und End-Flag werden zum Markieren des ersten und letzten Fragments gesetzt, das Reserved-Flag muss den Wert 0 haben.

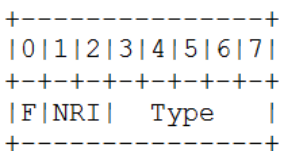

Abb. 57: FU Indicator [RFC3984]

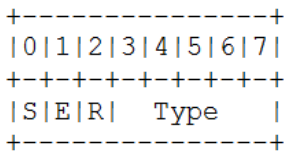

Abb. 58: FU Header [RFC3984]

Die DON der FU-B Pakete folgt nach dem FU Header und ist analog zu den STAP-B Paketen 16 Bit groß.

3.3.3 Transport der Parameter Sets

Es bestehen zwei Wege die Parameter Sets zu transportieren. Zum einen können die Parameter Sets wie jede andere NAL Unit als RTP Payload übertragen werden. Wenn UDP als Transportprotokoll für RTP eingesetzt wird, besteht bei dieser Variante jedoch das Risiko des Verlusts der Parameter Sets. Als Folge dessen ist der ganze Stream nicht dekodierbar. Eine sichere Variante stellt die Übertragung der Parameter Sets als Teil der Session Description dar. Da diese üblicherweise über TCP transportiert wird, können die Parameter Sets so nicht verloren gehen.

3.3.3.1 Transport der Parameter Sets über SDP

Da die Parameter Sets H.264 spezifisch sind, werden sie in SDP unter dem Attribut fmtp transportiert. Die Codec spezifischen Parameter von SDP, also die Werte des fmtp Attributs, werden durch Semikola getrennt. Jeder Parameter hat einen Namen und einen Wert. Die Parameter Sets haben den Namen sprop-parameter-sets, wobei sprop für Stream Properties steht. Der Wert des Parameter sprop-parameter-sets ist die Base64-codierte Form der Parameter Sets. Dabei wird jedes Parameter Set inklusive des zugehörigen NAL Unit Header eigenständig Base64-codiert. Die Trennung der Parameter Sets geschieht über Kommata, die Anordnung entspricht der Decoding Order. Eine Base64-Codierung ist nötig da SDP ein textbasiertes Protokoll ist, während die Parameter Sets binär codierte Daten darstellen.
Zur Veranschaulichung soll ein Beispiel dienen. Eine Zeile der Media Description im SDP-Protokoll könnte wie folgt aussehen:

a=fmtp:96 packetization-mode=1; sprop-parameter-sets=Z0LAHpZmDAQ0,aM48gA==

Der Wert **96** ist der verwendete RTP Payload Type und wird in der Media Description zum Mappen der Parameter auf den zugehörigen Track verwendet. Die Zeile enthält zwei Codec spezifische Parameter: **packetization-mode** und **sprop-parameter-sets**. Der Wert des Parameters **packetization-mode** ist **1**, der Wert von **sprop-parameter-sets** ist **Z0LAHpZmDAQ0,aM48gA==**. Diese Zeichenfolge enthält ein Komma, d.h. sie stellt zwei Parameter Sets dar, **Z0LAHpZmDAQ0** und **aM48gA==**.

3.3.3.2 Base64 Codierung

Die Base64 Codierung wurde im Jahr 2003 von der IETF in RFC 3548 – Base16, Base32 and Base64 Data Encoding standardisiert. Entwickelt wurde sie, um den problemlosen Transport von binär codierten Daten über textbasierte Protokolle zu ermöglichen. [RFC3548]
Die Idee der Codierung ist das Übersetzen von binären Werten in Zeichen. Bei der Base64 Codierung wird dazu eine Tabelle mit 64 Zeichen verwendet, das sogenannte Base64 Alphabet. Als Alphabet dienen die Zeichen A-Z, a-z, 0-9, + und /. Da es 64 Zeichen gibt, können die Werte 0 bis 63 übersetzt werden, welche durch 6 Bit repräsentiert werden. Daraus ergibt sich, dass 3 Byte Binärdaten auf 4 Zeichen übersetzt werden.

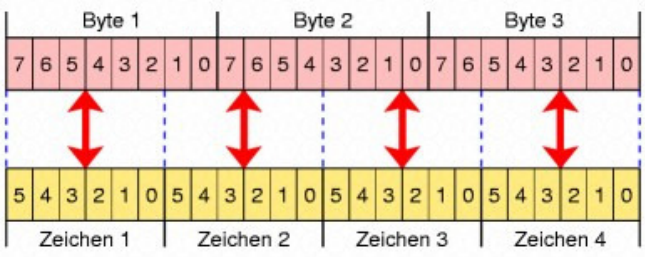

Abb. 59: Base64 Codierung [www16]

Ist die Anzahl an Bytes, aus denen die binär codierten Daten bestehen, nicht durch Drei teilbar, so werden ein oder zwei Bytes mit dem Wert Null angehängt. Im der Base64 codierten Zeichenfolge wird dann das Zeichen = ein- oder zweimal angehängt.

Zur Veranschaulichung soll das obige Beispiel fortgeführt werden:
Die Zeichenfolge **aM48gA==** besteht aus 8 Zeichen, was 6 Byte Binärdaten entspricht. Da am Ende der Zeichenfolge jedoch zweimal das Zeichen = angehangen wurde, wurden auch die Binärdaten mit zwei Byte vom Wert Null aufgefüllt, d.h. die ursprünglich codierten Daten haben eine Größe von 4 Byte. Gemäß Abbildung 60 ergeben sich aus der Zeichenkette die Base64 Werte 26, 12, 56, 60, 32 und

0. Übersetzt man nach Abbildung 59, ergeben sich für die Binärdaten die Werte 0x68, 0xCE, 0x3C, 0x80 bzw.

0110 1000 1100 1110 0011 1100 1000 0000.

Das erste Byte ist der NAL Unit Header. Das Forbidden Bit hat den Wert 0, der NAL Reference Indicator den Wert 3 und der NAL Unit Type den Wert 8. Es handelt sich also um ein Picture Parameter Set.

Wert	Zeichen	Wert	Zeichen	Wert	Zeichen	Wert	Zeichen
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

Abb. 60: Base64 Alphabet [www16]

4 Kriterien zur Bewertung der Videoqualität

Die für die Videoqualität entscheidenden Parameter lassen sich in verschiedene Kategorien einteilen. Eine große Gewichtung bei der Bestimmung der Videoqualität haben Parameter der Netzwerkebene, da sie einen Aufschluss über die Qualität der zum Transport verwendeten Infrastruktur geben. Die Parameter des Transports geben Aufschluss darüber, wie groß der notwendige Overhead ist, um das Video zu transportieren. Schließlich haben noch die Parameter von H.264 selbst erheblichen Einfluss auf die Videoqualität.

4.1 Parameter der Netzwerkebene

Ein Parameter der Netzwerkebene, der einen wesentlichen Einfluss auf die Qualität der Übertragung hat, ist der Jitter. Der Jitter beschreibt Schwankungen der Paketlaufzeiten, genauer gesagt Schwankungen der Zeitabstände bei der Ankunft aufeinander folgender Pakete. Große Schwankungen wirken sich wie Paketverluste aus, da die Pakete nicht mehr rechtzeitig zum Dekodieren ankommen und somit verworfen werden. Es gibt verschiedene Varianten des Jitters. Die im Programm verwendete Variante entspricht der Definition des Interarrival Jitter nach RFC 3550, also die auch in den Reports von RTCP genutzte Variante. Zur Berechnung wird das Zeitintervall zwischen der Ankunft des aktuellen Paket und des vorherigen Pakets sowie das Zeitintervall zwischen den Timestamps der beiden Pakete benötigt. Der Interarrival Jitter wird rekursiv berechnet nach folgenden Regeln:

$$D(i,j) = (Rj - Ri) - (Sj - Si) = (Rj - Sj) - (Ri - Si)$$

$$J(i) = J(i-1) + (|D(i-1,i)| - J(i-1))/16$$

Dabei ist R_i die Ankunftszeit des aktuellen Paket und R_j die Ankunftszeit des vorherigen Pakets. S_i ist der Timestamp des aktuellen RTP Pakets und S_j der Timestamp des vorherigen RTP Pakets. Der Faktor 1/16 dient dazu, Rauschen zu dämpfen. Gleizeitig werden aber auch die Spitzen bei der Schwankung des Jitters gedämpft.
Neben dem Jitter, der sich wie Paketverluste auswirken kann, haben natürlich auch die Paketverluste selbst großen Einfluss auf die Qualität des Videos. Hier gilt es zu unterscheiden, welche Art von Paketverlust auftritt. Geht ein einzelnes Paket verloren, ist die Wahrscheinlichkeit hoher Qualitätseinbußen eher gering. Geht jedoch eine Reihe von aufeinander folgenden Paketen verloren, ist die Wahrscheinlichkeit, dass damit auch wichtige Informationen verloren gehen, deutlich höher. Je länger die Reihe von verlorenen Paketen, desto wahrscheinlicher ist eine Qualitätsminderung. Sowohl Single Losses als auch Burst Losses und deren Dauer können über die RTP Sequenznummer festgestellt werden. Die RTP Sequenznummer kann auch dazu genutzt werden, um die Anzahl an Paketen, die in vertauschter Reihenfolge ankommen, zu ermitteln. Dies kann durch sehr hohe Jitterwerte vorkommen oder auch durch einen Wechsel beim Routing. Ein Routingwechsel könnte über die Auswertung des IP Headers festgestellt werden, da sich in der Regel mit einem Routingwechsel auch die Anzahl an Hops ändert. Jedoch ist es auf Ebene von Java nicht möglich, den IP Header auszuwerten. Hier können abhängig vom Transportprotokoll einige Parameter des Transport Layer im OSI Modell ausgewertet werden. Der Network Layer ist nicht zu erreichen.
Zudem ist die Bitrate ein wichtiger Parameter auf Netzwerkebene. Generell lässt sich sagen, dass eine hohe Bitrate auch hoher Qualität entspricht, da entsprechend mehr Informationen im Bitstrom enthalten sind. Jedoch kann eine hohe Bitrate auch dazu führen, dass der Decoder es nicht mehr schafft, den Bitstrom vollständig zu verarbeiten. Dies sollte jedoch eigentlich durch die Angabe des Levels vermieden werden.
Der letzte ausgewertete Parameter der Netzwerkebene ist die IP Adresse des Servers. Durch die Zuordnung der IP Adresse könnte man eine Aussage über die geographische Region, in der der Server steht, tätigen. Dies hat jedoch nur indirekt Einfluss auf die Qualität der Übertragung. Eine große

geographische Distanz bedeutet lange Paketlaufzeiten und damit können auch die absoluten Schwankungen in der Paketlaufzeit größer sein. Dies wird jedoch durch den Jitter ausgewertet.

4.2 Parameter des Transports von H.264 Videos

Hier wird zunächst einmal die Anzahl an Paketen festgehalten, die für den Transport eingesetzt wurden. Neben der Anzahl an Paketen wird auch die durchschnittliche Anzahl an Bits pro Paket festgehalten. Mit diesen beiden Parametern kann man abschätzen, welcher Overhead nötig ist, um den H.264 Bitstrom zu transportieren und welche durchschnittliche Größe die Slices haben. Die durchschnittliche Anzahl an Bits pro Paket wird inklusive RTP Header und exklusive UDP Headern angegeben.
Als weiterer Parameter des Transports werden die in Kapitel 3.3 beschriebenen Payload Formate von RTP dokumentiert. Hier ist zunächst der Packetization Mode festzuhalten, da durch ihn bestimmt wird, welche Pakettypen verwendet werden und ob die Pakete außerhalb der Decoding Order transportiert werden können. Neben dem Packetization Mode müssen dann natürlich noch die Pakettypen selbst festgehalten werden. Hier reicht eine Einteilung, ob die Pakete einzelne NAL Units enthalten bzw. Aggregation Packets oder Fragmentation Units sind. Im Falle von Aggregation Packets ist es zudem sinnvoll, die durchschnittliche Anzahl von NAL Units, die im Paket enthalten sind, zu dokumentieren. Werden Fragmentation Units eingesetzt, wird die durchschnittliche Anzahl an Fragmenten, in die eine NAL Unit eingeteilt wird festgehalten. Die Anzahl der Fragmente bzw. die Anzahl an NAL Units pro Paket kann bestimmt werden, indem man für jeden Pakettypen sowohl die Anzahl an Paketen festhält als auch die Anzahl an NAL Units. Bei der Aggregation wird dazu das NAL Size Feld des RTP Payload ausgewertet, bei der Fragmentierung das S-Flag des FU Header. Die Auswertung der Pakettypen ist sinnvoll, um eine Einschätzung über die Schwere eines möglichen Paketverlustes treffen zu können. Geht ein Aggregation Packet verloren, so gehen zwar mehrere NAL Units verloren, jedoch sind diese sehr klein und tragen daher wenig Information in sich. Geht ein Fragmentation Unit verloren, so kann die ganze NAL Unit nicht verarbeitet werden. Da die NAL Unit sehr groß ist trägt sie viel Information in sich, d.h. der Verlust wird sich stärker auf die Videoqualität auswirken als der Verlust eines Aggregation Packet.

4.3 Parameter von H.264

Bei der Auswertung der H.264 Parameter kann man unterscheiden zwischen Parametern der Parameter Sets, Parametern der NAL Units, Parametern der Slice Header und Parametern der Slice Daten/Macroblöcke.

4.3.1 Auswertung der Parameter Sets

Die größte Anzahl an Informationen lässt sich aus den Parameter Sets gewinnen. Sehr wichtig für hohe Videoqualität ist die Auflösung, mit der das Video codiert ist. Die Auflösung lässt sich aus zwei Parametern des Sequence Parameter Sets berechnen: pic_width_in_mbs_minus1 und pic_height_in_map_units_minus1. Eine Map Unit ist abhängig davon, ob Macroblock Adaptive Frame Field Coding (MBAFF) verwendet wird, ein einzelner Macroblock oder ein Macroblock Paar. MBAFF ist ein Feature von H.264, um eine höhere Kompression bei der Codierung von Interlaced Video, also Halbbildern, zu erzielen. Android unterstützt nur das Baseline Profile, welches wiederum nur Progressive Video, also Vollbilder, erlaubt. Deshalb entspricht eine Map Unit in diesem Fall einem Macroblock, d.h. pic_height_in_map_units_minus1 gibt die Bildweite in Macroblöcken an, welche eine Weite von 16 Samples haben.
Ein weiterer bedeutender Parameter ist die verwendete Bildrate. Die Bildrate ist keine zum Dekodieren notwendige Information, daher ist auch die Angabe optional. Ist eine Bildrate angegeben, ist diese Teil der Video Usability Informations (VUI). Die VUI Parameter werden in Annex E des

H.264 Standards definiert. Auch die Bildrate wird über zwei Parameter ausgedrückt: num_units_in_tick und time_scale. Das Verhältnis von num_units_in_tick und time_scale beschreibt die kleinste mögliche Zeiteinheit im Video. Ist beispielsweise num_units_in tick gleich 1001 und time_scale gleich 60000, so ist die kleinste erlaubte Zeiteinheit 1001/60000, was einer Frequenz von 59,94 Hz entspricht. Die kleinste erlaubte Zeiteinheit bezieht sich auf die Field Rate, d.h. die Frame Rate entspricht der doppelten Zeiteinheit bzw. der halben Frequenz, also 29,97 Hz.
Die Anzahl der Referenzbilder gibt Aufschluss über die Gewichtung eines Paketverlustes. Je mehr Bilder als Referenzen dienen, desto größer ist die Wahrscheinlichkeit, dass eine Macroblock Partition oder ein Submacroblock nicht auf das verlorene Paket referenzieren. Jedoch sollte die Anzahl an Referenzbildern nicht zu groß gewählt werden, da die dekodierten Bilder im Speicher des Dekoders vorliegen müssen. Wie in Kapitel 2.2.2.2 beschrieben dient der Parameter max_num_ref_frames zur Bestimmung der Anzahl der Referenzbilder.
Der pic_order_cnt_type ist ein Parameter des Sequence Parameter Sets. Er gibt vor, nach welchem Schema die Anzeigereihenfolge der Bildsequenz berechnet werden soll. Ist der pic_order_cnt_type gleich Null, so dient der Parameter pic_order_cnt_lsb des Slice Headers zur Bestimmung der Anzeige Reihenfolge. Sollte der pic_order_cnt_type gleich Eins sein, so folgt im Sequence Parameter Set eine Beschreibung des Picture Order Count Cycle, welcher einer Group of Pictures entspricht. Ist der pic_order_cnt_type gleich zwei, so wird die Dekodierreihenfolge als Anzeigereihenfolge verwendet. Üblicherweise wird pic_order_cnt_type Null verwendet. In diesem Fall geschieht die Berechnung der Anzeigereihenfolge periodisch über den pic_order_cnt_lsb, für den im Sequence Parameter Set ein maximaler Wert enthalten ist. Spätestens wenn der maximale Wert erreicht ist, muss ein Instantaneous Decoding Refresh folgen, d.h. der pic_order_cnt_lsb wird zurück auf Null gesetzt und es folgt ein vollständig mit I-Slices codiertes Bild. Der Parameter, der den maximalen Wert beschreibt, heißt log2_max_pic_order_cnt_lsb_minus4 und beschreibt somit die maximale Größe zwischen komplett aus I-Slices codierten Bildern.
Ein weiteres Merkmal für die Zuverlässigkeit bei Paketverlusten ist die Slice Group Size. Je größer die Anzahl an Slice Groups desto höher muss auch die Anzahl an Slices sein. Und je größer die Anzahl an Slices, desto weniger schwerwiegend ist ein Verlust eines einzelnen Slices.
Level und Profil gehören in die Auswertung, da sie Informationen über die verwendeten Features von H.264 geben und Grenzwerte für die Übertragung liefern. Das Profil muss bei der Auswertung unter Android immer Baseline sein.
Aus diesem Grund muss auch der eingesetzte Entropiecodierer CAVLC sein. Welcher Entropiecodierer eingesetzt wird, gibt Aufschluss über den Kompressionsfaktor und den nötigen Rechenaufwand zum Dekodieren.
Das Chroma Format gibt Aufschluss über die Farbdetails, ein höheres Chroma Format bietet aber nur sehr geringe Qualitätssteigerung. Ebenso wie der Entropiecodierer ist auch das Chroma Format im Baseline Profil vorgeschrieben. Es kann nur Chroma Format 4:2:0 verwendet werden.

4.3.2 Auswertung der NAL Units

Die Auswertung der NAL Units besteht aus den beiden variablen Feldern des Headers. Durch die Auswertung des NAL Reference Indicator erhält man die Information, ob die NAL Unit als Referenz verwendet wird oder nicht. Durch die Einschränkung auf das Baseline Profil sind B-Slices verboten, d.h. dass alle NAL Units, die Slices in sich tragen, auch als Referenz dienen. Nebem dem NAL Reference Indicator wird auch der NAL Unit Type bestimmt. Hier ist insbesondere die Häufigkeit von Instantaneous Decoding Refreshs interessant, da mit dem IDR ein I-Frame übertragen wird. Spätestens mit dem I-Frame werden die Qualitätseinbußen durch vorherige Paketverluste behoben. Zudem ist interessant, ob Supplement Enhancement Information Messages und Access Unit Delimiter verwendet werden. Diese werden zum Dekodieren nicht benötigt, stellen also auch eine Art Overhead dar.

4.3.3 Auswertung der Slice Header

Die Auswertung des Slice Headers ergibt die Information, von welchem Typ der aktuelle Slice ist. Der Name des zugehörigen Parameters ist slice_type. Die Häufigkeit von I-Slices gegenüber P-Slices und B-Slices, ist eine entscheidende Information über die Kompressionsrate und gibt Aufschluss über die Wahrscheinlichkeit einer Qualitätsminderung bei Paketverlusten.

Ein weiterer Parameter des Slice Headers, der zur Auswertung dient, ist first_mb_in_slice. Das Zählen der Werte gleich null und ungleich null, läßt einen Rückschluss auf die durchschnittliche Anzahl an Slices pro Frame zu. Diese ist in der Regel konstant, kann aber durchaus variabel sein.
Als letzter Parameter des Slice Headers wird die long_term_pic_num ausgewertet. Zunächst wird angenommen, das der Slice Teil eines ShortTermRefPics ist. Ist die long_term_pic_num im Slice Header enthalten, handelt es sich um ein LongTermRefPic. Der Verlust von LongTermRefPics wiegt deutlich schwerer als der Verlust eines ShortTermRefPics, da ein LongTermRefPic für mehr Bilder als Referenz dient als ein ShortTermPic.

4.3.4 Auswertung der Slice Daten

Die auftreteden Macroblöcke geben Aufschluss über den Informationsgehalt des Bildes. So bedeuten viele I-Macroblöcke viele neue Information, während P- und B- Macroblöcke auf Bewegung hindeuten. Je kleiner die Partitionen, desto feiner ist die Bewegung. Skip-Macroblöcke geben Aufschluss über globale Bewegung. Die Auswertung des Macroblocktypen geschieht über die Parameter mb_type und mb_skip_run der Slice Daten.

5 Das Android Framework

Das Android Framework liefert ein komplettes Software-Paket für mobile Endgeräte. Entwickelt wurde das Software-Paket von der Open Handset Alliance. Die Open Handset Alliance ist ein Zusammenschluss von verschiedenen Netzbetreibern, Chipherstellern, Geräteherstellern, Softwareunternehmen und kommerziellen Dienstleistern. Im November 2007 haben die 34 Gründungsmitglieder die Entwicklung von Android angekündigt. Bis heute ist die Anzahl der Mitglieder auf 47 gewachsen. Den maßgeblichen Anteil bei der Entwicklung von Android hat die Firma Google. Im September 2009 wurde das erste Gerät, welches mit Android läuft, auf den Markt gebracht, das G1. Gebaut wurde das G1 vom chinesichen Hersteller HTC, der Netzbetrieb wurde in den USA und Teilen Europas von T-Mobile übernommen.
Das Android Framework besteht aus vier Ebenen: dem Betriebsystem, einer Laufzeitumgebung und zugehörigen Bibliotheken, dem Application Framework sowie den Applikationen selbst. Das Betriebssytem basiert auf dem Linux-Kernel 2.6. Darauf aufgesetzt wurde eine eigens für Android entwickelte virtuelle Maschine, die Dalvik Virtual Machine. Die Dalvik VM wurde so konzipiert, dass mehrere Instanzen parallel laufen können. Dies ist nötig, da jede Android Applikation als Prozess mit eigener Instanz der Dalvik VM läuft. Das Application Framework stellt für Entwickler eine Reihe von Klassen und Schnittstellen zur Verfügung, von denen etwa ein Drittel Android spezifisch ist.

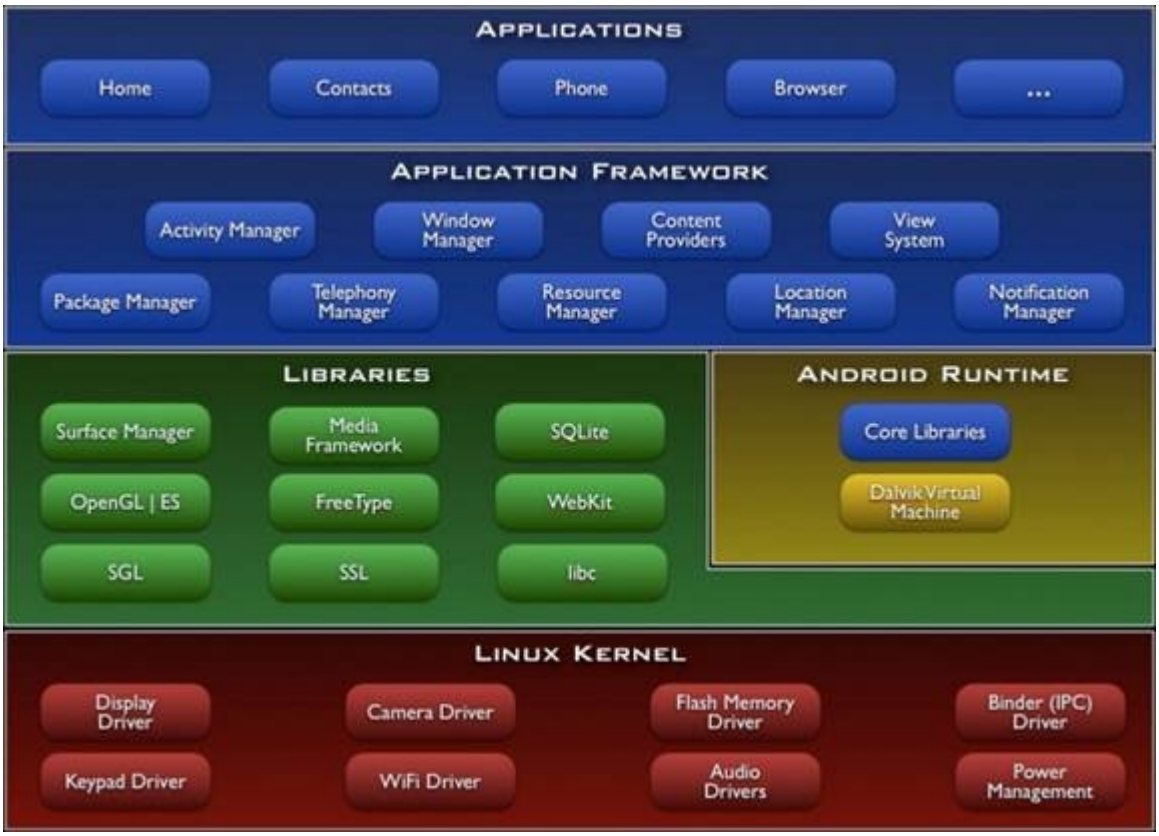

Abb. 61: Android Architektur [www17]

Android ist ein Open Source Projekt, d.h. die Quellcodes sind frei verfügbar. Entwicklern soll damit die Möglichkeit gegeben werden, auch die Kern-Funktionalitäten eines mobilen Geräts in ihren Anwendungen nutzen zu können. Außerdem soll so die Unterstützung von neuen Technologien gesichert werden.

5.1 Die vier Building Blocks einer Anwendung

Eine Android Applikation besteht immer aus einer Kombination der folgenden Blöcke:

- Activities
- Services
- Broadcast Receiver
- Content Provider

Innerhalb einer Applikation müssen nicht alle Blöcke vorkommen, sie dürfen aber mehrfach vorkommen. Eine Applikation kann also beispielsweise aus drei Activities bestehen. Welche Blöcke zu einer Applikation gehören, wird im Manifest der Applikation festgehalten. Activities sind die gebräuchlichste Form der Building Blocks. Deshalb sollen diese etwas genauer betrachtet werden als die Übrigen.

5.1.1 Activities

Eine Activity dient dazu, mit dem User zu interagieren. Deshalb muss jede Activity mindestens ein View besitzen. Ein View ist ein rechteckiger Ausschnitt, der zur Darstellung von Inhalten auf dem Bildschirm dient. Der Ausschnitt muss nicht notwendigerweise den ganzen Bildschirm füllen. Activities werden vom System über einen Activity Stack verwaltet. Die Activity, die an oberster Stelle des Stacks steht, ist die Activity, die im Vordergrund des Bildschirms angezeigt wird. Sie hat den Focus. Füllt diese Activity den Bildschirm nicht vollständig aus, bleibt die im Stack darunter liegende Activity sichtbar ohne den Focus zu haben.
Activites sind zustandsbehaftete Objekte. Die erlaubten Zustände und die möglichen Übergänge zwischen den Zuständen sind in Abbildung 62 zu sehen.

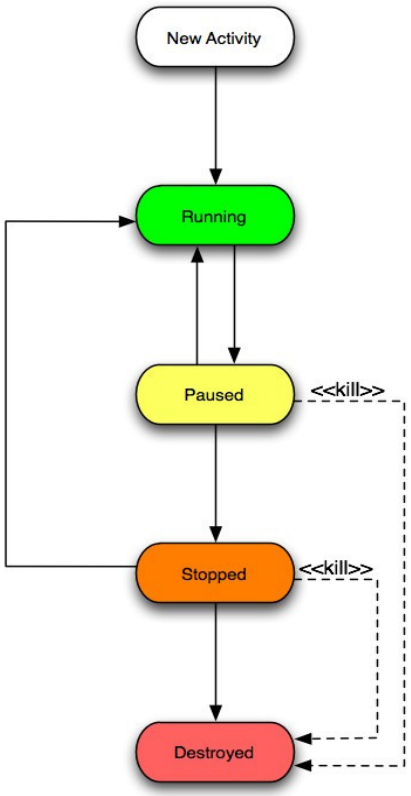

Abb. 62: Activity Lifecycle [www18]

Es kann nur eine Activity existieren, die den Zustand Running hat. Diese liegt im Activity Stack oben. Verliert eine Activity den Focus, weil eine neue Activity in den Vordergrund kommt, bleibt aber dennoch sichtbar, so wird sie in den Zustand Paused versetzt. Verliert die Activity den Focus und ist nicht mehr sichtbar, so wird sie in den Zustand Stopped versetzt. Die Activities in Zuständen Paused und Stopped können vom System bei Speichermangel gekilled werden. Dabei werden zuerst die Activites gekilled die im Zustand Stopped sind.
Activities werden von der Oberklasse Activity vererbt. In den Activities können Methoden der Klasse Activity überschrieben werden, die beim Übergang zwischen den einzelnen Zuständen aufgerufen werden. Abbildung 63 ordnet dem Zustandsübergang die zugehörige Methode zu.

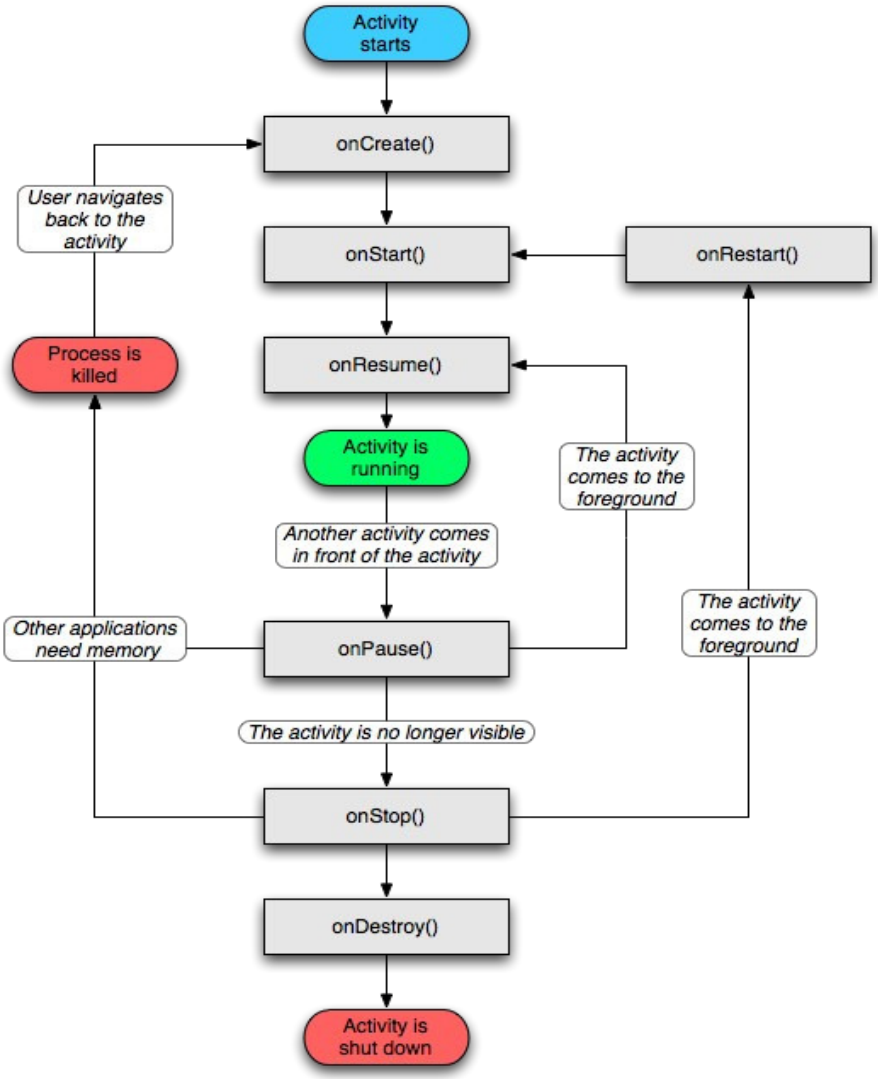

Abb. 63: Zustandspfade einer Activity [www19]

Das Diagramm zeigt drei wichtige Abschnitte. Der innerste Abschnitt, zwischen onResume() und onPause() beschreibt die Zeit, in der eine Activity den Focus hat, also im Vordergrund steht. Eingehüllt wird dieser vom Abschnitt zwischen onStart() und onStop(), welcher die Zeit beschreibt, in der die Activity sichtbar ist. Der äußerste Abschnitt, zwischen onCreate() und onDestroy(), beschreibt die Zeit in der die Activity existiert. Die onCreate()-Methode muss in jeder Activity implementiert werden, die restlichen Methoden der Zustandsübergänge sind optional. Innerhalb der onCreate()-Methode erfolgt ein Aufruf der Methode setContentView(int). Diese Methode beschreibt über einen Identifier das View oder Layout welches vorgibt, was dargestellt werden soll. Neben der setContentView()-Methode muss

auch ein Aufruf der onCreate()-Methode der Oberklasse Activity erfolgen. Die minimale Activity hat also folgenden Aufbau:

```
public class MyActivity extends Activity
{
	@Override
	public void onCreate(Bundle savedInstanceState)
	{
		super.onCreate(savedInstanceState);
		setContentView(R.layout.mylayout);
	}
}
```

Die Klasse R wird von Android automatisch generiert und beinhaltet verschiedene Identifier. So auch Identifier für mögliche Layouts, die wiederum ein oder mehrere Views enthalten.
In einem Bundle können verschiedene elementare Datentypen festgehalten werden. Bundles werden benutzt um den Zustand einer Actvity zu sichern und wiederherzustellen oder um Daten unter Activities auszutauschen.

5.1.2 Services, Broadcast Receiver und Content Provider

Ein Service ist ein Teil einer Applikation, welcher im Hintergrund läuft. Services haben also kein User Interface. Es existieren zwei Wege, um einen Service zu starten. Zum einen können Services direkt gestartet werden, zum anderen kann ein Kanal geöffnet werden, über den Clients den Service an sich binden und mit ihm kommunizieren können. Eine mögliche Anwendung für einen Service ist die Wiedergabe von Musiktiteln. Nachdem der Benutzer den Titel ausgewählt hat, möchte er möglicherweise andere Aufgaben erledigen und die Musik im Hintergrund weiter laufen lassen.

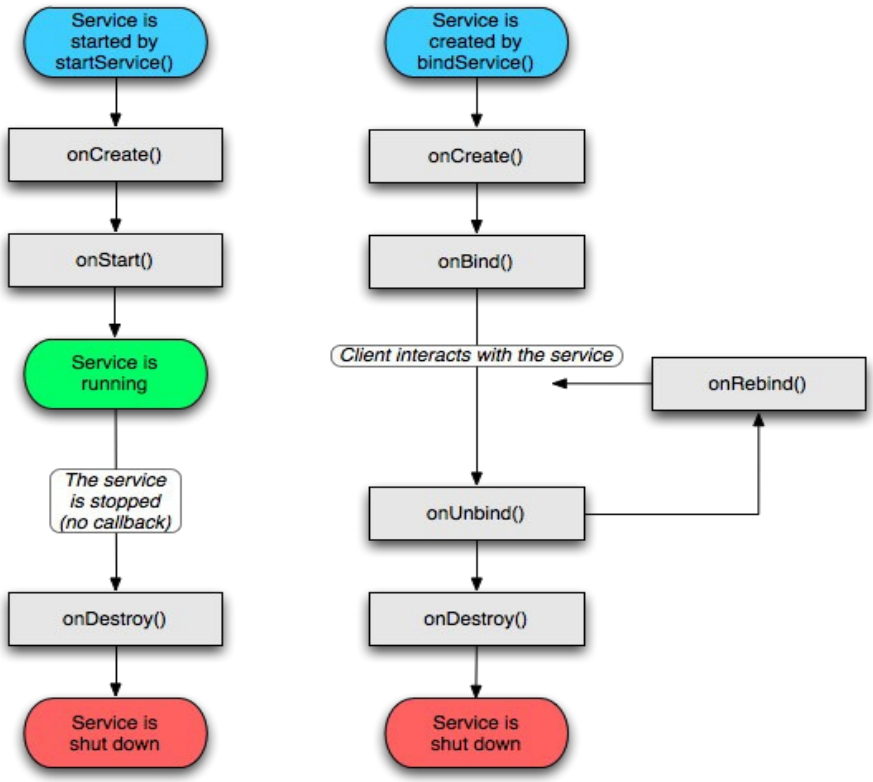

Abb. 64: Zustandspfade von Services [www19]

Broadcast Receiver sind Objekte, die Aufgaben als Reaktion auf eine Nachricht ausführen. Damit sie dies tun, müssen sie registriert und ihnen ein Nachrichtenfilter zugewiesen werden. So kann man beispielsweise Aufgaben als Reaktion auf einen Anruf oder den Empfang einer SMS ausführen lassen, aber auch selbst definierte Broadcasts können versendet werden. Die Registrierung eines Broadcast Receiver kann dynamisch erfolgen oder im Manifest der Applikation festgehalten werden.
Applikationen können Daten auf herkömmliche Weise in Dateien oder einer Datenbank speichern. Eine Alternative dazu stellen Content Provider dar. Daten sollten in einem Content Provider gespeichert werden, wenn es nötig ist, die Daten zwischen mehreren Applikationen teilen zu können, wie beispielsweise Kontaktinformationen.

5.2 Intents und Intent-Filter

Ein Intent dient als Verbindung zwischen einzelnen Komponenten/Blöcken. Android unterscheidet zwischen impliziten und expliziten Intents. Ein expliziter Intent ruft die Komponente direkt über ihren Namen auf, d.h. die aufgerufene Komponente ist vorher bekannt. Ein impliziter Intent gibt nur eine Aktion und die Daten an, auf der die Aktion ausgeführt werden soll an. Welche Komponente auf den Intent reagiert, hängt von den Komponenten selbst ab. Damit die Komponenten wissen auf welche impliziten Intents sie reagieren sollen, können ihnen Intent-Filter zugewiesen werden. Ein Intent-Filter beinhaltet die Aktion, auf die die Komponente reagieren soll sowie eine Kategorie, die angibt unter welchen Bedingungen auf den Intent reagiert wird. Die Intent-Filter der Komponenten werden im Manifest der Anwendung festgehalten. Der Aufruf eines impliziten Intents sieht z.B. folgendermaßen aus:

```
Uri uri = Uri.parse("http://www.fh-koeln.de");
Intent intent = new Intent(Intent.ACTION_VIEW, uri);
startActivity(intent);
```

Als Aktion wird hier das Anzeigen der Daten auf dem Bildschirm gewählt, signalisiert durch `Intent.ACTION_VIEW`. Die Daten, die angezeigt werden sollen, stehen in der URI. Android löst nun anhand der Intent-Filter auf, welche Komponente zum Anzeigen der Daten dienen kann. Werden mehrere Komponenten gefunden, so wird eine Liste von verfügbaren Komponenten zur Auswahl bereitgestellt.
Neben den Daten können Intents eine Menge von Extras in Form eines Bundles mitgegeben werden. Dies dient dazu, Daten zwischen Komponenten zu transportieren:

```
Bundle bundle = new Bundle();
bundle.putString("socket", socket);
bundle.putString("file", file);

Intent intent = new Intent(StartActivity.this, VideoActivity.class);
intent.putExtras(bundle);
startActivity(intent);
```

In diesem Beispiel werden zwei Strings, enthalten in `socket` und `file`, in ein Bundle verpackt. Dieses Bundle wird dem Intent als Extra angehängt. Die Komponente, die auf den Intent reagiert, kann die beiden Strings anhand ihrer Beschreibung `"socket"` und `"file"` identifizieren und auswerten. Der Intent hier ist ein expliziter Intent. Die benötigten Argumente um den Intent zu konstruieren sind der aktuelle Context in dem sich die Komponente befindet sowie der Name der aufgerufenen Komponente. Die aktuelle Komponente ist im Beispiel die StartActivity, die aufgerufene Komponente die VideoActivity.

5.3 Das Android Manifest

Jede Applikation muss eine XML-Datei mit dem Namen AndroidManifest.xml in ihrem Wurzelverzeichnis haben. Zur Erläuterung eines Manifests soll das Manifest der entwickelten Anwendung dienen:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="de.mediaplayer"
          android:versionCode="1"
          android:versionName="1.0.0">
   <uses-sdk android:minSdkVersion="2" />
   <uses-permission android:name="android.permission.INTERNET" />
   <application android:icon="@drawable/icon"
                android:label="@string/app_name"
                android:debuggable="true">
      <activity android:name=".StartActivity"
                android:label="MediaPlayer">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
      <activity android:name=".VideoActivity"
                android:label="MediaPlayer"
                android:screenOrientation="landscape">
         <intent-filter>
            <action android:name="android.intent.action.VIEW" />
            <category android:name="android.intent.category.DEFAULT" />
         </intent-filter>
      </activity>
      <activity android:name=".StatsActivity"
                android:label="MediaPlayer"
                android:screenOrientation="portrait">
         <intent-filter>
            <action android:name="android.intent.action.VIEW" />
            <category android:name="android.intent.category.DEFAULT" />
         </intent-filter>
      </activity>
   </application>
</manifest>
```

Das Tag `<manifest>` muss in jedem Manifest vorkommen, einen Namespace deklarieren, den Java Package Namen der Applikation definieren und das Tag `<application>` enthalten. Alle weiteren Tags und Attribute sind optional. Die Attribute `versionCode` und `versionName` dienen zur Versionsverwaltung. Während `versionCode` zur internen Verwaltung dient, ist `versionName` eine Versionsnummer, die dem User angezeigt werden kann. Seit der Veröffentlichung vom Android SDK 1.5 wird die Angabe des Tags `<uses-sdk>` mit dem Attribut `android:minSdkVersion` empfohlen, da mit dem Update auf Android 1.5 Änderungen in der API vorgenommen wurden und so Kompatibilitätsprobleme ausgeschlossen werden können. Zudem werden hier Rechte definiert, die den Zugriff auf geschützte Bereiche der API erlauben. Ebenso können Rechte definiert werden, die andere Anwendungen benötigen, um auf die Komponenten der Anwendung zuzugreifen. Dies geschieht über die Tags `<uses-permission>` und `<permission>`. Da die entwickelte Anwendung über Sockets kommuniziert, ist es nötig ihr die Rechte `android.permission.INTERNET` einzuräumen. Bei der Installation von Anwendungen wird dem User mitgeteilt, welche Rechte eine Applikation besitzt.
Das Tag `<application>` beinhaltet die Komponenten der Anwendung. Dies können die in Kapitel 5.1 beschriebenen Building Blocks sein, repräsentiert werden die Building Blocks durch die Tags `<activity>`, `<service>`, `<receiver>` und `<provider>`. Die Attribute `android:icon` und

android:label müssen angegeben werden, da keine Defaulteinstellungen für das Icon und den Namen der Anwendung existieren. Das Attribut android:debuggable ermöglicht es, eine Anwendung in einer Entwicklungsumgebung zu debuggen.
Der grundlegende Aufbau der vier Building Blocks ist gleich, deshalb reicht hier eine Erläuterung für das <activity> Tag. Zunächst wird der Block über das Attribut android:name lokalisiert. Der Wert des Attributs muss ein voll qualifizierter Klassenname sein, also beispielsweise de.mediaplayer.VideoActivity. Beginnt der Klassenname mit einem Punkt, wird das im <manifest> Tag definierte Package vorangestellt. Für jeden Block kann das Attribut android:label belegt werden. Dieser Wert wird in der Titelleiste des Bildschirms dem User angezeigt. Schließlich können alle Blöcke mit Ausnahme des Content Provider das Tag <intent-filter> beinhalten. Welche weiteren Tags und Attribute innerhalb des Building Block benötigt werden, ist stark von der Anwendung abhängig. Das obige Manifest legt für zwei der drei Activities die Ausrichtung des Bildschirms über das Attribut android:screenOrientation fest.
Das <intent-filter> Tag muss ein <action> Tag beinhalten. Desweiteren beinhaltet es in der Regel auch noch ein <category> Tag. Das <action> Tag füllt den Intent Filter mit einer Aktion, die im Attribut android:name festgehalten ist. Die Aktion android.intent.action.MAIN sagt über eine Activity aus, dass die Activity als Einstiegspunkt einer Anwendung dienen kann und keine weiteren Daten benötigt. Der Wert android.intent.action.VIEW entspricht dem Beispiel aus Kapitel 5.2 und sagt aus, dass die Activity Daten auf dem Bildschirm ausgibt. Ist der Block ein Broadcast Receiver, so könnte der Wert z.B. android.provider.Telephony.SMS_RECEIVED sein, was zur Folge hätte, dass der Broadcast Receiver seine Aufgabe beim Empfang einer SMS ausführt. Das <category> Tag legt fests unter welchen Bedingungen ein Block ausgeführt werden kann. Der Wert android.intent.category.LAUNCHER sagt aus, dass dieser Block als Einstiegspunkt dient, wenn die Anwendung aus dem Menu gestartet wird.

5.4 Views und Layouts

Ein User Interface wird unter Android aus Views und View Groups zusammen gesetzt. Views treten als rechteckige Objekte auf, die verantwortlich sind für das Zeichnen des Inhalts und das Event Handling. Die Klasse View stellt auch die Oberklasse für Widgets dar. Widgets sind Objekte, die für die Interaktion mit dem User verantwortlich sind, beispielsweise Buttons, Checkboxen und Textfelder. Eine View Group ist eine Sammlung von Views. Die Klasse ViewGroup dient als Oberklasse für verschiedene Layouts.

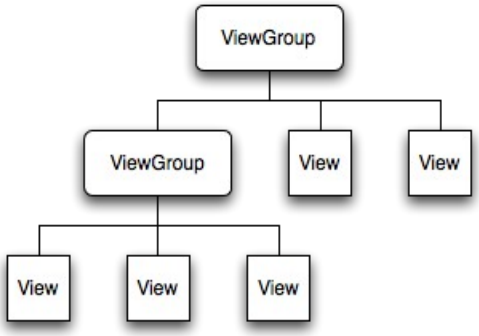

Abb. 65: View Hierarchie [www20]

Layouts beschreiben wie die einzelnen Views angeordnet werden, beispielsweise linear oder absolut bzw. relativ zu anderen Views. Dabei kann ein Objekt innerhalb eines Layouts selbst wieder Layout sein, d.h. es sind beliebig komplexe Anordnungen von Views möglich.

5.4.1 Erstellung des User Interface

Es gibt zwei Varianten, um User Interfaces zu erstellen. Eine dynamische Variante ist die Zusammenstellung der Views und Layouts im Quellcode. Eine Ergänzung dazu ist die Erstellung des User Interface in einer XML-Datei. Die Datei enthält eine XML-Deklaration gefolgt vom Wurzel-Element der View Hierarchie. Im unten stehenden Beispiel ist das Wurzel-Element eine View Group, genauer gesagt ein AbsoluteLayout. Innerhalb des AbsoluteLayout werden zwei Views angeordnet, eine VideoView und eine selbst geschriebene View. Diese muss über den voll qualifizierten Klassennamen angegeben werden. Typisch für das AbsoluteLayout ist die Angabe von Positionen, an denen die Views liegen. Wird keine Position angegeben, so wird die View an Position (0, 0) platziert. Für alle Views kann eine Höhe und Weite definiert werden. Diese kann in Pixeln ausgedrückt werden oder auch in Relation zum Eltern Element.

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent">
      <VideoView
            android:id="@+id/screen"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent">
      </VideoView>
      <de.mediaplayer.ResultView
            android:id="@+id/result"
            android:layout_x="460px"
            android:layout_y="0px"
            android:layout_width="20px"
            android:layout_height="20px">
      </de.mediaplayer.ResultView>
</AbsoluteLayout>
```

Um die einzelnen Elemente im Quellcode ansprechen zu können, müssen sie mit einer ID versehen werden. Android legt dann in einer Resourcenklasse R entsprechende Identifier an.

```
public final class R
{
      public static final class id
      {
            public static final int result=0x7f05000b;
            public static final int screen=0x7f050005;
      }
      public static final class layout
      {
            public static final int video=0x7f030002;
      }
}
```

Die VideoView z. B. lässt sich im Quellcode dann über folgende Zeile ansprechen:

```
VideoView videoView = (VideoView) findViewById(R.id.screen);
```

Weist man einem Layout ebenfalls eine ID zu, so lassen sich während der Laufzeit weitere Views hinzufügen oder entfernen. Die XML-Datei wurde in einem Resourcenordner unter dem Namen video.xml gespeichert. Deshalb existiert in der Resourcendatei R ein Eintrag mit dem Namen *video*. Durch den Aufruf von

```
setContentView(R.layout.video);
```

wird das in der XML-Datei definierte Layout angezeigt.

5.4.2 Eigene Views und Layouts schreiben

Neben den bereits vordefinierten Views und Layouts besteht die Möglichkeit vollkommen frei eigene Views und Layouts zu kreieren. Um dies zu tun, muss die eigene View die Klasse View erweitern bzw. das eigene Layout die Klasse ViewGroup erweitern. Die Erstellung eines eigenen Layouts gestaltet sich ganz analog zur Erstellung einer eigenen View, jedoch lassen sich mit den gegebenen Layouts die meißten Anforderungen erfüllen, so dass hier nur die Erzeugung einer eigenen View erläutert werden soll. Beim Erstellen der eigenen View müssen innerhalb der Klasse explizit zwei Konstruktoren angegeben werden. Einer dient zur dynamischen Erzeugung der View aus dem Quellcode heraus, der Andere dient zur Erzeugung über die XML-Datei. Zum Zeichnen der View muss noch die onDraw()-Methode der Klasse View überschrieben werden.

```
public class ResultView extends View
{
	Paint color;
	int alpha = 255;
	int red = 255;
	int green = 0;
	int blue = 0;

	public ResultView(Context context)
	{
		super(context);
		color = new Paint();
		color.setARGB(alpha, red, green, blue);
	}

	public ResultView(Context context, AttributeSet attrs)
	{
		super(context, attrs);
		color = new Paint();
		color.setARGB(alpha, red, green, blue);
	}

	@Override
	protected void onDraw(Canvas canvas)
	{
		color.setARGB(alpha, red, green, blue);
		canvas.drawCircle(10, 10, 5, color);
	}
}
```

Das View aus dem obigen Beispiel zeichnet einen einfachen Kreis auf dem Bildschirm. Der erste Konstruktor dient für die Erzeugung aus dem Quellcode, der Zweite für die Erzeugung innerhalb der XML-Datei. Abbildung 66 zeigt die obige View. Rechts oben im Bild ist der gezeichnete Kreis zu erkennen.

5.4.3 VideoView und MediaController

Die MediaPlayer-Klasse kann für das Abspielen von Audio- und Video-Dateien sowie Streams verwendet werden. Jedoch ist die Implementierung des MediaPlayer relativ komplex, eine deutlich einfachere Möglichkeit bietet die VideoView-Klasse. Wurde in der XML-Datei für das Layout eine VideoView angelegt, so lässt sich das Abspielen des Videos durch wenige Zeilen starten:

```
Uri uri = Uri.parse("rtsp://hueffmeyer.dyndns.org/IceAge3.mp4");
Videoview videoView = (VideoView) findViewById(R.id.screen);
videoView.setVideoURI(uri);
```

Das Navigieren innerhalb des Videos kann über Buttons geregelt werden. Der Nachteil von Buttons ist jedoch, dass sie Platz auf dem Bildschirm einnehmen, dieser Platz wiederum fehlt dann für die Darstellung des Videos oder sonstiger Informationen. Zwar könnte man die verwendeten Views in einer ScrollView bündeln, dies hat aber den Nachteil, dass die Steuerung und das gesteuerte Element nicht unbedingt zeitgleich sichtbar sind. Android bietet für dieses Problem eine elegante Lösung, den MediaController. Der MediaController ist eine View die Buttons für die Steuerung des MediaPlayer bzw. des VideoView enthält. Der Vorteil des MediaController ist, dass er sich verstecken und wieder hervorholen lässt. Wann der MediaController sichtbar ist, wird über Events bestimmt. Er erscheint beim Berühren einer anderen View. Diese View ist sein Anker. Für den Fall, dass Video abgespielt wird, ist es sinnvoll, dass die durch den MediaController kontrollierte View und die View auf der dieser verankert ist übereinstimmen. Bei der Wiedergabe von Audio-Dateien kann es jedoch sinnvoll sein, den MediaController auf einer anderen View zu verankern.

```
MediaController mediaController = new MediaController(this);
videoView.setMediaController(mediaController);
```

Wird nicht explizit ein Anker angegeben, so dient die View, die der MediaController kontrolliert, auch gleichzeitig als Anker. Abbildung 66 zeigt den MediaController.

Abb. 66: MediaController

Bei der Darstellung von H.264 codierten Videos, die über RTP transportiert wurden, kam es unter Android 1.1 zu erheblichen Problemen. Wurde ein Slice innerhalb von Fragmentation Units transportiert, brach die Darstellung an dieser Stelle ab. Um dieses Problem zu beheben, mussten die Fragmentation Units zunächst wieder zu einer NAL Unit zusammen gebaut und in einem RTP Paket verpackt werden. Dieses RTP Paket wurde an den MediaPlayer bzw. die VideoView weitergeleitet und konnte dann dargestellt werden. Die zusätzlich nötigen Schreiboperationen führten jedoch bei einer Anhäufung von Fragmentation Units, wie sie beispielsweise bei schnellen Szenenwechseln auftreten, zu leichtem Ruckeln des Videos. Das Problem bei der Verarbeitung von Fragmentation Units wurde mit dem Release der Nachfolgeversion Android 1.5 behoben.

6 Softwarearchitektur und Implementierung

In diesem Kapitel soll im ersten Teil der entstandene Software Entwurf sowie der Weg dorthin beschrieben werden. Im zweiten Teil sollen Implementierungsdetails erläutert werden.

6.1 Die Softwarearchitektur

6.1.1 Zielsetzung

Ziel dieser Arbeit war es, eine gute Software zu entwickeln. Welche Faktoren bei der Sicherung der Qualität von Software betrachtet werden müssen, wird in der ISO Norm 9126 beschrieben. Gute Software zeichnet sich durch die folgenden Merkmale aus: Funktionalität, Zuverlässigkeit, Nutzbarkeit, Performance/Leistung, Wartbarkeit/Erweiterbarkeit und Portierbarkeit.

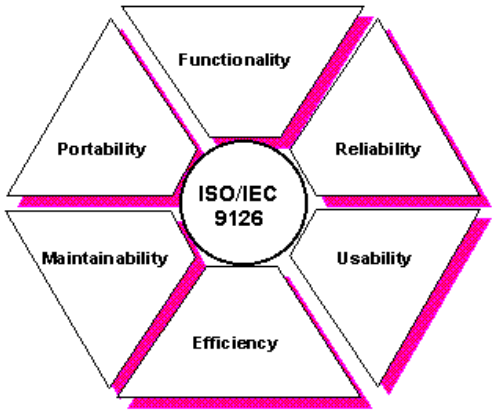

Abb. 67: Merkmale guter Software [www21]

Funktionalität bedeutet für die entworfene Software vor allem, dass die ausgewählten Parameter korrekt aus dem Datenstrom ausgelesen werden, d.h. dass die richtigen Messwerte bestimmt und dargestellt werden. Die *Zuverlässigkeit* wird durch eine hohe Fehlertoleranz gesichert, d.h. die Software fällt möglichst selten aus, weil sie in Fehlerzustände gerät. *Nutzbarkeit* heißt für den Entwurf, dass der Benutzer die Software leicht bedienen kann. Es soll also einfach sein, die Software mit den richtigen Eingaben zu steuern. Die Software sollte dem Benutzer möglichst viel Aufwand abnehmen. Für die entstandene Software bedeutet dies, dass der Benutzer möglichst wenig Einstellungen konfiguriert, sondern die Software diese automatisch erkennt. Ausgaben sollen leicht verständlich und übersichtlich angeordnet sein. Die *Performance* bzw. *Leistung* der Software ist ein Faktor, der entscheidend von den zur Verfügung stehenden Resourcen abhängt, da viele Prozesse durch den H.264 Standard vorgeschrieben werden. An den übrigen Stellen muss darauf geachtet werden, möglichst einfache Algorithmen einzusetzen. Es gibt keinen schnelleren Code als keinen Code. Die *Wartbarkeit* soll ein einfaches Einarbeiten in die Software für Entwickler ermöglichen, d.h. der Quellcode sollte einfach zu analysieren sein. Die *Erweiterbarkeit* soll sicherstellen, dass zusätzliche Features, beispielsweise die Unterstützung weiterer Codecs, leicht in die Software integrierbar sind. Die *Portierbarkeit* soll dazu dienen, dass die Software mit möglichst wenig Aufwand in eine andere Umgebung übertragen werden kann. Im Falle der erstellten Software heißt das, dass sie mit wenig Änderungen unter einem anderen System als Android laufen kann.

6.1.2 Lösungsansätze

Erste Aufgabe bei der Entwicklung der Software war es eine Möglichkeit zu finden, um die zwischen MediaPlayer und Streaming Server ausgetauschten Daten abzufangen.
Der erste Lösungsansatz war es, einen Netzwerksniffer unter Android einzusetzen. Android bietet jedoch nicht von Haus aus die Möglichkeit Netzwerkverkehr mitzulesen. Für diesen Lösungsansatz wäre es nötig gewesen, die libpcap-Bibliothek, die unter anderem in Wireshark eingesetzt wird, auf Android zu portieren und die Funktionen über Java Native Interface (JNI) bekannt zu machen, so dass sie innerhalb einer Anwendung nutzbar sind. Die Vorteile dieses Ansatzes sind zum einen die hohe Geschwindigkeit, da große Teile der Software mit C++ geschrieben wären und zum anderen der Zugriff auf untere OSI-Schichten. So wäre es möglich gewesen, den IP Header auszuwerten. Es hat sich jedoch rausgestellt, dass es zum Zeitpunkt des Beginns dieser Arbeit praktisch nicht möglich war eine eigene Bibliothek zu verwenden. Während dies im Emulator zwar durchaus möglich ist, fehlten auf einem echten Gerät die Berechtigungen, um eine Bibliothek unterzubringen. Anfang Mai dieses Jahres ist mit dem Release von Android 1.5 und dem Android SDK 1.5 auch das Android Native Development Kit (NDK) erschienen. Dieses erlaubt es nun, C und C++ Bibliotheken in die eigene Anwendung einzubetten und so native Funktionen zu nutzen.
Ein weiterer Lösungsansatz war die Implementierung eines RTSP-Clients und die Aufzeichnung der Daten als MP4-Datei. Dieser Ansatz erschien jedoch äußerst resourcenlastig, da das Gerät parallel lesen, schreiben und decodieren müsste. Zudem würden die direkten Auswirkungen der Netzparameter nicht sichtbar. Aufgrund der Nachteile dieses Ansatzes hatte er nur eine Berechtigung als Notlösung.
Der schließlich implementierte Lösungsansatz ist eine Art Proxy zwischen dem MediaPlayer und dem Streaming Server. Der MediaPlayer richtet seine Anfrage an die Zwischenstelle, welche die Anfrage manipuliert und an den Streaming Server weiterleitet. Der Streaming Server sendet die Daten nach Vorgabe der manipulierten Anfrage, so dass die Zwischenstation diese abgreifen, auswerten und an den MediaPlayer weiterleiten kann. Um leichte Portierbarkeit zu gewährleisten, ist die Software so entworfen, dass möglichst wenige Android spezifische Funktionen eingesetzt werden, d.h. die Funktionen der Android API werden nur eingesetzt, um das User Interface zu gestalten.

6.1.3 Der Entwurf

Android spezifische Funktionen übernehmen nur das Einlesen der URI des Videos und die Darstellung der ausgewerteten Parameter. Diese Aufgaben werden von drei Activities übernommen. Eine Activity für das Einlesen, zwei Activities für die Darstellung der Informationen, abhängig davon ob eine detaillierte Darstellung erfolgen soll oder die Darstellung über die in Abbildung 66 gezeigte „Ampel“.
Als Schnittstelle zwischen den beiden Activities und den Klassen zur Auswertung des Videos dient die Klasse StreamingHandler. Der StreamingHandler hat eine Reihe von Membern für die Auswertung der einzelnen Protokolle, einen RTSPHandler, eine Liste von RTPHandlern und einen RTCPHandler. Die Größe der Liste der RTPHandler ist abhängig von der Anzahl der transportierten Tracks. Jeder der ProtokollHandler besteht aus einem ServerHandler und einem PlayerHandler, diese haben jeweils einen Socket und sind zum Teil als Threads implementiert. Die RTPHandler haben als Member einen Codec und einen abstrakten Parser. Abhängig vom Codec kann dem RTPHandler ein konkreter Parser zugewiesen werden, an den die Pakete zur Auswertung übergeben werden. Dieser Aufbau soll die leichte Erweiterbarkeit der Software um neue Parser für andere Codecs sichern. Der RTSPHandler besitzt ein Feld von abstrakten Parsern. Er bekommt eine Liste mit Codecs übergeben nach denen er in der Session Description sucht. Die konkreten Parser bekommen das SDP-Protokoll übergeben und können so die dort gespeicherten Informationen, wie die Parameter Sets, auswerten. Die Software ist so in der Lage, auf leistungsstärkeren Maschinen mehrere Codecs parallel auszuwerten.
Abbildung 68 zeigt das Klassendiagramm für die nicht Android spezifischen Klassen. Die Klasse H264Parser hat sehr viele Member die den Parametern und Prozessen des H264 Standards entsprechen. Diese sind aus Gründen der Übersichtlickeit nicht im Klassendiagramm enthalten, sondern nur die Member der selbst definierten Prozesse. Ebenso sind nicht alle Member der CAVLC-Klasse enthalten. Diese enthält als Member sehr viele Felder. Die Funktion dieser Felder wird im Folgenden erklärt.

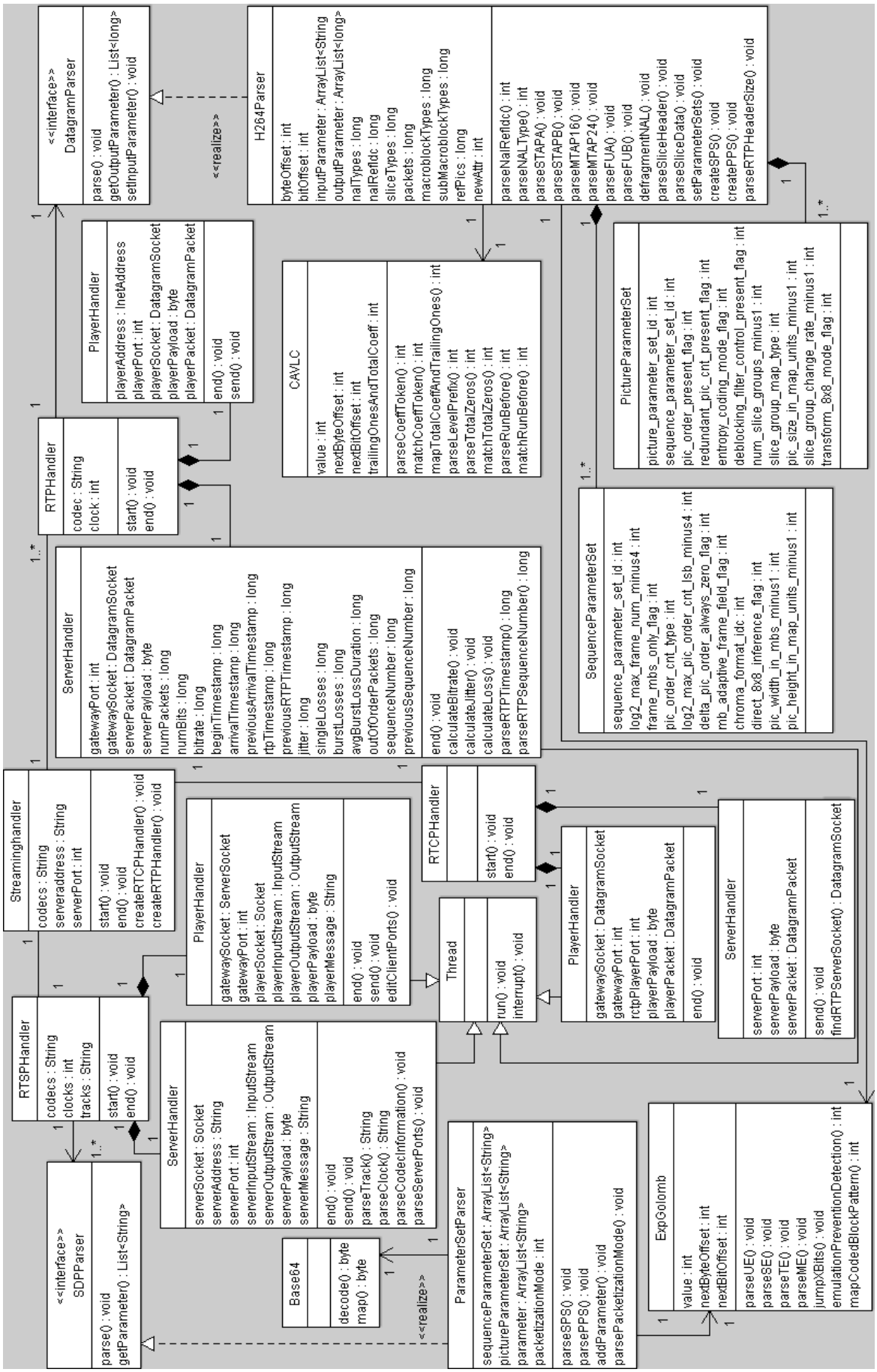

Abb. 68: Klassendiagramm

6.1.3.1 Die ProtokollHandler

Eine Übersicht über den Ablauf zur Erzeugung der RTSPHandler, RTPHandler und RTCPHandler beschreibt Abbildung 69. Der StreamingHandler erzeugt zunächst einen RTSPHandler mit seinen Komponenten ServerHandler und PlayerHandler, welche als Threads implementiert sind. Der MediaPlayer stellt eine Verbindung zum PlayerHandler her, die Verbindung zum Streaming Server stellt der ServerHandler her. Damit ist die RTSP Kommunikation zwischen MediaPlayer und Streaming Server möglich. Empfängt der PlayerHandler einen SETUP Request, manipuliert er die Portnummern und sendet das manipulierte SETUP an den Streaming Server. Den neu eingesetzten Portnummern entsprechend, erzeugt der PlayerHandler zwei RTPHandler. Die niedrigere Portnummer dient dem Transport der Audio- und Videodaten, die höhere Portnummer dient dem Transport der RTCP Reports vom Server zum MediaPlayer. Die ServerHandler der RTPHandler verrichten ihre Arbeit auch in Threads, während die PlayerHandler einfache Klassen sind. Empfängt der ServerHandler des RTSPHandler ein SETUP Response, so liest er die Port Nummern des Servers aus, wobei die höhere Portnummer für den Empfang der RTCP Reports eingesetzt wird. Anhand dieser Portnummer erzeugt der ServerHandler genau einen RTCPHandler, welcher die RTCP Pakete vom MediaPlayer in Richtung Server verarbeitet. Der PlayerHandler des RTCPHandlers ist als Thread implementiert, der ServerHandler als einfache Klasse.

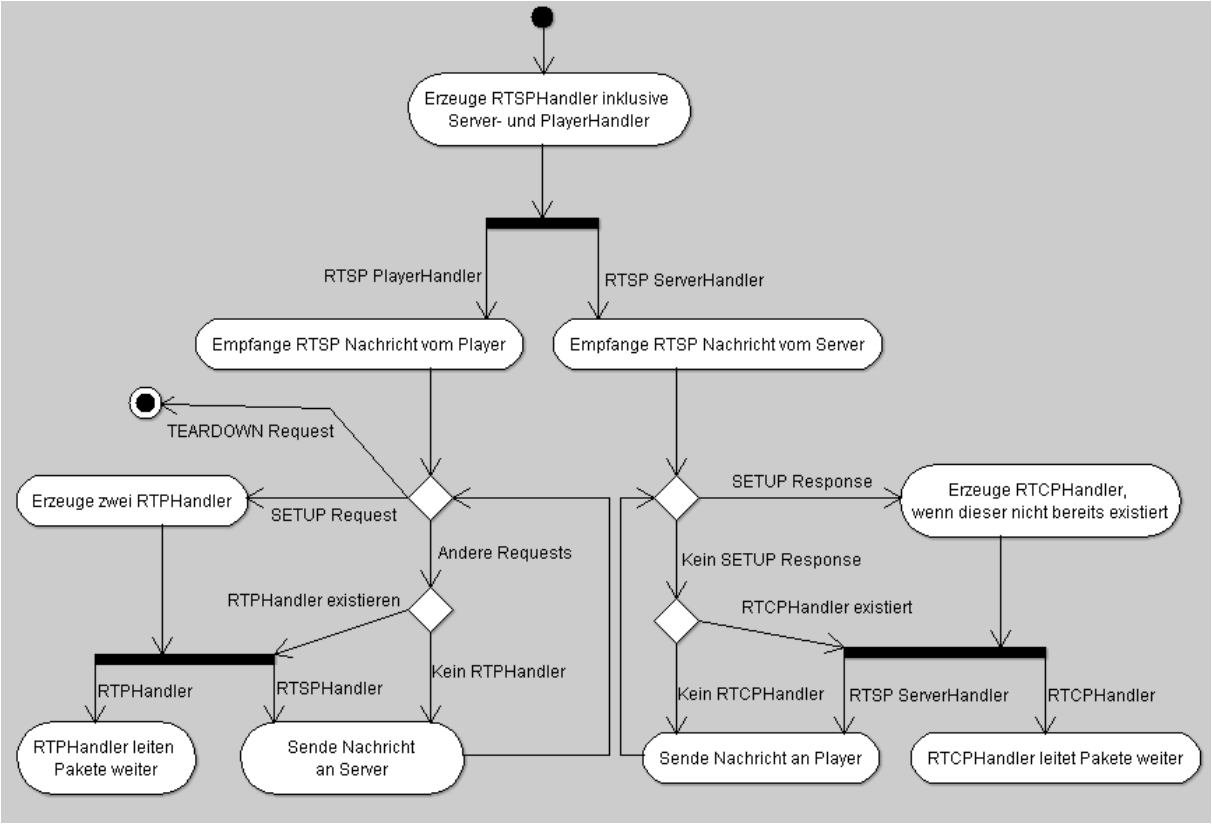

Abb. 69: Erzeugung der ProtokollHandler

6.1.3.1.1 Der RTSPHandler

Damit die Software leicht um neue Codecs erweiterbar ist, der Benutzer sich aber gleichzeitig nicht darum kümmern soll, welche Codecs tatsächlich genutzt werden, wird dem StreamingHandler bei seiner Erzeugung eine Liste von Codecs (genauer MIME-Types) übergeben, nach denen gesucht werden soll. Diese Liste wird an den RTSPHandler weitergereicht. Der ServerHandler des RTSPHandlers schaut im DESCRIBE Response nach, welche Codecs enthalten sind. Ist ein Codec aus

der Liste enthalten, wird für diesen die Tracknummer und die RTP Clock Reference eingelesen. Sind in der Session Description für den Codec wichtige Informationen enthalten, können ein entsprechender SDP Parser angelegt und die Informationen eingelesen werden.
Der PlayerHandler des RTSPHandlers durchsucht die SETUP Requests nach den eingelesenen Tracknummern. Kann eine Tracknummer einem Codec zugeordnet werden, wird ein RTPHandler mit dem entsprechenden Codec und der zugehörigen Clock Reference als Parameter erzeugt. Kann die Tracknummer im SETUP keinem Codec zugeordnet werden, wird dem RTPHandler der Wert null für den Codec und die Clock Reference übergeben. Der RTPHandler kann nun anhand des übergebenen Codec einen Parser erzeugen, wenn ein Parser für diesen Codec implementiert ist. Der Benutzer kann so das Video auswerten, ohne im Vorhinein wissen zu müssen, mit welchen Codecs das Video codiert wurde. Der zweite RTPHandler, der erzeugt wird, dient dem Transport der RTCP Reports in Richtung Player. Die Abbildungen 70 und 71 zeigen das Verhalten der Threads von ServerHandler und PlayerHandler des RTSPHandlers.

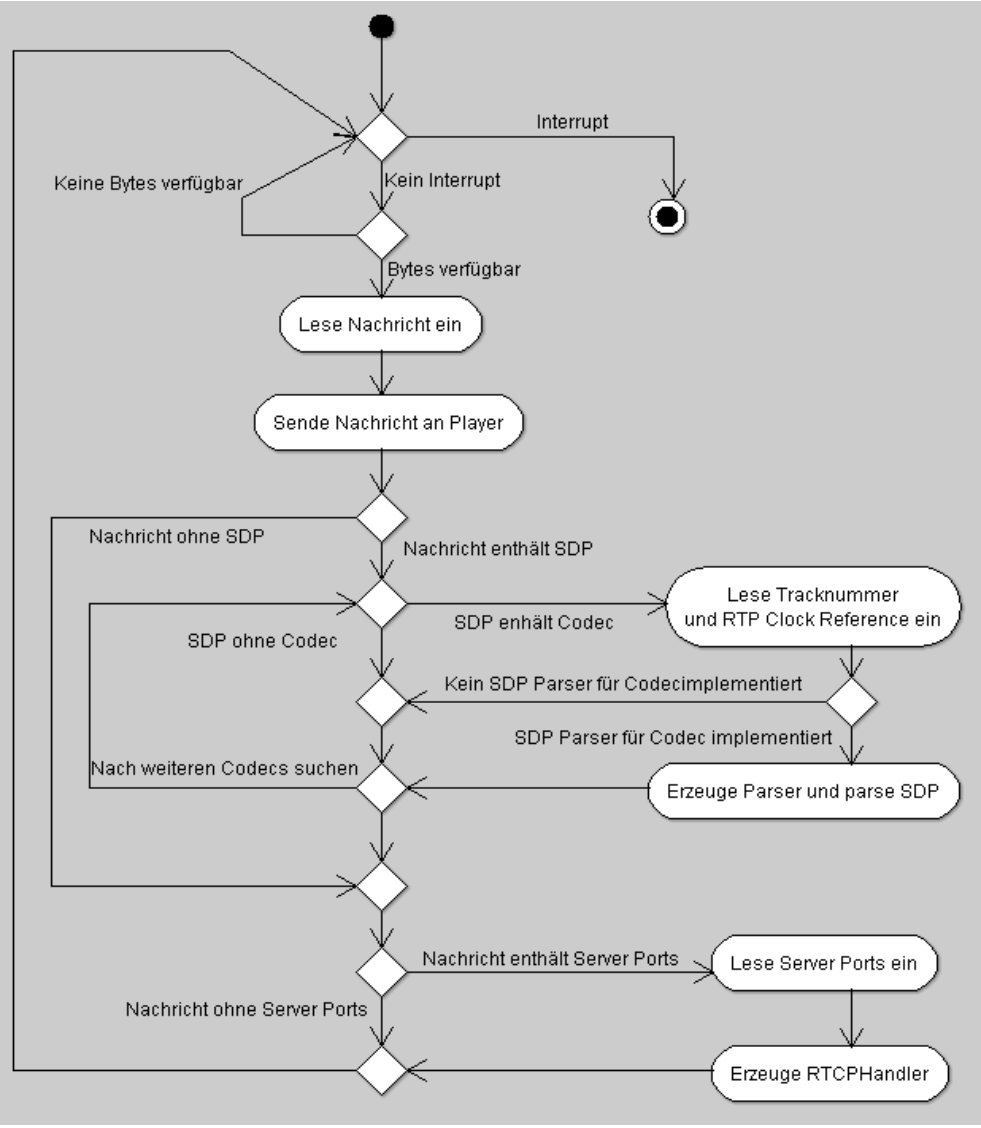

Abb. 70: Aktivitätsdiagramm des ServerHandler Threads

Das Weiterleiten des empfangenen Datenstroms geschieht innerhalb des ServerHandler unmittelbar nach dem Empfang. Im PlayerHandler kann dies jedoch erst am Ende eines Schleifendurchlaufs geschehen, da vor dem Weiterleiten, die SETUP Requests manipuliert werden müssen.

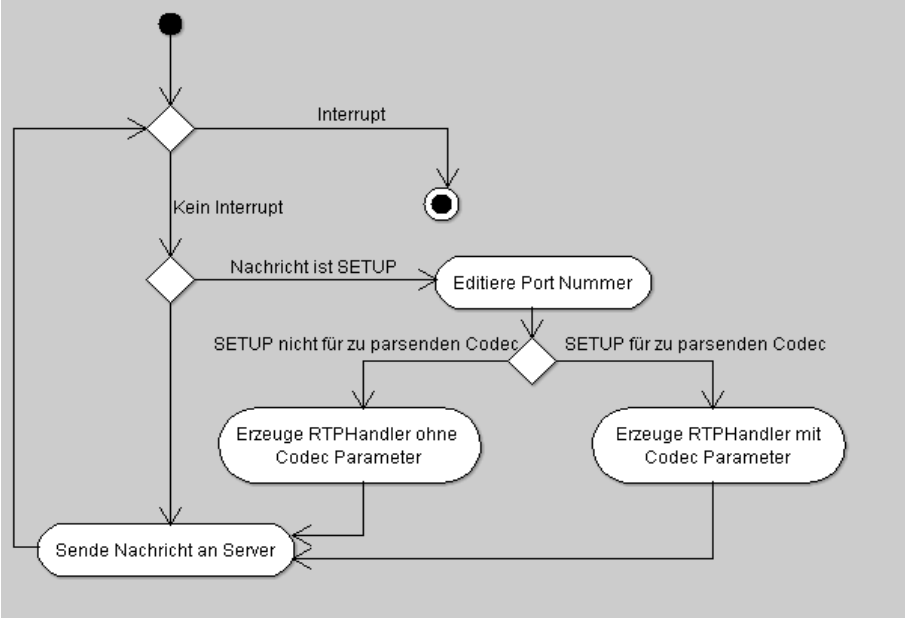

Abb. 71: Aktivitätsdiagramm des PlayerHandler Threads

6.1.3.1.2 Der RTPHandler

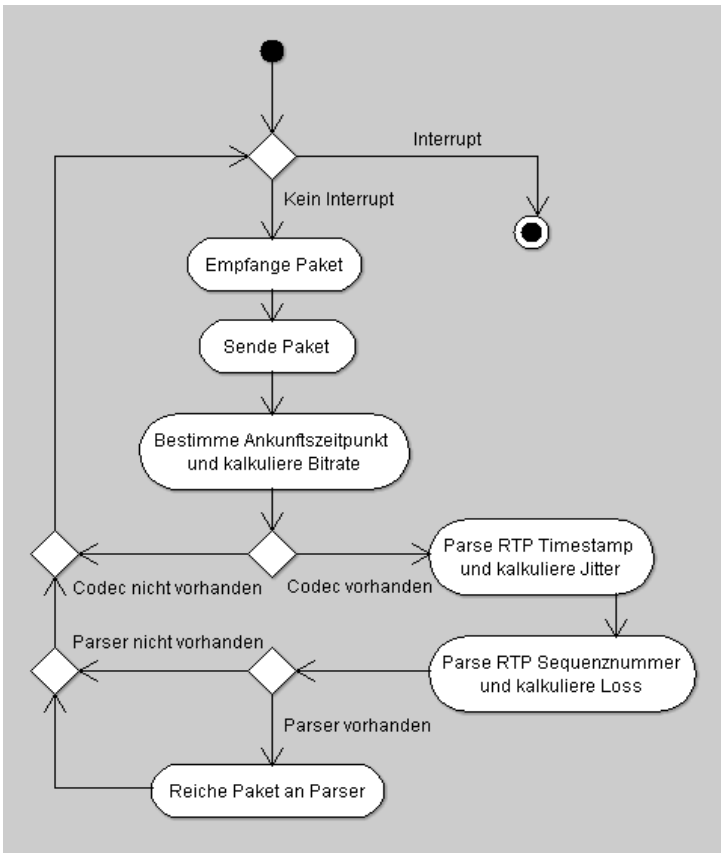

Abb. 72: Aktivitätsdiagramm des ServerHandler Threads

Die RTPHandler dienen dem Transport der einzelnen Tracks. Um die Daten vom Streaming Server zum MediaPlayer zu leiten, werden wieder ein ServerHandler und ein PlayerHandler eingesetzt. Jedoch ist nur der ServerHandler als Thread implementiert, da der PlayerHandler keine Daten von außerhalb der Anwendung empfängt. Der PlayerHandler repräsentiert nur einen Socket von dem aus die Daten zum MediaPlayer gesendet werden. Der ServerHandler übernimmt die Auswertung des RTP Header, d.h. die Bestimmung der Bitrate, des Jitters und der Anzahl an Paketen und verlorenen Pakete. Abbildung 72 zeigt das Aktivitätsdiagramm des ServerHandler Threads. Im Konstruktor des RTPHandler wird anhand des Codecs gegebenenfalls der entsprechende Parser erzeugt. Werden, wie bei H.264, dazu Parameter der Session Description benötigt, können diese über die Verbindung zum StreamingHandler aus dem Parser des RTSPHandler geholt werden. Die vom ServerHandler empfangenen Pakete werden an den PlayerHandler weitergeleitet und an den Parser übergeben.

6.1.3.1.3 Der RTCPHandler

Der RTCPHandler übernimmt das Weiterleiten der Receiver Reports für alle Tracks. Die Sender Reports werden über RTPHandler geleitet. Da die Pakete nur vom MediaPlayer in Richtung Streaming Server transportiert werden müssen, reicht es, den PlayerHandler als Thread zu implementieren, während der ServerHandler als einfache Klasse implementiert ist. Jedoch besitzt der ServerHandler keinen eigenen Socket. Der zum Testen verwendete Darwin Streaming Server akzeptiert Receiver Reports nur dann, wenn die Reports von dem Socket aus versandt werden, an den er selbst seine Sender Reports richtet. Das bedeutet, dass zum Senden der Receiver Reports der Socket des ServerHandler des RTPHandlers des jeweiligen Tracks genutzt werden muss. Anhand des Quellports des ankommenden Pakets identifiziert der RTCPHandler den PlayerHandler eines RTPHandler. Der ServerHandler dieses RTPHandlers beinhaltet den zum Senden nötigen Socket.

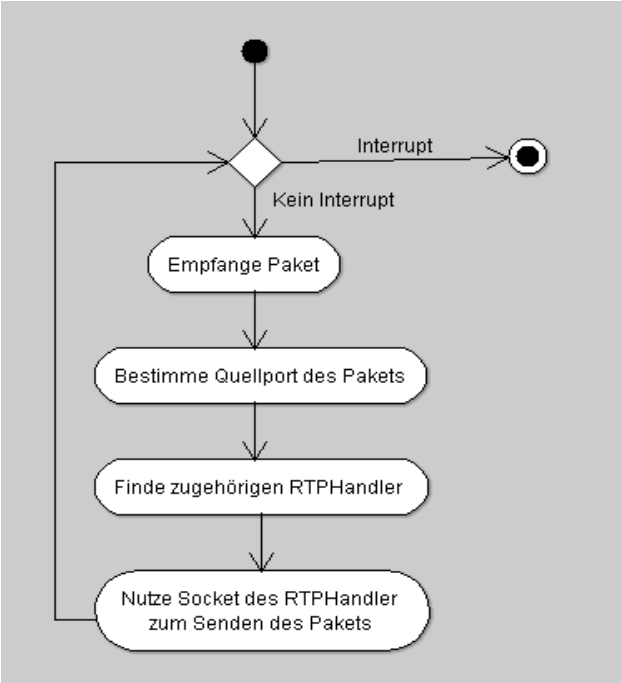

Abb. 73: Aktivitätsdiagramm des PlayerHandler Threads

6.1.3.2 Die Parser

Aus Abbildung 68 lässt sich erkennen, dass zwei Schnittstellen für Parser in der Softwarearchitektur enthalten sind. Eine Schnittstelle für die über RTP transportierten Datagramme, der DatagramParser,

und eine Schnittstelle für mögliche Metainformationen der Audio- und Videodaten, der SDPParser. Um H.264 codierte Videos zu parsen, ist es nötig, neben dem DatagramParser auch einen SDPParser zu implementieren, um an die Informationen des Parameter Sets zu gelangen. Für andere Codecs kann es ausreichen, den DatagramParser zu implementieren.

6.1.3.2.1 Der SDPParser und der ParameterSetParser

Implementierungen der SDPParser-Schnittstelle erhalten vom RTSPHandler das SDP-Protokoll zum Parsen. Da das SDP-Protokoll textbasiert ist, wird es als String übergeben. Die ausgewerteten Parameter können als Liste von Strings abgefragt werden.
Die H.264 spezifische Realisierung der SDPParser-Schnittstelle ist die Klasse ParameterSetParser. Der ParameterSetParser sucht zunächst die Zeile des SDP-Protokolls, in der die NAL Units der Parameter Sets stehen. Diese ist gemäß Kapitel 3.3.3.1 durch den Parameter „sprop-parameter-sets“ innerhalb des fmtp-Attributs gekennzeichnet. Nachdem die Parameter Sets innerhalb des SDP-Protokolls gefunden wurden, müssen die einzelnen NAL Units heraus gefiltert werden. Die gefilterten NAL Units müssen dann Base64 dekodiert werden. Das erste Byte der dekodierten NAL Unit ist der NAL Header, welcher angibt ob es sich um ein Sequence Parameter Set oder ein Picture Parameter Set handelt. Abhängig vom NAL Unit Type wird nun das Parameter Set geparst. Zur späteren Auswertung werden die Parameter in zwei Listen geschrieben, eine Liste für die Sequence Parameter Sets und eine Liste für die Picture Parameter Sets.

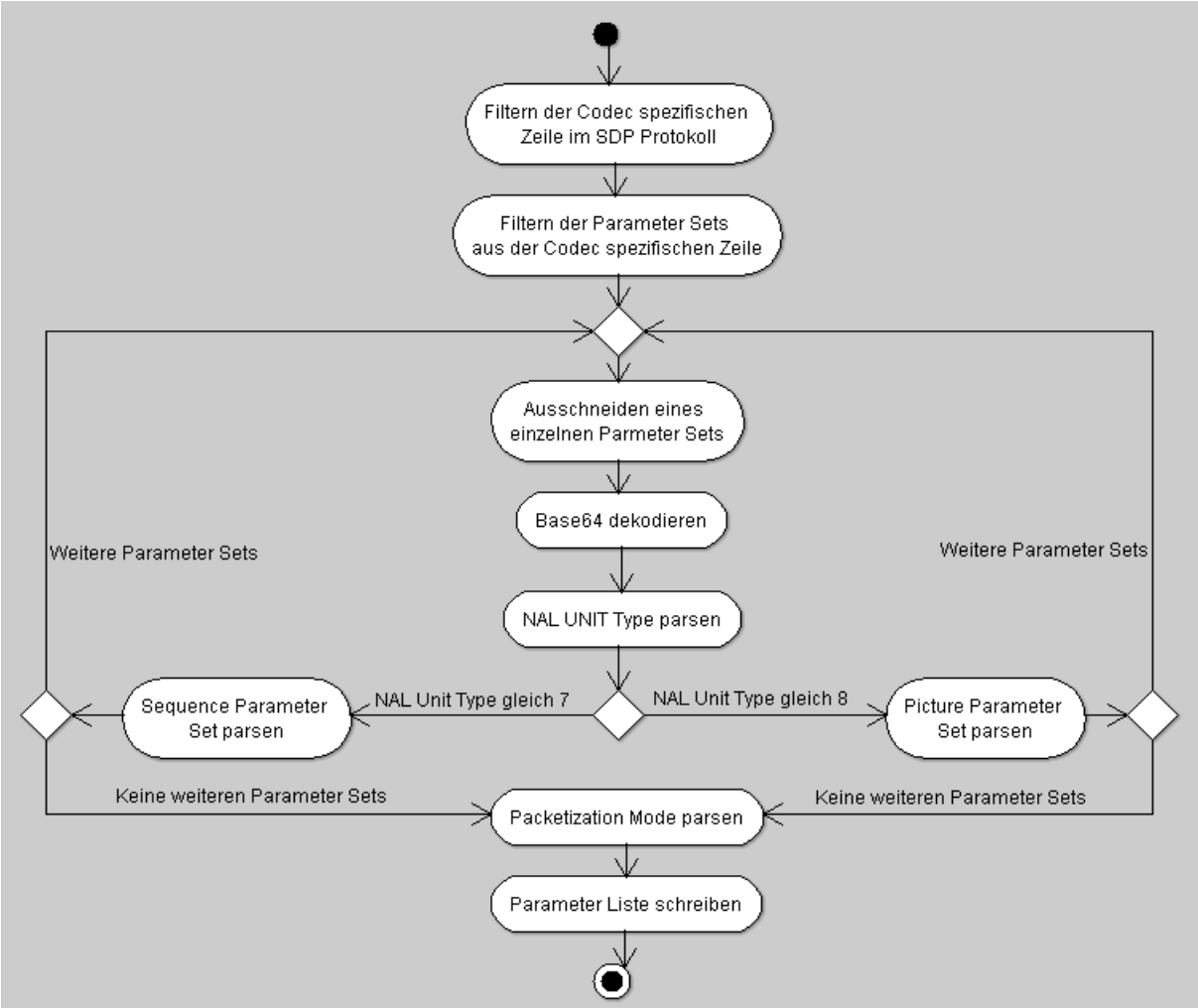

Abb. 74: Aktivitätsdiagramm des ParameterSetParser

Um den Parameter identifizieren zu können, wird neben dem Wert des Parameters auch sein Name festgehalten. Der Name des Parameters wird in spitzen Klammern festgehalten, gefolgt vom Wert des Parameters:

<Parameter>Wert

Ein Listeneintrag für das Profil hat beispielweise die Form:

<profile_idc>66

Um verschiedene Parameter Sets des gleichen Typs zu trennen, wird am Anfang eines Parameter Sets entweder <sequence_parameter_set> oder <picture_parameter_set> in die Liste geschrieben. Aus den Listen für die Sequence Parameter Sets und die Picture Parameter Sets werden nach dem Parsen die zur Auswertung verwendeten Parameter herausgesucht und in eine gemeinsame Liste geschrieben. Die Einträge in dieser Liste haben jedoch eine ausdruckstärkere Form. Beispielsweise steht als erster Eintrag das Profil in dieser Liste. Statt den Werten 66, 77 oder 88 wird jedoch Baseline, Main oder Extended in der Liste eingetragen. Nach den Parametern, die zur Auswertung benötigt werden, werden die kompletten Parameter Sets an diese Liste angehängt. Diese Liste kann der DatagramParser abfragen und somit die Parameter Sets rekonstruieren.
Die Parameter Sets bestehen aus binär codierten Parametern und Exp-Golomb codierten Parametern. Deshalb besitzt die Klasse ParameterSetParser eine Assoziation zu der Klasse ExpGolomb. Wie die zum Parsen eines Exp-Golomb codierten Parameter notwendigen Schritte implementiert sind, wird in Kapitel 6.1.3.3 erläutert. Der syntaktische Aufbau der Parameter Sets steht in den Kapiteln 7.3.2.1 und 7.3.2.2 des H.264 Standards. Das Parsen der Parameter Sets entspricht diesem Aufbau.

6.1.3.2.2 Der DatagramParser und der H264Parser

Die Implementierungen der DatagramParser-Schnittstelle erhalten von dem mit Ihnen assoziiertem RTPHandler die Datagramme, die sie auswerten sollen. Der Übergabetyp ist ein DatagramPacket. Das Datagram beinhaltet dabei noch den RTP Header, jedoch nicht mehr den UDP Header.
Die H.264 spezifische Implementierung des DatagramParser ist die Klasse H264Parser. Der H264Parser bekommt als Inputparameter die vom ParameterSetParser erzeugte Liste mit den Parameter Sets. Aus dieser Liste generiert der H264Parser die entsprechende Anzahl an Sequence Parameter Sets und Picture Parameter Sets. Als Outputparameter erzeugt der H264Parser eine Liste von Long-Feldern. In diesen Feldern werden beispielweise die Anzahl an NAL Unit Types, Slice-Types und Macroblock-Typen gespeichert.
Zunächst wertet der Parser den CRSC count, das Extension-Flag und gegebenenfalls die Extension-Größe innerhalb des RTP Headers aus, um dessen gesamte Größe zu bestimmen. Dies ist nötig, um das erste Byte der NAL Unit bzw. den Packet Type zu identifizieren. Die zur Bewertung der Qualität genutzten Parameter des RTP Headers werden vom RTPHandler ausgewertet.
Ist die Größe des RTP Header bestimmt, kann für jedes Paket der NAL Header bzw. der Packet Type ausgewertet werden. Es wird also das erste Byte nach dem RTP Header ausgelesen. Ist das Paket ein Single NAL Unit Packet, können die eingelesenen Daten direkt für die Statistik verwendet werden. Für Aggregation Packets muss in einer Schleife gemäß dem in Kapitel 3.3.2 beschriebenen Aufbau des Aggregation Paket jede NAL Unit im Paket geparst werden. Im Falle einer Fragmentation Unit wird das Fragment aus dem Paket extrahiert und an ein Feld angehängt. Dieses Feld wird bei Ankunft einer Fragmentation Unit mit gesetztem Start-Flag neu angelegt. Trifft ein Paket mit gesetztem End-Flag ein, wird nach dem Anhängen des letzten Fragments die NAL Unit geparst.
Die NAL Unit Types 1 und 5 enthalten Slice Header und Slice Daten, der NAL Unit Type 2 enthält nur Slice Header, d.h. diese Typen müssen detaillierter betrachtet werden. Das Parsen der Slice Daten kann nur auf leistungstärkeren Maschinen vorgenommen werden, da der Rechenaufwand sehr groß ist. Insbesondere muss hierfür das Defragmentieren der Fragmentation Units stattfinden, d.h. es sind viele Schreiboperationen notwendig, die zeitaufwendig sind. Werden die Slice Daten nicht geparst, sondern nur die Slice Header, entfällt das Defragmentieren der NAL Unit. Der Slice Header steht im ersten Fragment, welches durch das Start-Flag gekennzeichnet ist.

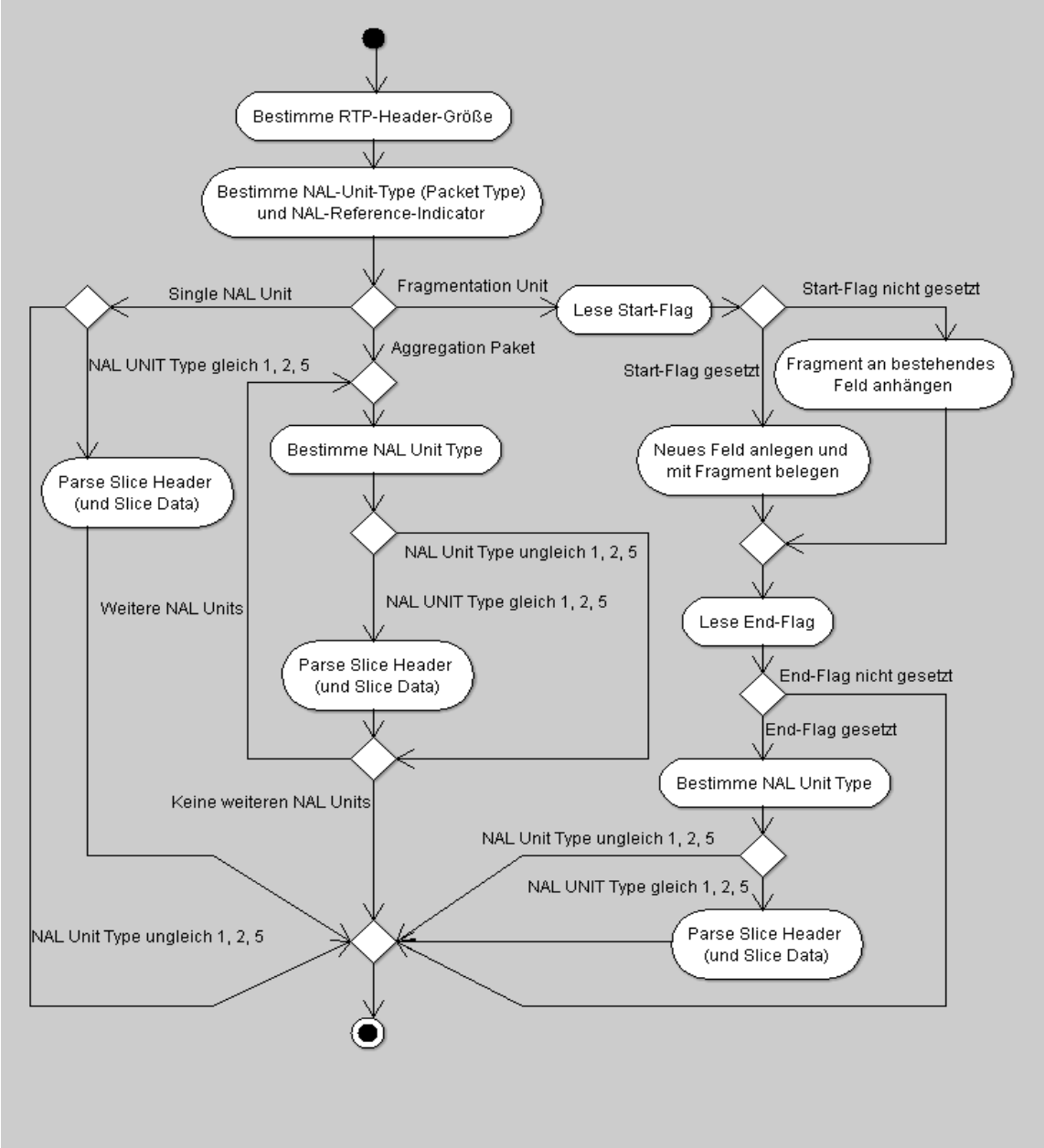

Abb. 75: Aktivitätsdiagramm des H264Parsers

6.1.3.3 Die Entropiecodierer

Wie in Kapitel 2.2.1.4 beschrieben, kommen bei der Codierung im Baseline Profil zwei Entropiecodierer zum Einsatz: die Exp-Golomb-Codierung und die Context-Adaptive-Variable-Length-Codierung.

6.1.3.3.1 Exp-Golomb-Coding

Da alle Varianten einfache Ableitungen der Unsigned-Exp-Golomb-Codierung sind, soll das Parsen der Exp-Golomb-Codierten Parameter nur für eben diese detailliert erörtert werden.
In einer Schleife wird die Anzahl an führenden Nullen bestimmt. Mit jedem Schleifendurchlauf wird ein Bitzähler und ggf. ein Bytezähler erhöht. Wird der Bytezähler erhöht, ist es nötig zu prüfen, ob ein Emulation Prevention Byte vom Encoder eingefügt wurde. Ist dies der Fall, muss das Prevention Byte verworfen werden, d.h. der Bytezähler wird nocheinmal um Eins erhöht. Nachdem das erste Bit mit

Wert Eins entdeckt wurde, werden in einer Schleife genauso viele Bits eingelesen wie führende Nullen eingelesen wurden. In der Schleife werden wieder der Bitzähler und ggf. der Bytezähler erhöht. Auch die Prüfung auf ein Emulation Prevention Byte ist wieder nötig. Die eingelesenen Bits beschreiben den Wert in Binärcodierung. Nachdem alle Bits eingelesen wurden, wird der Wert $2^{(\text{Anzahl führende Nullen})} - 1$ zum eingelesenen Wert hinzu addiert.

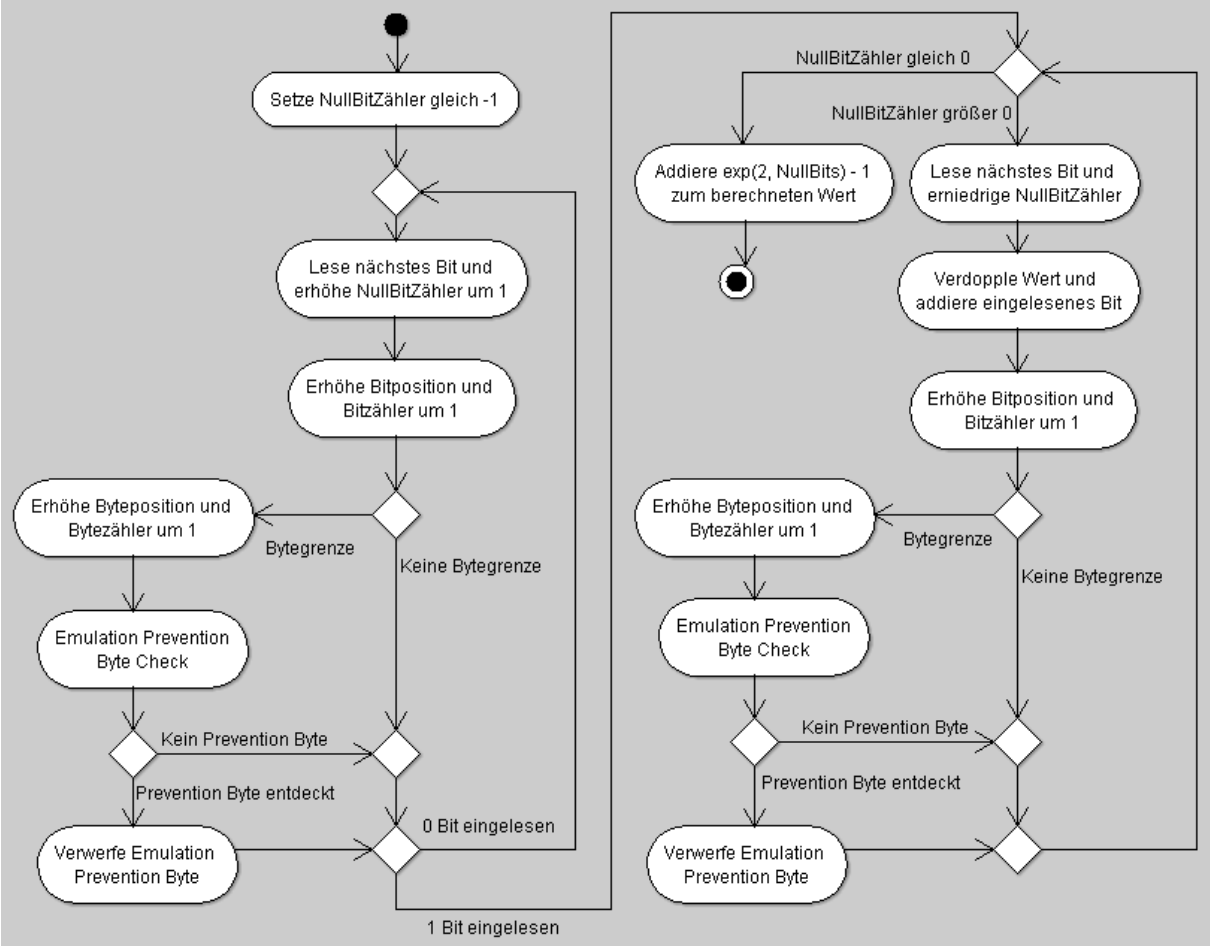

Abb. 76: Aktivitätsdiagramm Unsigned-Exp-Golomb-Dekodierung

6.1.3.3.2 Context-Adaptive-Variable-Length-Coding

Die Klasse CAVLC enthält Methoden zum Parsen der Parameter coeff_token, level_prefix, total_zeros und run_before. Die Erläuterung dieser Parameter steht in Kapitel 2.2.1.4.2. Mit Ausnahme des level_prefix werden die Parameter durch Codeworte verschiedener Länge repräsentiert. Softwaretechnisch erfolgt die Abbildung der Codeworte durch ihre Länge und ihren binären Wert. Das Codewort **0000 0101 1** wird beispielsweise durch die Länge 9 und den Wert 11 identifiziert.
Zum Parsen des coeff_token werden in einer Schleife die führenden Nullen eingelesen. Mit jedem Schleifendurchlauf wird eine Variable, die die aktuelle Länge notiert, hochgezählt. Nachdem das erste Bit mit dem Wert Eins entdeckt wurde, wird in einer Liste nachgeschaut, ob das Codewort der entsprechenden Länge und mit dem eingelesenen Wert existiert. Existiert es, wird der Wert, den das Codewort repräsentiert, zurückgegeben. Existiert das Codewort nicht, so werden weitere Bits eingelesenen bis ein existierendes Codewort gefunden wird. Um den Suchaufwand gering zu halten, sind die Codewörter in verschiedene Listen eingeteilt. Für den Parameter coeff_token existieren dazu 51 Listen. Die Zuteilung der Codewörter in eine Liste erfolgt dabei über den in Kapitel 2.2.1.4.2 beschriebenen Wert nC sowie die Länge des Codewortes. Das oben beschriebene Codewort hat die Länge 9. Laut Tabelle 9.5 des H.264 Standards existiert für die nC Werte 4-7 ein Codewort der Länge

9 mit dem Wert 11. Dieses Codewort findet sich also in der Liste **ct_9_nc4567**. Diese Liste beinhaltet alle coeff_token (ct) Codewörter der Länge 9, die bei einem nC Wert von 4-7 auftreten können. Zum Parsen des obigen CoeffToken geschieht folgendes:

- Der Wert nC wird anhand der umgebenden 4x4 Blöcke berechnet. Nehmen wir an, er wird als 4 berechnet.
- Die Funktion parseCoeffToken liest fünf Bits mit dem Wert 0 ein, danach ein Bit mit dem Wert 1, d.h. das aktuelle Codewort hat die Länge 6, den Wert 1 und liegt im nC-Bereich 4-7
- Die Funktion parseCoeffToken überprüft Liste **ct_6_nC4567** nach einem Codewort mit dem Wert 1, dieses existiert nicht.
- Ein weiteres Bit mit dem Wert 0 wird eingelesen, das aktuelle Codewort hat nun die Länge 7 und den Wert 2.
- Die Funktion parseCoeffToken überprüft Liste **ct_7_nc4567** nach dem Codewort und findet es nicht.
- Ein weiteres Bit mit dem Wert 1 wird eingelesen, das Codewort hat die Länge 8 und den Wert 5.
- Die Funktion parseCoeffToken überprüft Liste **ct_8_nc4567** nach dem Codewort und findet es nicht.
- Ein weiteres Bit mit dem Wert 1 wird eingelesen, das aktuelle Codewort hat die Länge 9 und den Wert 11.
- Die Funktion parseCoeffToken überprüft Liste **ct_9_nc4567** nach dem Codewort und findet es. Der zugehörige Eintrag in der Liste gibt an, dass das Codewort in Zeile 38 von Tabelle 9.5 steht.
- Durch Nachschauen in der Liste **trailingOnesAndTotalCoeffs** werden die gesuchten Parameter, TrailingOnes und TotalCoeff, mit den Werten 0 und 11 bestimmt.

Der Parsingprozess für den Parameter total_zeros ist analog aufgebaut. Als Übergabeparameter des H264Parser dienen hier statt nC der Parameter maxNumCoeff, welcher angibt, wieviele Koeffizienten maximal in einem Block vorkommen können, also beispielsweise maximal 4 Koeffizienten in einem 2x2 Chroma Block, und der Parameter TotalCoeff. Da auch das Codewort 0 vorkommen kann, wird auf die erste Schleife zum Einlesen der führenden Nullen verzichtet. D.h. sofort nach dem ersten Schleifendurchlauf wird geprüft, ob es ein Codewort der entsprechenden Länge mit dem eingelesenen Wert gibt. Die Listen enthalten diesmal die Länge der Codeworte und ihren Wert. Die Einteilung in die Listen erfolgt über die beiden Übergabeparameter. Auch hier soll der Parsingprozess an einem Beispiel demonstriert werden:

- Es wird angenommen, dass es sich um einen Luma Block handelt. Da ein Luma Block immer 4x4 Samples groß ist, hat der Parameter maxNumCoeff den Wert 16. Der Luma Block soll tatsächlich sechs Koeffizienten ungleich Null enthalten, d.h. TotalCoeff hat den Wert 6.
- Die Funktion parseTotalZeros liest ein Bit mit dem Wert 1 ein, das Codewort hat die Länge 1 und den Wert 1.
- TotalCoeff ist gleich 6 und maxNumCoeff gleich 16, deshalb wird die Liste **tz_6_maxNumCoeffX** zum Vergleich genutzt. Das obige Codewort wird in dieser Liste nicht gefunden.
- Ein weiteres Bit mit Wert 0 wird eingelesen, das Codewort hat die Länge 2 und den Wert 2.
- Da sich TotalCoeff und maxNumCoeff nicht ändern, wird die gleiche Liste wie beim ersten Durchgang zum Vergleich benutzt. Auch diesmal wird das Codewort nicht gefunden.
- Ein weiteres Bit mit Wert 1 wird eingelesen, das Codewort hat die Länge 3 und den Wert 5.
- Es wird wieder in der obigen Liste nachgesehen. Dieses Codewort steht an sechster Stelle in der Liste, der Index hat also den Wert 5. Der Listenindex gibt gleichzeitig den Wert des Parameters total_zeros an.

Der Parsingprozess für den Parameter run_before ist eine einfachere Form der obigen Parsingprozesse. Einziger Übergabeparameter des H264Parsers ist die Variable zerosLeft. Hier existieren sieben kleine Listen mit kurzen Codewörtern. Unmittelbar nach dem Einlesen eines Bits wird in einer Liste, die durch die Variable zerosLeft bestimmt wird, nachgeschaut, ob das Codewort existiert. Die Codeworte werden wieder über die Länge und den Wert identifiziert.

Der Parsingprozess für den level_prefix ist sehr einfach. Der level_prefix wird durch eine Serie von Bits mit dem Wert 0, gefolgt von genau einem Bit mit dem Wert 1, repräsentiert. Der Wert des level_prefix entspricht der Anzahl an führenden Nullen.

6.1.3.4 Die Activities

Beim Starten der Anwendung wird die StartActivity ausgeführt. Diese besitzt ein einfaches User Interface. Das User Interface besteht aus zwei EditBoxen zum Einlesen eines Sockets und eines Dateinamens, sowie drei Buttons zum Starten der Wiedergabemodi und zum Beenden. Beim Starten der Wiedergabe wird über einen expliziten Intent eine der Activities StatsActivity oder VideoActivity gestartet. Die eingelesenen Daten werden in einem Bundle dem Intent mitgegeben.
Die beiden Activities, die durch den Intent angesprochen werden können, besitzen einen StreamingHandler als Member sowie eine VideoView zur Darstellung des Videos. Aus dem Bundle werden der Hostname, der Port und der Filename entnommen. Hostname und Port werden dem StreamingHandler übergeben, damit dieser eine Verbindung zum Streaming Server aufbauen kann. Der StreamingHandler akzeptiert TCP-Verbindungen des MediaPlayers auf Port 8554. Die Portnummer musste so gewählt werden, dass sie nicht im Bereich der Well Known Ports liegt, also größer als 1024 ist. Android basiert auf einem Linux-Kernel, so dass keine Sockets im Bereich der Well Known Ports geöffnet werden können. Die VideoView stellt eine Verbindung zu Port 8554 her und fragt per RTSP das im Bundle übergebene File an. Ab diesem Punkt läuft die Kommunikation zwischen Streaming Server und MediaPlayer, wie in den vorigen Kapiteln dargestellt.
Die Darstellung der Messergenisse in detaillierter Form oder über die Ampel läuft nach dem gleichen Schema ab. Wann die Messergebnisse aktualisiert werden, wird durch einen eigenen Thread geregelt. In diesem Thread läuft ein Timer, der vorgibt, in welchen Intervallen die Messergebnisse geupdatet werden. Das Neuzeichnen einer View darf nur von dem Thread durchgeführt werden, der die View auch erzeugt hat. Deshalb wird der Thread, in dem das User Interface erzeugt wurde, über einen Handler von dem Thread mit dem Timer informiert, wenn das Neuzeichnen erfolgen soll.

6.2 Die Implementierung

6.2.1 Die ProtokollHandler

6.2.1.1 Der RTSPHandler

Der PlayerHandler des RTSPHandler nimmt den Verbindungswunsch des MediaPlayer entgegen. Dazu muss der PlayerHandler einen ServerSocket öffnen. Hat der PlayerHandler den Verbindungswunsch akzeptiert, so repräsentiert ein Socket die Verbindung zum MediaPlayer. Um sicherzustellen, dass die Verbindung nicht aufgrund von Timeouts auf der Transportschicht abbricht, sendet der PlayerHandler TCP-Keep-Alives. Ein TCP-Keep-Alive ist ein einzelnes TCP Paket ohne Nutzdaten, aber mit gesetztem ACK-Flag. Steht die Verbindung noch, antwortet die Gegenseite ebenfalls mit einem ACK.

```
gatewaySocket = new ServerSocket(8554);
playerSocket = gatewaySocket.accept();
playerSocket.setKeepAlive(true);
```

Der `gatewaySocket` akzeptiert die Verbindung und gibt den Socket `playerSocket` zurück, der nun die Verbindung zum MediaPlayer repräsentiert.
Die Verbindung zum Streaming Server richtet der ServerHandler ein, indem er selbst einen Socket öffnet. Vom StreamingHandler werden die Serveradresse und der Port an den RTSPHandler übergeben, so dass sie dem ServerHandler bekannt sind. Die Bezeichnung des geöffneten Sockets ist `serverSocket`. Dies soll hervorheben, dass der Socket die Verbindung zum Server repräsentiert. Der Objekttyp ist jedoch ein Socket, kein ServerSocket. Auch der ServerHandler sendet TCP-Keep-Alives an den Streaming Server.

```
serverSocket = new Socket(serverAddress, serverPort);
serverSocket.setKeepAlive(true);
```

Das Auslesen und Senden der Daten geschieht über InputStreams und OutputStreams. Dies passiert bei ServerHandler und PlayerHandler ganz analog. Sind die Sockets erzeugt, kann jeweils ein InputStream und ein OutputStream dem Socket entnommen werden.

```
serverInputStream = serverSocket.getInputStream();
serverOutputStream = serverSocket.getOutputStream();
```

Aus dem InputStream werden in einer Schleife die verfügbaren Bytes entnommen und in einen String umgewandelt, da das RTSP-Protokoll textbasiert ist. Dieser String kann dann ausgewertet werden. Analog werden zum Senden die Strings wieder in Bytes umgewandelt.

```
serverPayload = new byte[serverInputStream.available()];
serverInputStream.read(serverPayload);
serverMessage = new String(serverPayload);
```

6.2.1.2 Die RTPHandler

Da der ServerHandler des RTPHandler UDP Pakete empfängt, öffnet er einen DatagramSocket. Die Portnummer, auf der der Socket geöffnet wird, ist der `gatewayPort`, welcher der Methode `createRTPHandler` übergeben wurde. Im Gegensatz zu den Komponenten des RTSPHandler wird kein Inputstream empfangen, sondern Objekte vom Typ DatagramPacket. Die maximale Größe der DatagramPackets wird praktisch durch die Größe von Ethernet-Frames festgelegt, da es üblich ist, ein UDP Paket in einem IP Paket und dieses wiederum in einem Ethernet Frame zu transportieren. Ethernet Frames können einen Payload von 1500 Bytes transportieren. Abzüglich des IP Headers von 20 Byte und des UDP Headers von 8 Byte, können also maximal 1472 Byte Nutzdaten in einem UDP Paket transportiert werden.

```
gatewaySocket = new DatagramSocket(gatewayPort);
serverPayload = new byte[1472];
serverPacket = new DatagramPacket(serverPayload,
                                  serverPayload.length);
```

Innerhalb einer Schleife werden die Pakete durch die Methode `receive` empfangen. Diese Methode blockiert, bis ein Paket an ihrem Socket eintrifft. Die Länge des Pakets muss in jedem Schleifendurchlauf neu gesetzt werden, da ansonsten die Länge des vorherigen Pakets als maximale Länge dient.

```
serverPacket.setLength(1472);
gatewaySocket.receive(serverPacket);
```

Auf Seiten des PlayerHandlers wird ein unbestimmter Socket geöffnet, d.h. der Socket besitzt keine Portnummer. Das DatagramPacket muss zum Versand mit einer Zieladresse und einem Zielport versehen werden. Die Adresse ist vom Typ InetAddress, der Port vom Typ int.

```
playerAddress = InetAddress.getByName("localhost");
playerPort = playerPort;
playerSocket = new DatagramSocket();
playerPayload = new byte[1472];
playerPacket = new DatagramPacket(playerPayload,
                                  playerPayload.length,
                                  playerAddress,
                                  playerPort);
```

6.2.1.3 Der RTCPHandler

Der PlayerHandler des RTCPHandler ist analog zum ServerHandler des RTPHandler aufgebaut, d.h. er besitzt einen `gatewaySocket`, auf dem die UDP Pakete empfangen werden. Von diesem aus werden die Pakete an den ServerHandler weitergeleitet. Der ServerHandler muss jedoch erst den richtigen Socket zum Versenden suchen, da der Streaming Server sonst eventuell die RTCP Pakete nicht korrekt zuordnet. Deshalb liest der PlayerHandler den Port `rtcpPlayerPort`, von dem das RTCP Paket versendet wurde. Die Methode `findRTPServerSocket` des ServerHandlers ordnet dem `rtcpPlayerPort` den Socket zu, von dem das Paket versendet werden muss.

```
public DatagramSocket findRTPServerSocket(int rtcpPlayerPort)
{
	for(RTPHandler rtpHandler:streamingHandler.getRTPHandlerList())
	{
		if(rtpHandler.getPlayerHandler().getPlayerPortNumber()
							== rtcpPlayerPort)
		{
			return rtpHandler.getServerHandler().getSocket();
		}
	}
	return null;
}
```

6.2.2 Die Parser

Der SDPParser und der DatagramParser sind einfache Java Interfaces, d.h. es werden nur abstrakte Funktionsprototypen definiert.

6.2.2.1 Der ParameterSetParser

Der erste Schritt zum Parsen der Parameter Sets ist das Extrahieren der Sets aus dem SDP-Protokoll. Der Beginn der Parameters Sets wird durch das Attribut sprop-parameter-sets gekennzeichnet, das Ende eines Attributs wird durch das Zeilenende oder ein Semikolon gekennzeichnet, wenn weitere Attribute folgen. In der String-Variable `sdp` wird das SDP-Protokoll übergeben, `help` ist ebenfalls eine String Variable zum Zwischenspeichern. `begin` und `endAttribut` sind Integer Variablen zum Markieren der Position innerhalb eines Strings. Zunächst werden sämtliche Zeichen vor den Parameter Sets abgeschnitten. Als zweiter Schritt werden alle Zeichen hinter dem Zeilenumbruch verworfen. Beinhaltet der verbliebene String nun noch ein Semikolon, werden auch alle Zeichen hinter dem Semikolon und das Semikolon selbst verworfen.

```
begin = sdp.indexOf("sprop-parameter-sets") + 21;
help = sdp.substring(begin, sdp.length() - 1);
endAttribut = help.indexOf("\n") - 1;
help = help.substring(0, endAttribut);
if(help.contains(";"))
{
	endAttribut = help.indexOf(";");
	help = sdp.substring(begin, endAttribut);
}
```

Die einzelnen Parameter Sets sind nun in der Variable `help` gespeichert und durch Kommata getrennt. In einer Schleife werden dann die Parameter Sets ausgeschnitten, Base64 dekodiert und geparst.

```
while(help.contains(","))
{
	end = help.indexOf(",");
	String parameterSet = help.substring(0, end);
	byte[] ps = base64.decode(parameterSet);
	if((ps[0] & 0x1f) == 7)
```

```
            parseSPS(ps);
        if((ps[0] & 0x1f) == 8)
            parsePPS(ps);
        help = help.substring(end + 1, help.length());
    }
```

Welche Art von Parameter Set vorliegt, wird dem NAL Unit Header, also dem ersten Byte entnommen. Die Methoden `parseSPS` und `parsePPS` entsprechen im Wesentlichen dem syntaktischen Aufbau aus dem H.264 Standard. Entsprechend dem Syntax Element erfolgt der Aufruf des Entropiecodierers und ein Eintrag in die zugehörige Liste.

```
expGolomp.parseUE(sps, 4, 0);
sequenceParameterSet.add("<sequence_parameter_set_id>" +
                                                expGolomp.getValue());
```

`sps` ist das an die `parseSPS` Methode übergebene Bytefeld. Die beiden Integerparameter entsprechen der aktuellen Byteposition innerhalb des Bytefeldes sowie der aktuellen Bitposition innerhalb des aktuellen Byte.

6.2.2.2 Der H264Parser

Als erstes durchläuft jedes Datagram die `parse` Methode des H264Parsers. Hier wird die Größe des RTP Header bestimmt sowie der NAL Unit Type bzw. der Packet Type. Abhängig vom NAL Unit Type bzw. dem Packet Type wird ein entsprechender Mechanismus zum Parsen angewandt.

```
public void parse(DatagramPacket packet)
{
    headerSize = parseRTPHeaderSize(packet.getData());
    nalOrPacketType = parseNALType(packet.getData()[headerSize]);

    switch(nalOrPacketType)
    {
        case 1:     nal_unit_type = nalOrPacketType;
                    nalTypes[nal_unit_type]++;
                    parseSliceHeader(packet.getData(),
                                     headerSize + 1);
                    parseSliceData(packet.getData(),
                                   this.byteOffset,
                                   this.bitOffset);
                    break;
        ...

        case 24:    parseSTAPA(packet);
                    break;
    }
}
```

Single NAL Units können direkt ausgwertet werden, während Aggregation Packets und Fragmentation Units eines weiteren Zwischenschritts benötigen.

```
public void parseSTAPA(DatagramPacket packet)
{
    int size, offset = headerSize + 1;
    byte[] rtpPacket = packet.getData();

    while(offset < packet.getLength())
    {
        nalTypes[24]++;
        size = (((rtpPacket[offset] & 0xff) << 8) |
                                (rtpPacket[offset + 1] & 0xff));
```

```
                nal_unit_type = parseNALType(rtpPacket[offset + 2]);
                nalTypes[nal_unit_type]++;
                if(nal_unit_type == 1 || nal_unit_type == 5)
                {
                      parseSliceHeader(rtpPacket, offset + 3);
                      parseSliceData(packet.getData(),
                                    this.byteOffset,
                                    this.bitOffset);
                }
                offset += (size + 2);
          }
    }
```

Im Wesentlichen laufen hier die gleichen Schritte wie beim Parsen der Single NAL Unit Packets ab. Im Falle der Aggregation Packets jedoch innerhalb einer Schleife, die dem Aufbau des Aggregation Packet entspricht. Im Falle der Fragmentation Units muss die NAL Unit zunächst wieder zusammen gebaut und dann geparst werden.
Das Parsen der Syntax Elemente von Slice Header und Slice Data funktioniert analog zum Parsen der Parameter Sets. Jedoch ist die syntaktische Struktur deutlich komplexer und die Parameter sind voneinander abhängig. Deshalb werden die Parameter als Member der Klasse aufgenommen, um ein schnelles Verarbeiten zu ermöglichen. Als Beispiel soll hier der Parameter `slice_type` dienen. Abhängig vom `slice_type` können verschiedene andere Syntax Elemente in Substrukturen vorkommen.

```
expGolomb.parseUE(rtpPacket,
                  expGolomb.getNextByteOffset(),
                  expGolomb.getNextBitOffset());
slice_type = expGolomb.getValue();

...

if(slice_type == 1 || slice_type == 6)
{
      expGolomb.jumpXBits(rtpPacket,
                          expGolomb.getNextByteOffset(),
                          expGolomb.getNextBitOffset(),
                          1);
      direct_spatial_mv_pred_flag = expGolomb.getValue();
}
```

6.2.3 Die Entropiecodierer

Die Entropiecodierer arbeiten auf Bytefeldern. Zum Parsen eines Parameters muss ihnen der aktuelle Offset innerhalb dieses Feldes, sowie der Offset innerhalb des aktuellen Bytes übergeben werden.

```
public void parseSE(byte[] data, int byteOffset, int bitOffset)
```

Nachdem der Parameter eingelesen wurde steht in den Membern `nextByteOffset` und `nextBitOffset` die nächste Position innerhalb des Feldes und des Bits. Der Exp Golomb Parser schreibt den Wert des Parameters in die Membervariable `value`. Der CAVLC Parser gibt die bestimmten Werte direkt zurück.

6.2.4 Das User Interface

6.2.4.1 Das User Interface der Android Version

Das User Interface wird unter Android aus drei Activities gebildet. Jede der Activities hat ein auf XML basierendes Interface. Die StartActivity besteht aus fünf Bedienelementen, während die anderen

beiden Activities Views zur Darstellung des Videos und der Messergebnisse besitzen. Drei der fünf Bedienelemente sind Buttons, welche mit einem OnClickListener belegt werden, so dass auf eine Berührung der Buttons reagiert werden kann.

```
public void onCreate(Bundle icicle)
{
	super.onCreate(icicle);
	setContentView(R.layout.start);
	url = (EditText)findViewById(R.id.url);
	play = (ImageButton)findViewById(R.id.play);
	setOnClickListener();
}

public void setOnClickListener()
{
	play.setOnClickListener(new View.OnClickListener()
	{
		@Override
		public void onClick(View view)
		{
			String socket = url.getText().toString()
			Bundle bundle = new Bundle();
			bundle.putString("socket", socket);
			Intent intent = new Intent(StartActivity.this,
							VideoActivity.class);
			intent.putExtras(bundle);
			startActivity(intent);
		}
	});
}
```

Zur Erläuterung sollen hier nur der Button `play` und die Editbox `url` dienen. Beide werden der Anwendung durch die Methode `findViewById` bekannt gemacht. Wird nun der Button berührt, so wird der Text der Editbox eingelesen. Der eingelesene Text wird in einem Bundle verpackt und ein expliziter Intent erzeugt. Diesem Intent wird das Bundle angehängt. Die Methode `startActivity` dient zur Ausführung des Intent. Die Activity VideoActivity führt bei der Erzeugung folgendes aus:

```
public void onCreate(Bundle icicle)
{
	...

	Bundle bundle = this.getIntent().getExtras();
	String[] socket = bundle.getString("socket").split(":");
	String host = socket[0];
	int port;
	if(socket.length > 1)
		port = Integer.parseInt(socket[1]);
	else
		port = 554;
	String file = bundle.getString("file");

	String[] codecs = {"H264", "MP4V-ES"};
	streamingHandler = new StreamingHandler(host, port, codecs);
	streamingHandler.start();

	mediaController = new MediaController(this);
	videoView = (VideoView) findViewById(R.id.screen);
	videoView.setVideoURI(Uri.parse("rtsp://localhost:8554/" +
							file));
	videoView.setMediaController(mediaController);
	startHandler();
}
```

Aus dem übergebenen Bundle werden die Strings `socket` und `file` entnommen. Aus dem `socket` werden der `host` und der `port` gefiltert. Diese werden zusammen mit der Liste von Codecs an einen StreamingHandler übergeben. Die VideoView wird erzeugt und mit einem MediaController versehen. Die VideoView fragt auf dem lokalen Socket ein File an, welches ebenfalls von der StartActivity eingelesen wurde. Als letztes wird noch ein Handler erzeugt und gestartet, um das User Interface zu updaten.

```
public void startHandler()
{
	updater = new Thread(new Timer());
	updater.start();
	handler = new Handler()
	{
		public void handleMessage(Message msg)
		{
			switch (msg.what)
			{
				case updateMessage:	setColourLight();
							break;
			}
			super.handleMessage(msg);
		}
	};
}
```

Bevor der Handler erzeugt wird, wird ein Thread mit einem Timer gestartet. Dieser Thread sendet in bestimmten Abständen die `updateMessage`. Erreicht diese Message den Handler wird die Funktion `setColourLight` ausgeführt.

```
public class Timer implements Runnable
{
	@Override
	public void run()
	{
		...
		while(!Thread.currentThread().isInterrupted())
		{
			Thread.sleep(updateIntervall);
			Message msg = new Message();
			msg.what = updateMessage;
			handler.sendMessage(msg);
		}
	}
}
```

6.2.4.2 Das User Interface der portierten Version

Das User Interface besteht für die portierte Version aus einem JFrame. Dem JFrame wird ein JPanel hinzugefügt. Das Panel wiederum beinhaltet mehrere JLabel zur Anzeige der ausgewerteten Daten. Das Updaten der JLabel übernimmt wieder ein Timer Thread. Allerdings ist hier kein Handler nötig.

7 Testing

In diesem Kapitel werden in ersten Teil kurz die zur Qualitätssicherung eingesetzten Methoden erläutert. Im zweiten Teil des Kapitels soll der Einfluss der entwickelten Software auf die Performance der Geräte dargestellt werden.

7.1 Software Testing

Für das Testen der Software wurden drei verschiedene Testphasen durchlaufen:

- Systemtest
- Modultest
- Funktionstest.

Die entwickelte Software stellt dabei das gesamte System dar. Die Software lässt sich in drei Module unterteilen:

- Das User Interface
- Der StreamingHandler und die ProtokollHandler
- Die Parser und Entropiecodierer

Das erste Modul, das User Interface, war sehr einfach zu testen. Die einzelnen Funktionen, beispielsweise das Betätigen der Buttons und der Aufruf einer zweiten Activity, sind leicht überschaubar und liefern unmittelbar ein sichtbares Ergebnis. Im Falle der portierten Version ist das Testen besonders leicht, da hier nur ein einzelnes Fester zum Anzeigen der Daten dient.
Der StreamingHandler und die ProtokollHandler waren ebenfalls relativ leicht zu testen. Zwar ist die Implementierung deutlich aufwändiger als beim User Interface, die Funktionalität ist jedoch im wesentlich auf das Manipulieren und Auswerten einiger Headerdaten sowie das Weiterleiten der Daten beschränkt. Zum Testen dieses Moduls wurde hauptsächlich der Netzwerksniffer Wireshark eingesetzt. Durch den Vergleich zwischen den mit Wireshark eingelesenen Parametern und den von der Software bestimmten Parametern ließen sich Fehler schnell lokalisieren und beheben. Die Integration in das User Interface ist ebenfalls ein einfacher Schritt, da der StreamingHandler als Schnittstelle dient.
Der mit deutlichem Abstand größte Testaufwand musste für die korrekte Funktionalität der Parser und Entropiecodierer aufgebracht werden. Die Schwierigkeit beim Testen der Funktionen der Parser und Entropiecodierer war, die Stelle zu lokalisieren an der sich der Fehler befand. Wurde ein Parameter des H.264 Standards nicht korrekt eingelesen, so machte sich dies nicht zwangsweise direkt bemerkbar, sondern oft erst einige Bits und Parameter später. Ebenso konnte in einem Video ein Fehlerzustand erreicht werden, in einem anderen Video hingegen erledigte die Software ihre Aufgabe fehlerfrei. Um die Fehler in den Funktionen dieses Moduls zu beheben, wurde eine Kombination aus H.264 Standard, Wireshark und H264Visa eingesetzt. H264Visa ist ein Tool zur Auswertung von H.264 Dateien, d.h. der Wert einiger H.264 Parameter kann mit H264Visa bestimmt werden. Mit Wireshark konnte der empfangene Bitstrom rekonstruiert werden und mit dem H.264 Standard der theoretische Aufbau vorhergesagt werden. Durch den Vergleich mit den von der Software eingelesenen Parametern konnte so der Fehler geortet werden. Funktionstest und Modultest mussten hier in einem Schritt abgearbeitet werden, da das Modul nicht eigenständig getestet werden konnte.
Das Auswerten der Slice Daten kann dazu führen, dass die Software in einen Fehlerzustand gerät. Ein Fehlerzustand kann auftreten, wenn die Maschine, auf der die Software läuft, nicht schnell genug die ankommenden Pakete verarbeiten kann. Einige Pakete werden dann verworfen. Sind diese Pakete Teil einer Fragmentation Unit, kann die NAL Unit nicht mehr korrekt zusammengefügt werden und somit auch nicht mehr korrekt geparst werden. Deshalb wurden als letzter Schritt bei der Erstellung der Software an einigen Stellen im Quellcode Mechanismen zum Vermeiden von Fehlerzuständen

eingefügt. Diese Stellen sind im Quellcode mit dem Kommentar „needed to avoid fault state“ versehen. Dennoch ist es nur unter sehr hohem Aufwand möglich alle Fehlerzustände abzufangen. Je anspruchsvoller das Video, desto wahrscheinlicher das Verwerfen von Paketen und somit auch das Auftreten von Fehlerzuständen.

7.2 Leistungsbetrachtung

Der Einfluss der entwickelten Software auf die Performance unter Android wird anhand eines einfachen TestPlayers gemessen. Dieser TestPlayer ist selbst geschrieben. Der Quellcode kann in Anhang E gefunden werden. Die Bedienung des TestPlayer gestaltet sich analog zur Bedienung der entwickelten Software. Der TestPlayer stellt eine Anfrage direkt an den Streaming Server ohne die Daten auszuwerten. Der Einfluss der Software auf die Performance eines PCs wird durch den Vergleich einer direkten Anfrage an den Streaming Server und einer Anfrage an die Software, die die Anfrage an den Streaming Server weiterleitet, ermittelt.
Als Videodaten dienen drei Videos, die in jeweils fünf verschiedenen Qualitätsstufen encodiert wurden. Die Videos wurden den Quellen [video1], [video2] und [video3] entnommen. Die Stufen sind durch die Endungen Min, Low, Mid, High und Max markiert. Tabelle 1 zeigt die genauen Encodierungsparameter der Videos. In den Spalten der Bitrate ist zuerst die von der Software in einem lokalen Netz gemessene Bitrate angegeben. In Klammern folgt die Bitrate, die im Session Description Protokoll angegeben wird. Zum Messen der Bitrate wurde ein lokales Netz verwendet, damit Paketverluste nicht das Ergebnis verfälschen.
Um die Leistungsfähigkeit der portierten Version zu testen, wurden die drei Videos zusätzlich in drei weiteren Stufen encodiert: Main_Low, Main_Mid und Main_High. Auflösung, Bitrate und Framerate entsprechen dabei den Endungen Low, Mid und High. Die Videos sind jedoch im Main Profile encodiert, so dass der Einfluss von B-Slices und B-Macroblöcken betrachtet werden kann.
Der Audio Track wurde bei allen Videos mit dem Codec AAC (Advanced Audio Coding) encodiert. AAC ist ebenso wie H.264 Teil des MPEG-4 Standards. Alle Audio Tracks sind mit einer Bitrate von 64 kb/s encodiert.
Die Leistungsbetrachtung unter Android erfolgt auf einem T-Mobile G1. Das G1 besitzt einen ARM Prozessor mit 528 MHz Taktfrequenz. Der Arbeitsspeicher ist 192 MB groß. Bei der Auswertung der Videos werden die Slice Daten nicht berücksichtigt. Hier haben erste Tests ergeben, dass sich selbst bei einer geringen Anzahl an Macroblöcken die Videoqualität drastisch verschlechtert.
Die Leistungsbetrachtung der portierten Version der Software erfolgt auf einem Notebook, welches über einen Intel Dual Core Prozessor mit 1.60 GHz Taktfrequenz und 1GB Arbeitsspeicher verfügt. Das Betriebssystem ist Windows Vista. Die portierte Version besitzt selbst keinen MediaPlayer, sondern nur ein Fenster zur Darstellung der ausgewerteten Parameter. Zum Testen wurde der VLC-Player (Video LAN Client) verwendet. Der VLC-Player ist Teil des Open Source Projekt Video LAN und ist für eine Vielzahl von Betriebssystemen verfügbar. Für die portierte Version erfolgt die Auswertung des Videostreams inklusive des Parsens der Slice Daten.

7.2.1 Leistungsbetrachtung unter Android

Das Auswerten der Videos unter Android hat praktisch keinen Einfluss auf die Qualität der Übertragung. Die Software selbst verursacht auch bei den Max-Videos keine Blockbildung und kein Ruckeln. Die Darstellung entspricht der maximal möglichen Qualität. Bei hoher Videoqualität treten innerhalb der ersten 2-3 Sekunden der Übertragung sehr kleine Paketverluste auf. Dies ist jedoch ein Problem, welches Android-spezifisch ist, da im TestPlayer die gleichen Artefakte auftreten.
Der Jitter hat einen erheblichen Einfluss auf die Leistung der Software. Dabei spielt es keine Rolle, ob das Video in einer hohen oder niedrigen Bitrate encodiert wurde. Bei Jitterwerten kleiner als 50 ms kann das Video problemlos dargestellt werden. Jitterwerte zwischen 50 ms und 70 ms verursachen Ruckeln und Artefakte. Bei noch größeren Werten kann es zum Abbrechen der Darstellung kommen. Dabei wird kein TEARDOWN-Request gesendet, d.h. der Streaming Server sendet weiterhin Daten.

Name	Bitrate Video Track	Bitrate Gesamt	Framerate	Auflösung	Macroblöcke/ Frame	Profil
Sommermaerchen_ Min.mp4	107 (108) kb/s	166 (167) kb/s	15	144 * 96	54	Baseline
Sommermaerchen_ Low.mp4	156 (157) kb/s	215 (216) kb/s	20	192 * 128	96	Baseline
Sommermaerchen_ Mid.mp4	205 (207) kb/s	264 (266) kb/s	25	240 * 160	150	Baseline
Sommermaerchen_ High.mp4	257 (259) kb/s	316 (318) kb/s	25	384 * 256	384	Baseline
Sommermaerchen_ Max.mp4	304 (307) kb/s	363 (366) kb/s	25	480 * 320	600	Baseline
IceAge3_Min.mp4	105 (107) kb/s	166 (169) kb/s	15	144 * 96	54	Baseline
IceAge3_Low.mp4	154 (156) kb/s	215 (218) kb/s	20	192 * 128	96	Baseline
IceAge3_Mid.mp4	200 (202) kb/s	261 (264) kb/s	25	240 * 160	150	Baseline
IceAge3_High.mp4	251 (254) kb/s	312 (316) kb/s	25	384 * 256	384	Baseline
IceAge3_Max.mp4	302 (305) kb/s	363 (367) kb/s	25	480 * 320	600	Baseline
Simpsons_Min.mp4	121 (122) kb/s	185 (186) kb/s	15	144*96	54	Baseline
Simpsons_Low.mp4	159 (160) kb/s	223 (224) kb/s	20	192*128	96	Baseline
Simpsons_Mid.mp4	202 (203) kb/s	266 (267) kb/s	23,98	240*160	150	Baseline
Simpsons_High.mp4	236 (238) kb/s	300 (302) kb/s	23,98	384*256	384	Baseline
Simpsons_Max.mp4	277 (279) kb/s	341 (343) kb/s	23,98	480*320	600	Baseline
Sommermaerchen_ Main_Low.mp4	157 (157) kb/s	215 (216) kb/s	20	192*128	96	Main
Sommermaerchen_ Main_Mid.mp4	203 (204) kb/s	262 (263) kb/s	25	240*160	150	Main
Sommermaerchen_ Main_High.mp4	257 (259) kb/s	316 (318) kb/s	25	384 * 256	384	Main
IceAge3_ Main_Low.mp4	150 (152) kb/s	211 (214) kb/s	20	192*128	96	Main
IceAge3_ Main_Mid.mp4	190 (196) kb/s	251 (258) kb/s	25	240*160	150	Main
IceAge3_ Main_High.mp4	230 (241) kb/s	292 (303) kb/s	25	384 * 256	384	Main
Simpsons_ Main_Low.mp4	168 (172) kb/s	232 (236) kb/s	20	192*128	96	Main
Simpsons_ Main_Mid.mp4	203 (211) kb/s	268 (275) kb/s	23,98	240*160	150	Main
Simpsons_ Main_High.mp4	238 (251) kb/s	303 (315) kb/s	23,98	384*256	384	Main

Tabelle 1: Encoding Settings

7.2.2 Leistungsbetrachtung der portierten Version

Bei der portierten Version ist der Einfluss der Software auf die Übertragung von anspruchsvollen Videos deutlich sichtbar. Hier werden zum Teil Pakete verworfen. Dies betrifft vor allem Fragmentation Units, da diese einen hohen Aufwand zum Verarbeiten benötigen. Aber gerade die fragmentierten NAL Units (also die großen NAL Units) tragen viele Informationen. Ein Verlust wiegt deshalb doppelt schwer. Desweiteren ist trotz ähnlicher Bitrate ein Unterschied zwischen den einzelnen Videos erkennbar. Schnelle Szenenwechsel führen oft zum Ruckeln oder zur Blockbildung.

Viel Bewegung mit wenigen Szenenwechseln kann gut verarbeitet werden.
Um zu unterscheiden, ob Paketverluste durch die Übertragung oder die Software zu Stande kommen, wurde die Software in einem lokalen Netz getestet. Hier ist der Jitter konstant niedrig und es treten praktisch keine Paketverluste auf. Entdeckt die Software nun eine Lücke in den RTP-Sequenznummern, muss der Paketverlust durch Verwerfen der Pakete entstanden sein.
B-Slices haben einen kleinen Effekt auf die Leistungsfähigkeit der portierten Version. Es werden nicht so viele Pakete verworfen, wenn B-Slices verwendet werden. Dies macht sich in einer besseren Qualität bei der Darstellung bemerkbar.

Name	Subjektive Qualität	Ruckeln	Blockbildung und Artefakte	Verworfene Pakete
Sommermaerchen_ Min.mp4	Mittelmäßig	Nie	Nie	0 %
Sommermaerchen_ Low.mp4	Gut	Nie	Nie	0 %
Sommermaerchen_ Mid.mp4	Sehr gut	Nie	Sehr selten	0 %
Sommermaerchen_ High.mp4	Gut	Selten	Selten	1 %
Sommermaerchen_ Max.mp4	Mittelmäßig	Selten	Häufig	2%
IceAge3_Min.mp4	Mittelmäßig	Nie	Nie	0 %
IceAge3_Low.mp4	Gut	Nie	Sehr selten	0 %
IceAge3_Mid.mp4	Gut	Selten	Selten	1 %
IceAge3_High.mp4	Schlecht	Häufig	Häufig	2 %
IceAge3_Max.mp4	Schlecht	Häufig	Sehr Häufig	3 %
Simpsons_Min.mp4	Mittelmäßig	Nie	Nie	0 %
Simpsons_Low.mp4	Gut	Sehr selten	Sehr selten	1 %
Simpsons_Mid.mp4	Mittelmäßig	Selten	Selten	2 %
Simpsons_High.mp4	Schlecht	Häufig	Häufig	4 %
Simpsons_Max.mp4	Sehr schlecht	Sehr häufig	Sehr häufig	6 %
Sommermaerchen_ Main_Low.mp4	Gut	Nie	Nie	0 %
Sommermaerchen_ Main_Mid.mp4	Sehr gut	Nie	Sehr selten	0 %
Sommermaerchen_ Main_High.mp4	Sehr gut	Sehr selten	Sehr selten	0 %
IceAge3_ Main_Low.mp4	Gut	Nie	Sehr selten	0 %
IceAge3_ Main_Mid.mp4	Sehr gut	Sehr selten	Sehr selten	1 %
IceAge3_ Main_High.mp4	Mittelmäßig	Selten	Häufig	2 %
Simpsons_ Main_Low.mp4	Gut	Sehr selten	Sehr selten	1 %
Simpsons_ Main_Mid.mp4	Gut	Selten	Selten	1 %
Simpsons_ Main_High.mp4	Mittelmäßig	Häufig	Selten	2 %

Tabelle 2: Leistungsbetrachtung der portierten Version

8 Auswertung der Messergebnisse

Generell lässt sich sagen, dass die Parameter der Netzwerkebene und die Parameter von H.264 einen größeren Einfluss auf die Qualität des Videos haben als die Parameter des Transports. Die Parameter der Netzwerkebene sind sehr dynamisch. Mit den Veränderungen dieser Parameter kann die Qualität der Videos drastisch schwanken. Die statischen Parameter von H.264, also die Parameter Sets, beschreiben die maximal mögliche Videoqualität. Der Einfluss dieser Parameter auf die Qualität ist ebenfalls sehr groß. Die dynamischen Parameter von H.264 haben in Abhängigkeit von den Werten der Parameter der Netzwerkebene einen verschwindend geringen bis großen Einfluss. Gleiches gilt für die Parameter des Transports von H.264.

8.1 Einfluss der Netzwerkparameter

Parameter	Einfluss	Gemessene Werte (Android)	Typische Werte (Android)	Gemessene Werte (Portierung)	Typische Werte (Portierung)
Jitter	Sehr Hoch	10 ms - 295 ms	20 ms - 60 ms	10 ms - 154 ms	20 ms - 45 ms
Single Losses / 1000 Pakete	Gering	0 - 22	0 - 4	0 - 7	0 - 2
Burst Losses / 1000 Pakete	Hoch	0 - 26	0 - 6	0 - 11	0 - 2
Burst Loss Dauer	Sehr Hoch	2 - 327	3 - 20	2 - 35	3 - 10
Out of Order / 1000 Pakete	Gering	0 - 12	0 - 3	0 - 4	0 - 2
Server IP	Keiner	2 Server	-	2 Server	-

Tabelle 3: Einfluss der Netzwerkparameter

Man kann erkennen, dass Jitter und Verlustrate bei der Übertragung zum G1 in der Regel größer sind als bei der Übertragung zu einem PC. Die gemessenen Werte sind im Falle des G1 zudem von der Tageszeit abhängig. In der Mittagszeit und am frühen Abend sind Jitter und Verlustrate oft höher als am Morgen und in der Nacht. Einzelne Paketverluste und Pakete in vertauschter Reihenfolge haben nur in Ausnahmefällen Einfluss auf die Qualität. Resultat sind meistens Artefakte und Blockbildung. Burst Losses und Jitter wirken sich deutlich stärker aus. Hier kommt es zusätzlich zum Ruckeln des Videos.

8.2 Einfluss der Transportparameter

Der wichtigste Parameter des Transports ist das Verhältnis von Single NAL Unit Packets und Fragmentation Units. Bei einer hohen Anzahl an Paketverlusten oder einem sehr hohen Jitter wirkt sich ein großer Anteil an Fragmentation Units negativ aus. Zusammen mit schlechten Netzwerkparametern führt ein hoher Anteil an Fragmentation Units zu starkem Ruckeln des Videos. Der Einfluss von Aggregation Packets konnte nicht gemessen werden, da die zum Encodieren verwendete Software keine NAL Units mit dem gleichen Decoding Timestamp erzeugt hat.

Parameter	Einfluss	Einfluss abhängig von Parameter	Gemessene Werte	Typische Werte
Anzahl Pakete / Min.	Gering	-	981 - 2329	1500 - 2000
Bits / Paket	Gering	-	6172 - 7895	6300 - 7700
Single Packets / Fragmentation Units	Gering - Sehr Hoch	Jitter, Losses	5 - 1	3 - 2
Fragmente / NAL Unit	Gering - Hoch	Jitter, Losses	5 - 2	3 - 2

Tabelle 4: Einfluss der Transportparameter

8.3 Einfluss der H.264-Parameter

Parameter	Einfluss	Einfluss abhängig von Parameter	Gemessene (Encodierte) Werte
Bitrate	Sehr hoch	-	105 kb/s - 304 kb/s
Auflösung	Sehr hoch	-	144*96 - 480*320
Framerate	Hoch	-	15 - 25
Profil	Gering	-	Baseline, Main (nur Portierung)
Level	Keiner	-	13 - 51
Größe Referenzlisten	Gering - Mittel	Jitter, Losses	1 - 5
Maximale GOP	Gering - Mittel	Jitter, Losses	128 - 512
IDR Abstand	Gering - Hoch	Jitter, Losses	25 - 161
Anteil Non-Reference NAL Units	Gering - Hoch	Jitter, Losses	0 - 3
P-/I-Slice Verhältnis	Gering - Mittel	Jitter, Losses	12 - 88
B-/I-Slice Verhältnis	Gering - Mittel	Jitter, Losses	31 - 88
P-/I-Macroblock Verhältnis	Gering - Hoch	Jitter, Losses	2 - 17
B-/I-Macroblock Verhältnis	Gering - Hoch	Jitter, Losses	3 - 11
Anteil Skip-Macroblöcke	Gering - Hoch	Videoinhalt	32 % - 67 %

Tabelle 5: Einfluss der H.264-Parameter

Bitrate, Auflösung und Framerate sind von den H.264-Parametern die Wichtigsten. Die Bitrate lässt sich für die zum Testen verwendeten Videos mit hoher Auflösung noch weiter nach unten drücken. Jedoch führt dies zu einem größeren Anteil an Skip-Macroblöcken. Abhängig vom Videoinhalt macht sich dies bei der Darstellung bemerkbar. Bei den Sommermaerchen-Videos sind relativ viele weiche Übergänge, die bei einem hohen Anteil an Skip-Macroblöcken verschwommen wirken. Bei den Simpsons-Videos existieren viele große Flächen mit gleichem Helligkeits- und Farbanteil. Hier macht sich eine größere Anzahl von Skip-Macroblöcken kaum bemerkbar. Das Profil hat nur indirekt Einfluss auf die Qualität, da hier die verwendeten Techniken definiert werden. Der angegebene Level hat keinen Einfluss, da die Rahmenbedingungen durch die anderen Parameter festgelegt werden und

der Level nur Obergrenzen für die Rahmenbedingungen definiert. Der Einfluss der übrigen Parameter ist abhängig von den Netzwerkparametern. Bei geringen Paketverlusten und kleinem Jitter sind kaum Auswirkungen zu erkennen. Bei schlechten Netzwerkparametern kann die Qualität jedoch zum Teil stark beeinträchtigt werden. Der Einfluss der maximalen Group of Pictures Größe ist klein, da praktisch nie die maximale GOP Größe ausgenutzt wird. Eine Ausnahme hiervon bilden sehr kleine Werte, da hierdurch die Häufigkeit von IDRs gesteigert wird. Durch einen kleineren Abstand von IDRs erscheinen weniger Artefakte und Ruckeln. Im Main Profile können B-Slices und somit auch Non Reference NAL Units genutzt werden. Ein Verlust dieser NAL Units wirkt sich deutlich weniger stark aus als ein Verlust eines Reference NAL Units. Das Verhältnis von P- bzw. B- zu I-Slices ist nur bedingt aussagekräftig. Hieraus kann zwar abgelesen werden, wie lange der Decoder maximal braucht, um sich nach einem Paketverlust wieder zu synchronisieren. Jedoch können auch P- und B-Slices beinahe vollständig aus I-Macroblöcken bestehen. Ein Verlust kann deshalb fast genauso schwer wirken wie der eines I-Slices. Das Verhältnis von P- bzw. B- zu I-Macroblöcken ist deshalb aussagekräftiger.
Die zum Encodieren der Videos genutzte Software hat leider keine Unterstützung für mehrere Slices und Slice Groups geboten. Ebenso konnte der Einfluss eines anderen Chroma Formats als 4:2:0 nicht gemessen werden, da diese ausschließlich in den höchsten Profilen genutzt werden können. Der CABAC Entropiecodierer wurde nicht implementiert, so dass auch hier der Einfluss nicht ermittelt werden konnte.

9 Die entwickelte Software

9.1 Die Android Version

Das Benutzer Interface der StartActivity besteht aus 5 Widgets mit denen der Benutzer interagieren kann. In der ersten Zeile muss der Streaming Server angegeben werden, in der zweiten Zeile die angefragte Datei.

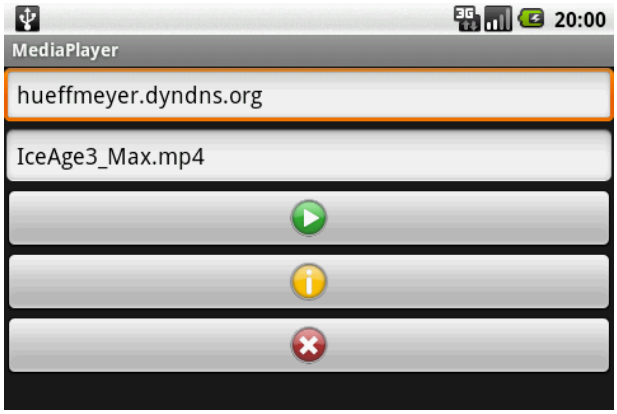

Abb. 77: Das User Interface

Mit dem grünen Play Button wird das Video gestartet und die Messergebnisse werden über eine „Ampel“ dargestellt. Die Statusbar und die Titelleiste wurden aus der Darstellung entfernt.

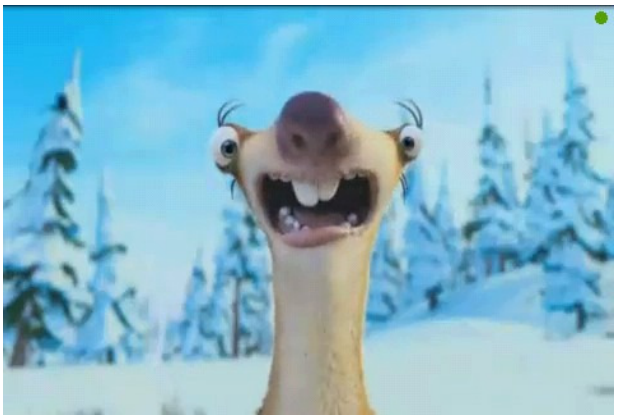

Abb. 78: Darstellung der Messergebnisse über die "Ampel"

Hinter dem gelben Info Button verbirgt sich die Kernfunktion der erstellten Software. Hier wird ein Teil des Bildschirms genutzt, um das Video anzuzeigen. Der andere Teil des Bildschirms wird genutzt, um die ausgelesenen Parameter anzuzeigen. Auch in dieser Variante wurden Statusbar und Titelleiste entfernt.

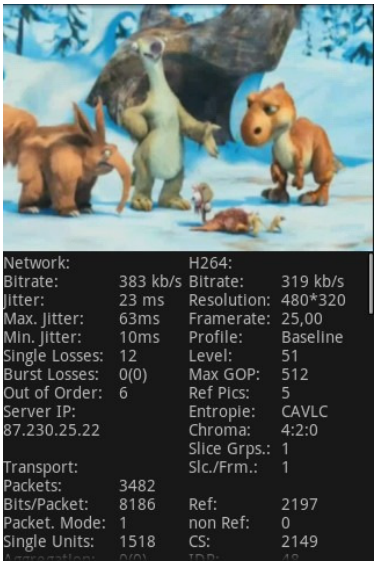

Abbildung 79a: Detaillierte Darstellung

Abbildung 79b: Detaillierte Darstellung

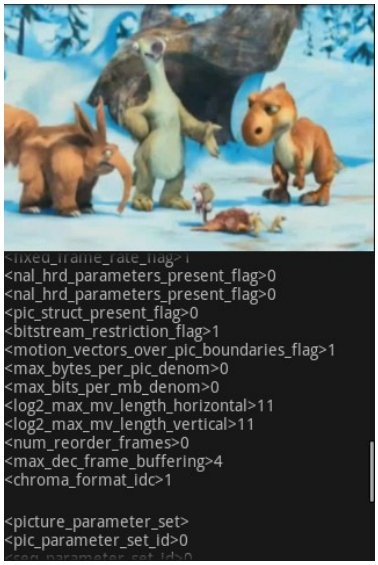

Abbildung 79c: Detaillierte Darstellung

Abbildung 79a zeigt auf der linken Seite die ausgewerteten Netzwerk- und Transport-Parameter. Auf der rechten Seite sind einige H.264-Parameter zu sehen. Weitere H.264-Parameter sind auf der rechten Seite von Abbildung 79b zu sehen. Die Anordnung der H.264-Parameter entspricht dabei den Gruppierungen aus Kapitel 4.3 mit Ausnahme der Macroblöcke. Abbildung 79c zeigt die Darstellung der kompletten Parameter Sets. Um andere Codecs auszuwerten, müssen diese Codecs in einer Liste an den StreamingHandler übergeben werden und ggf. ein oder zwei Parser implementiert werden. Abbildung 80 zeigt die Auswertung für MPEG-4 Visual. Dieser Codec wurde in der Liste eingetragen, jedoch ist kein Parser implementiert, so dass nur die Netzwerkparameter und einige Transportparameter ausgewertet werden können. Bei der Rückkehr zum Menu werden die ausgwerteten Daten in eine XML-Datei auf der SD-Karte geschrieben. Der Aufbau dieser XML-Datei kann in Anhang B gefunden werden.

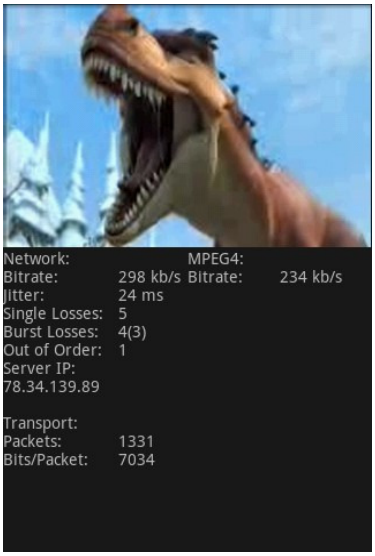

Abb. 80: MPEG-4 Auswertung

Der rote Exit Button aus Abbildung 77 dient zum Beenden der Anwendung.

9.2 Die portierte Version

Aus dem Quellcode der portierten Version lässt sich ein lauffähiges JAR-File erzeugen. Zum Entwickeln der Anwendung wurde das JDK 1.6.0 verwendet, weshalb auch die JRE 1.6.0 für den Betrieb nötig ist. Als Übergabeparameter bekommt das JAR-File in Argument 1 den Hostnamen und ggf. in Argument 2 eine Portnummer. Der Aufruf könnte z.B. folgendermaßen aussehen:

VideoParser.jar hueffmeyer.dyndns.org 554

Nachdem die Verbindung zum Server steht, kann ein MediaPlayer gestartet werden und die gewünschte Datei auf Port 8554 angefragt werden. Die URL kann beispielsweise so aussehen:

rtsp://localhost:8554/Simpsons_Main_High.mp4

Die Darstellung der eingelesenen Parameter zeigt Abbildung 81. Neben den H.264-Parametern befinden sich hier zusätzlich die in den Slices enthaltenen Macroblock- und Submacroblocktypen.

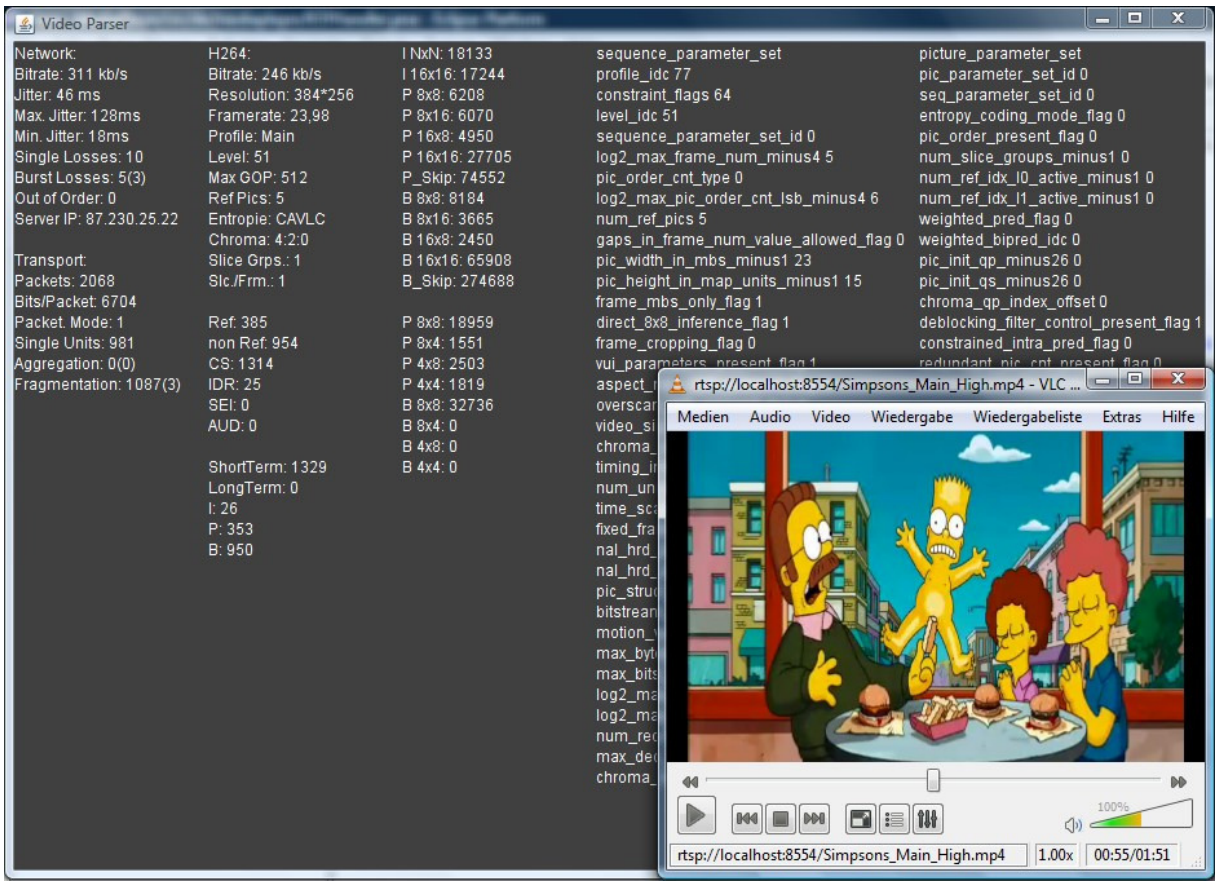

Abbildung 81: Darstellung der portierten Version

10 Schlussbetrachtung und Ausblick

H.264 ist ein sehr interessanter Standard für die Übertragung von Videodaten. Durch die hohe Kompressionsrate sind beachtliche Ergebnisse beim Streamen möglich. Der Einsatz des Baseline Profils ergibt qualitativ deutlich bessere Ergebnisse als beispielsweise der Einsatz von MPEG-4 Visual im Advanced Simple Profile. Die Leistungsfähigkeit der Endgeräte ist bereits jetzt recht groß und nimmt weiter zu, so dass in Zukunft weitere Features von H.264 auch auf mobilen Endgeräten genutzt werden können.
Android stellt eine attraktive Plattform zur Entwicklung von Anwendungen für mobile Geräte dar, die jedoch mit Startschwierigkeiten zu kämpfen hat. Die ersten Schritte in der Android Umgebung können mit Hilfe von Tutorials schnell getätigt werden und liefern eindrucksvolle Ergebnisse. Insbesondere die Kreation des User Interface innerhalb einer XML-Datei ist ein sehr nützliches Konzept. Die nächsten Schritte sind jedoch schwierig, da bisher recht wenig Dokumentation existiert. Dies gilt vor allem für die Android API. Hier sind oft nur Prototypen von Funktionen oder ein kurz gehaltener Satz zu finden. Das Problem der mangelhaften Dokumentation ist vermutlich jedoch nur temporär, da Android eine relativ neue Technologie ist
Die entwickelte Software konnte erfolgreich implementiert und getestet werden. Die Kernfunktionen der Software, also das Abgreifen und Weiterleiten der Daten, sind plattformunabhängig implementiert und bieten somit die Möglichkeit einer einfachen Portierung in eine andere Umgebung. Das User Interface muss plattformabhängig programmiert werden. Für Android wurde eine leicht zu steuernde Oberfläche eingerichtet. Um den Einsatz der Software auf einem PC zu ermöglichen, wurde eine sehr einfache Oberfläche gestaltet. Diese Oberfläche kann in Zukunft mit weiteren Funktionen versehen werden.
Durch Implementieren der DatagramParser-Schnittstelle kann die Software leicht um weitere Parser für andere Codecs erweitert werden. Als weitere Codecs sind beispielsweise MPEG-4 Visual oder AAC denkbar. Als weiteres Feature für die Dekodierung von H.264 ist die Implementierung eines CABAC Entropiedekodierers denkbar. Jedoch sollte die Software in diesem Fall auf einer recht leistungsstarken Maschine eingesetzt werden.

11 Literaturverzeichnis

[H264] International Telecommunication Union, Telecommunication Standardization Sector, ITU-T Recommendation H.264, 2005

[RFC2326] Internet Engineering Task Force, Network Working Group, Real Time Streaming Protocol (RTSP), 1998

[RFC3548] Internet Engineering Task Force, Network Working Group, RTP: A Transport Protocol for Real-Time Applications, 2003

[RFC3550] Internet Engineering Task Force, Network Working Group, The Base16, Base32, and Base64 Data Encodings, 2003

[RFC3984] Internet Engineering Task Force, Network Working Group, RTP Payload Format for H.264 Video, 2005

[RFC4566] Internet Engineering Task Force, Network Working Group, SDP: Session Description Protocol, 2006

[MPEG4] http://www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.htm

[www01] http://www.sony.co.uk/biz/view/ShowContent.action?site=biz_en_GB&contentId=1222694811694&sectiontype=NVM+Whitepapers

[www02] http://keyj.s2000.ws/files/projects/videocomp.pdf

[www03] http://www.enzyklopaedie-der-wirtschaftsinformatik.de/wi-enzyklopaedie/lexikon/technologien-methoden/Informatik--Grundlagen/Kodierung-von-Daten/Mediale-Daten--Kodierung/Videoformate

[www04] http://de.wikipedia.org/wiki/YCbCr

[www05] http://www.dma.ufg.ac.at/app/link/Grundlagen:Video.Postproduction/module/3927

[www06] http://de.wikipedia.org/wiki/Diskrete_Kosinustransformation

[www07] http://videocodecs.blogspot.com/

[www08] http://www.tkt.cs.tut.fi/kurssit/vc2006/sgn-3156-03.pdf

[www09] http://en.wikipedia.org/wiki/Flexible_Macroblock_Ordering

[www10] http://broadcastengineering.com/infrastructure/mpeg-havc/

[www11] http://www.videobits.org/prediction.html

[www12] http://pds12.egloos.com/pds/200901/08/58/a0101158_4965c1741c6f5.jpg

[www13] http://de.wikipedia.org/wiki/Containerformat

[www14] http://www.pacontrol.com/Ethernet.html

[www15] http://www.w3.org/2008/WebVideo/Fragments/wiki/UA_Server_RTSP_Communication

[www16] http://de.wikipedia.org/wiki/Base64

[www17] http://developer.android.com/guide/basics/what-is-android.html

[www18] http://stuffthathappens.com/blog/2008/11/05/android-activity-lifecycle-updated/

[www19] http://developer.android.com/guide/topics/fundamentals.html

[www20] http://developer.android.com/guide/topics/ui/index.html

[www21] http://www.torsten-horn.de/techdocs/sw-dev-process.htm

[video1] http://www.deutschlandeinsommermaerchen.kinowelt.de/

[video2] http://www.iceage3-derfilm.de/

[video3] http://www.simpsonsmovie.com/

12 Abbildungsverzeichnis

Anhang A - Verwendete Software

Die zur Entwicklung und zum Testen eingesetzten Programme sollen an dieser Stelle kurz vorgestellt werden.

A.1 Eclipse

Eclipse ist eine offene Entwicklungsumgebung. Primär dient sie der Entwicklung von Java Anwendungen, aber auch andere Aufgaben können mit Eclipse erledigt werden. Beispielsweise stehen auch Varianten für die Entwicklung von C/C++ und PHP Anwendungen zur Verfügung. Die Oberfläche kann durch verschiedene Perspektiven und Views variabel gestaltet werden. Eine Vielzahl von Plugins erweitert die Funktionalität der Entwicklungsumgebung.

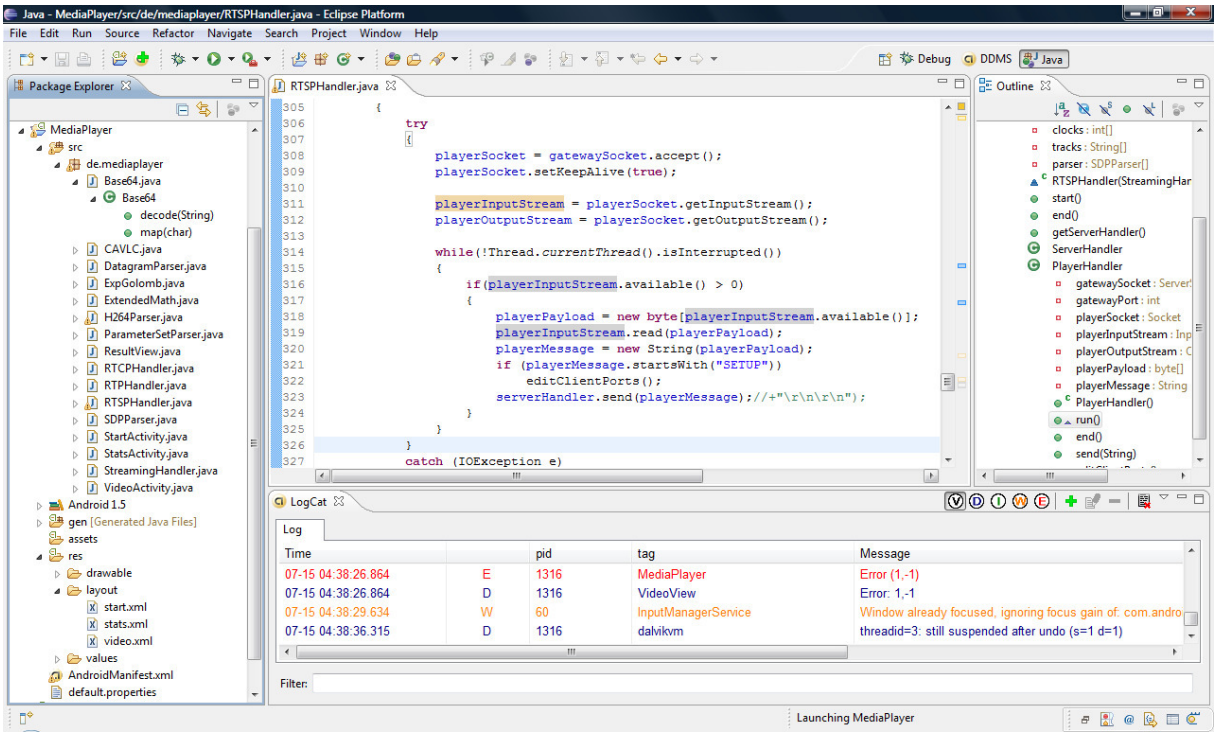

Abb. A.1: Eclipse

A.2 Android SDK

Das Android Software Development Kit (SDK) stellt die nötigen Vorraussetzungen, um eine Android Anwendung zu programmieren, zur Verfügung. Neben einigen nützlichen Tools bietet das SDK einen Emulator auf dem die Anwendungen getestet werden können. Jedoch sind einige Funktionen eines echten Gerätes nicht im Emulator nutzbar, beispielsweise wird RTSP Streaming nicht unterstützt. Für das SDK existiert ein Eclipse-Plugin. Anwendungen können damit aus Eclipse heraus auf das Gerät geladen, dort ausgeführt und unter Eclipse gedebuggt werden.

Abb. A.2: Android Emulator

A.3 Darwin Streaming Server

Der Darwin Streaming Server ist ein frei verfügbarer Streaming Server der Firma Apple. Er stellt eine einfache Version des Quicktime Streaming Server dar. Der Darwin Server unterstützt den Transport von Videos über RTSP. Als Eingangsdaten können Videos in den Containern MP4, 3GPP und Quicktime dienen. Die unterstützten Videocodecs sind H.264 und MPEG-4 Visual. Bei den Audiocodecs können MP3 und AAC verwendet werden.

A.4 Wireshark

Wireshark ist ein Programm, das Netzwerkverkehr analysieren kann. Es beherrscht viele Protokolle, so dass eine Auswertung der Headerdaten leicht möglich ist. Das User Interface besteht aus drei wichtigen Komponenten: Dem Protokollstrom, einem Fenster für die Darstellung eines ausgewählten Pakets sowie einem Fenster für die hexadezimale Dartellung der empfangenen Daten.

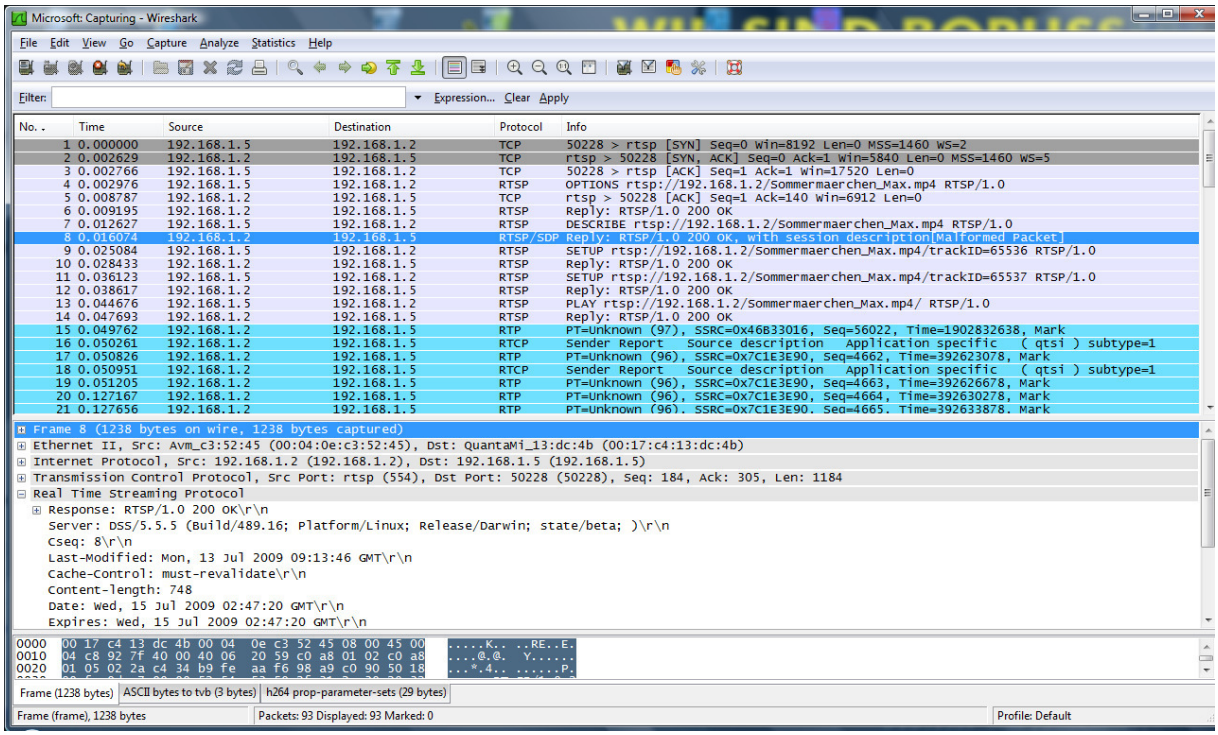

Abb. A.3: Wireshark

A.5 FFMPEG

FFMPEG ist eine freie Software zum Encodieren, Transcodieren und Streamen von Videos. Im Rahmen dieser Arbeit wurde FFMPEG zum Transcodieren benutzt. Für viele Plattformen stehen eine Reihe von Benutzeroberflächen zur Verfügung. In der Arbeit wurde unter Linux auf der Kommandozeile gearbeitet, da die Benutzeroberflächen nicht so flexibel wie die Kommandozeile sind. Das folgende Kommando zum Transcodieren eines Videos zeigt die wichtigsten Optionen, um das Video in das H.264 Baseline Profil zu konvertieren:

*marc@192.168.1.2:~$ ffmpeg -i Videos/Sommermärchen.mov -vcodec h264 -b 300k -s 240*160 -r 25 -refs 5 -bf 0 -coder 0 -acodec aac -ab 64k Sommermaerchen_Mid.mp4*

vcodec und *acodec* definieren die neuen Codecs. *b* und *ab* geben die Bitrate an die für den jeweiligen Track genutzt werden soll. *s* definiert die Auflösung, *r* die Framerate. *refs* gibt vor wie viele Referenzbilder verwendet werden sollen. Mit der Option *bf* wird die maximale Anzahl an aufeinander folgenden B-Frames vorgegeben. *coder* bestimmt den eingesetzten Entropiecodierer, der Wert 0 steht für CAVLC, der Wert 1 für CABAC.

A.6 MP4Box

MP4Box ist ebenfalls eine freie Software, die zum Multiplexen von verschiedenen Tracks in den MP4-Container dient. Genau wie FFMPEG arbeitet MP4Box auf Kommandozeilenebene. Aber auch hier existieren eine Reihe von Benutzeroberflächen. In dieser Arbeit wurde MP4Box insbesondere zum Hinzufügen von Hint-Tracks in den MP4-Container genutzt:

marc@192.168.1.2:~$ MP4Box -hint Sommermaerchen_Mid.mp4

A.7 H264Visa

H264Visa ist ein Programm um H.264 Spuren zu analysieren. Mit Hilfe dieser Software ist es möglich sämtliche Headerdaten innerhalb der Spur auszulesen. Daneben können die Macroblöcke detailliert betrachtet werden. H264Visa wurde eingesetzt um die Funktionalität der Software zu testen. Die Oberfläche besteht aus einer Anzeige des Videos sowie mehreren Fenstern. Über die Anzeige des Videos kann ein Raster gelegt werden, welches die im Bild enthaltenen Macroblöcke darstellt. Beim Testen der entwickelten Software wurden die Fenster Header Info und MB Info oft verwendet. Im Header Info Fenster können die NAL Unit Header, die Parameter Sets und die Slice Header der Video Spur angezeigt werden. Das MB Info Fenster stellt einen ausgewählten Makroblock detailliert dar. Mit dieser Funktion konnte das Parsen der Macroblöcke und Submacroblöcke auf seine Korrektheit geprüft werden.

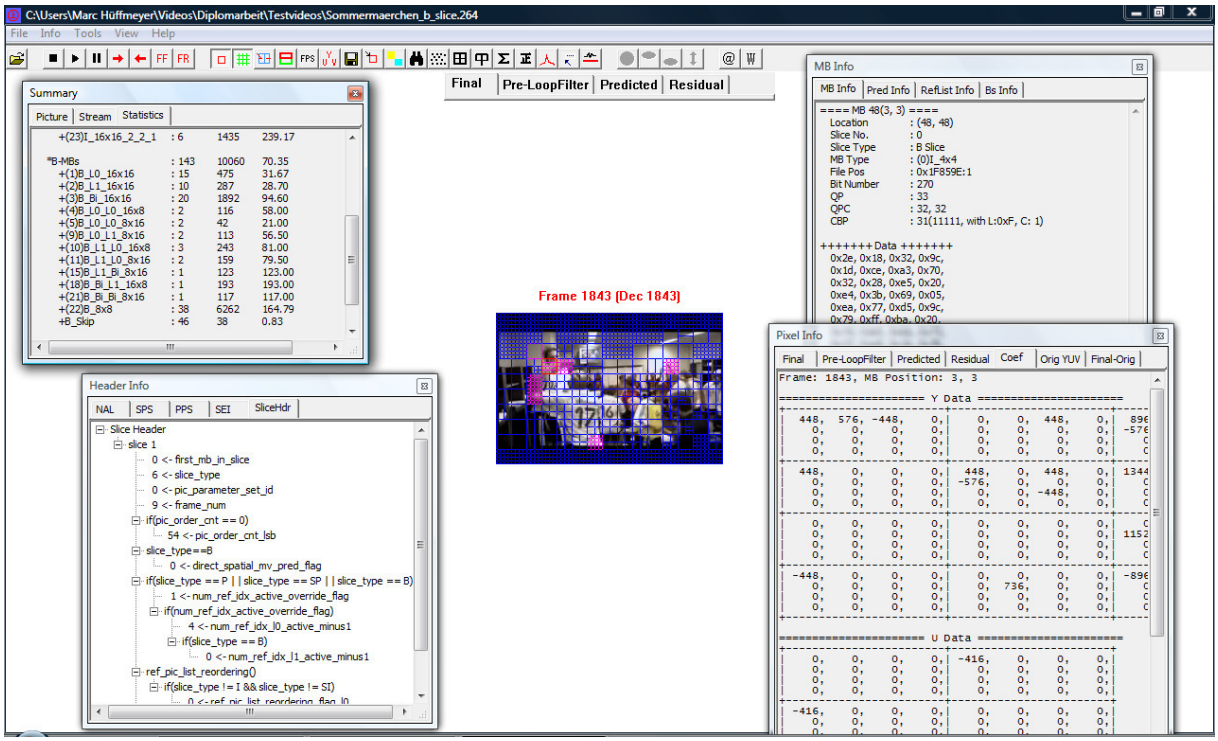

Abb. A.4: H264Visa

Anhang B - Ausgabeformat der Messdaten

Auf der SD-Karte des Android-Gerätes wird bei der Rückkehr aus dem Auswertungsbildschirm zum Menu der Software eine XML-Datei mit den ausgewerteten Daten angelegt. Für jede Messung wird dabei eine neue Datei angelegt. Die Dateien tragen die Namen video0.xml, video1.xml, video2.xml usw. Der Aufbau der Dateien ist unten abgebildet.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<video>
	<execution>
		<time>26 Jul 2009 08:33:37 GMT</time>
		<url>rtsp://hueffmeyer.dyndns.org:554/IceAge3_Mid.mp4</url>
	</execution>
	<network>
		<bit-rate>277 kb/s</bit-rate>
		<jitter>23 ms</jitter>
		<max_jitter>38 ms</max_jitter>
		<min_jitter>8 ms</min_jitter>
		<single-losses>0</single-losses>
		<burst-losses>4(3)</burst-losses>
		<out-of-order>0</out-of-order>
		<server-ip>78.34.183.103</server-ip>
	</network>
	<transport>
		<packets>2654</packets>
		<bits-per-packet>7042</bits-per-packet>
		<packetization-mode>1</packetization-mode>
		<single-packets>1863</single-packets>
		<aggregation-packets>0(0)</aggregation-packets>
		<fragmentation-units>791(2,9)</fragmentation-units>
	</transport>
	<h264>
		<parameter-sets>
			<video-bit-rate>214 kb/s</video-bit-rate>
			<resolution>240*160</resolution>
			<framerate>25,00</framerate>
			<profile>Baseline</profile>
			<level>51</level>
			<max-gop>512</max-gop>
			<max-ref-pics>3</max-ref-pics>
			<entropy-coder>CAVLC</entropy-coder>
			<chroma-format>4:2:0</chroma-format>
			<num-slice-groups>1</num-slice-groups>
			<slices-per-frame>1</slices-per-frame>
		</parameter-sets>
		<nal-units>
			<ref-nals>2132</ref-nals>
			<non-ref-nals>0</non-ref-nals>
			<coded-slices>2086</coded-slices>
			<instantaneous-decoding-refresh>46
							</instantaneous-decoding-refresh>
			<supplement-enhancement-message>0
							</supplement-enhancement-message>
			<access-unit-delimiter>0</access-unit-delimiter>
		</nal-units>
		<slices>
			<short-term-pics>2132</short-term-pics>
			<long-term-pics>0</long-term-pics>
			<i-slices>75</i-slices>
			<p-slices>2057</p-slices>
		</slices>
	</h264>
</video>
```

Anhang C - CAVLC Tabellen des H.264 Standards

TrailingOnes (coeff_token)	TotalCoeff (coeff_token)	0 <= nC < 2	2 <= nC < 4	4 <= nC < 8	8 <= nC	nC = = -1	nC = = -2
0	0	1	11	1111	0000 11	01	1
0	1	0001 01	0010 11	0011 11	0000 00	0001 11	0001 111
1	1	01	10	1110	0000 01	1	01
0	2	0000 0111	0001 11	0010 11	0001 00	0001 00	0001 110
1	2	0001 00	0011 1	0111 1	0001 01	0001 10	0001 101
2	2	001	011	1101	0001 10	001	001
0	3	0000 0011 1	0000 111	0010 00	0010 00	0000 11	0000 0011 1
1	3	0000 0110	0010 10	0110 0	0010 01	0000 011	0001 100
2	3	0000 101	0010 01	0111 0	0010 10	0000 010	0001 011
3	3	0001 1	0101	1100	0010 11	0001 01	0000 1
0	4	0000 0001 11	0000 0111	0001 111	0011 00	0000 10	0000 0011 0
1	4	0000 0011 0	0001 10	0101 0	0011 01	0000 0011	0000 0010 1
2	4	0000 0101	0001 01	0101 1	0011 10	0000 0010	0001 010
3	4	0000 11	0100	1011	0011 11	0000 000	0000 01

Abb. C.1: Ausschnitt aus Tabelle 9.5 [H264]

level_prefix	bit string
0	1
1	01
2	001
3	0001
4	0000 1
5	0000 01

Abb. C.2: Ausschnitt aus Tabelle 9.6 [H264]

total_zeros	TotalCoeff(coeff_token)						
	1	2	3	4	5	6	7
0	1	111	0101	0001 1	0101	0000 01	0000 01
1	011	110	111	111	0100	0000 1	0000 1
2	010	101	110	0101	0011	111	101
3	0011	100	101	0100	111	110	100

Abb. C.3: Ausschnitt aus Tabelle 9.7 [H264]

run_before	zerosLeft						
	1	2	3	4	5	6	>6
0	1	1	11	11	11	11	111
1	0	01	10	10	10	000	110
2	-	00	01	01	011	001	101
3	-	-	00	001	010	011	100

Abb. C.4: Ausschnitt aus Tabelle 9.10 [H264]

Anhang D - Quellcodes der entwickelten Software

D.1 Der plattformunabhängige Teil

D.1.1 Die Klasse StreamingHandler

```
public class StreamingHandler
{
	private String[] codecs;

	private String serverAddress;
	private int serverPort;

	private RTCPHandler rtcpHandler;
	private RTSPHandler rtspHandler;
	private ArrayList<RTPHandler> rtpHandlerList;


	public StreamingHandler(String serverAddress, int serverPort, String[] codecs)
	{
		this.codecs = codecs;
		this.serverAddress = serverAddress;
		this.serverPort = serverPort;
		rtspHandler = new RTSPHandler(this,
					this.serverAddress,
					this.serverPort,
					this.codecs);
		rtpHandlerList = new ArrayList<RTPHandler>();
	}


	public void start()
	{
		rtspHandler.start();
	}


	public void end()
	{
		rtcpHandler.end();
		rtspHandler.end();
		for(RTPHandler rtpHandler : rtpHandlerList)
			rtpHandler.end();
	}


	public void createRTCPHandler(int serverPort)
	{
		rtcpHandler = new RTCPHandler(this, this.serverAddress, serverPort);
		rtcpHandler.start();
	}


	public void createRTPHandler(int playerPort,
				int gatewayPort,
				String codec,
				int clockReference)
	{
		RTPHandler rtpHandler1, rtpHandler2;
		rtpHandler1 = new RTPHandler(this,
					gatewayPort,
					playerPort,
					codec,
					clockReference);
```

```
                rtpHandler2 = new RTPHandler(this,
                                             gatewayPort + 1,
                                             playerPort + 1,
                                             null,
                                             0);
                rtpHandlerList.add(rtpHandler1);
                rtpHandlerList.add(rtpHandler2);
                rtpHandler1.start();
                rtpHandler2.start();
        }


        public String getServerAddress()
        {
                return serverAddress;
        }


        public void setServerAddress(String serverAddress)
        {
                this.serverAddress = serverAddress;
        }


        public int getServerPort()
        {
                return serverPort;
        }


        public void setServerPort(int serverPort)
        {
                this.serverPort = serverPort;
        }


        public RTCPHandler getRTCPHandler()
        {
                return rtcpHandler;
        }


        public void setRTCPHandler(RTCPHandler rtcpHandler)
        {
                this.rtcpHandler = rtcpHandler;
        }


        public RTSPHandler getRTSPHandler()
        {
                return rtspHandler;
        }


        public void setRTSPHandler(RTSPHandler rtspHandler)
        {
                this.rtspHandler = rtspHandler;
        }


        public ArrayList<RTPHandler> getRTPHandlerList()
        {
                return rtpHandlerList;
        }


        public void setRTPHandlerList(ArrayList<RTPHandler> rtpHandlerList)
        {
                this.rtpHandlerList = rtpHandlerList;
        }
}
```

D.1.2 Die Klasse RTSPHandler

```
public class RTSPHandler
{
	private StreamingHandler streamingHandler;
	private ServerHandler serverHandler;
	private PlayerHandler playerHandler;

	private String[] codecs;
	private int[] clocks;
	private String[] tracks;
	private SDPParser[] parser;


	RTSPHandler(StreamingHandler streamingHandler,
		    String serverAddress,
		    int serverPort,
		    String[] codecs)
	{
		this.streamingHandler = streamingHandler;
		serverHandler = new ServerHandler(serverAddress, serverPort);
		playerHandler = new PlayerHandler();

		this.codecs = codecs;
		tracks = new String[codecs.length];
		clocks = new int[codecs.length];
		parser = new SDPParser[codecs.length];
	}


	public void start()
	{
		serverHandler.start();
		playerHandler.start();
	}


	public void end()
	{
		serverHandler.end();
		playerHandler.end();
	}


	public ServerHandler getServerHandler()
	{
		return serverHandler;
	}


	public class ServerHandler extends Thread
	{
		private InetAddress serverAddress;
		private int serverPort;

		private Socket serverSocket;
		private InputStream serverInputStream;
		private OutputStream serverOutputStream;
		private byte[] serverPayload;
		private String serverMessage;


		public ServerHandler(String serverAddress, int serverPort)
		{
			try
			{
				this.serverAddress = InetAddress.getByName(serverAddress);
				this.serverPort = serverPort;
				this.serverSocket = new Socket(this.serverAddress,
							       this.serverPort);
				this.serverSocket.setKeepAlive(true);
			}
			catch (IOException e)
			{
				Log.e("RTSP Server Constructor",
				     "IOException\n" + e.getMessage());
			}
```

```
}


public void run()
{
        try
        {
                serverInputStream = serverSocket.getInputStream();
                serverOutputStream = serverSocket.getOutputStream();

                while(!Thread.currentThread().isInterrupted())
                {
                        if(serverInputStream.available() > 0)
                        {
                                serverPayload =
                                        new byte[serverInputStream.available()];
                                serverInputStream.read(serverPayload);
                                serverMessage = new String(serverPayload);
                                playerHandler.send(serverMessage);
                                parseCodecInformation();
                                if(serverMessage.contains("server_port="))
                                        parseServerPorts();
                        }
                }
        }
        catch (IOException e)
        {
                Log.e("RTSP Server Thread", "IOException\n" + e.getMessage());
        }
}


public void end()
{
        try
        {
                serverSocket.close();
        }
        catch (IOException e)
        {
                Log.e("RTSP Server End", "IOException\n" + e.getMessage());
        }
        this.interrupt();
}


public void send(String message)
{
        try
        {
                serverOutputStream.write(message.getBytes());
        }
        catch (IOException e)
        {
                Log.e("RTSP Server Send", "IOException\n" + e.getMessage());
        }
}


public void parseCodecInformation()
{
        if(serverMessage.contains("sdp"));
        {
                String sessionDescription =
                        serverMessage.substring(serverMessage.indexOf("\r\n\r\n")
                        + 4, serverMessage.length());
                for(int i = 0; i < codecs.length; i++)
                {
                        if (sessionDescription.contains(codecs[i]))
                        {
                                tracks[i] = parseTrack(codecs[i]);
                                clocks[i] = parseClock(codecs[i]);
                                if(codecs[i].equals("H264"))
                                        parser[i] = new ParameterSetParser();
                                if(parser[i] != null)
                                        parser[i].parse(sessionDescription);
                        }
                }
        }
}
```

```
public void parseServerPorts()
{
        int index = serverMessage.indexOf("server_port=") + 12;
        String sRtcpPort = serverMessage.substring(index, index + 5);
        Pattern fiveDigits = Pattern.compile("[0-9]{5}");
        Matcher fiveDigitsMatcher = fiveDigits.matcher(sRtcpPort);
        if(!fiveDigitsMatcher.matches())
                sRtcpPort = serverMessage.substring(index, index + 4);
        int iRtcpPort = Integer.parseInt(sRtcpPort) + 1;
        if(streamingHandler.getRTCPHandler() == null)
                streamingHandler.createRTCPHandler(iRtcpPort);
}


private String parseTrack(String codec)
{
        int index = serverMessage.indexOf(codec);
        String info = serverMessage.substring(index, serverMessage.length());
        index = info.indexOf("trackID=");
        String track;

        track = info.substring(index + "trackID=".length(),
                               index + "trackID=".length() + 5);
        Pattern digits = Pattern.compile("[0-9]{5}");
        Matcher digitsMatcher = digits.matcher(track);
        if(!digitsMatcher.matches())
        {
                track = info.substring(index + "trackID=".length(),
                                       index + "trackID=".length() + 4);
                digits = Pattern.compile("[0-9]{4}");
                digitsMatcher = digits.matcher(track);
                if(!digitsMatcher.matches())
                {
                        track = info.substring(index + "trackID=".length(),
                                               index + "trackID=".length() + 3);
                        digits = Pattern.compile("[0-9]{3}");
                        digitsMatcher = digits.matcher(track);
                        if(!digitsMatcher.matches())
                        {
                                track = info.substring(index +
                                                       "trackID=".length(),
                                                       index +
                                                       "trackID=".length() + 2);
                                digits = Pattern.compile("[0-9]{2}");
                                digitsMatcher = digits.matcher(track);
                                if(!digitsMatcher.matches())
                                {
                                        track = info.substring(index +
                                                               "trackID=".length(),
                                                               index +
                                                            "trackID=".length() + 1);
                                        digits = Pattern.compile("[0-9]{1}");
                                        digitsMatcher = digits.matcher(track);
                                        if(!digitsMatcher.matches())
                                        {
                                                track = null;
                                        }
                                }
                        }
                }
        }
        return track;
}


private int parseClock(String codec)
{
        int index = serverMessage.indexOf(codec);

        String clock = serverMessage.substring(index + codec.length() + 1,
                                               index + codec.length() + 6);
        Pattern digits = Pattern.compile("[0-9]{5}");
        Matcher digitsMatcher = digits.matcher(clock);
        if(!digitsMatcher.matches())
        {
                clock = serverMessage.substring(index + codec.length() + 1,
                                                index + codec.length() + 5);
                digits = Pattern.compile("[0-9]{4}");
                digitsMatcher = digits.matcher(clock);
                if(!digitsMatcher.matches())
```

```
                        {
                                clock = serverMessage.substring(index + codec.length()+1,
                                                                index + codec.length()+4);
                                digits = Pattern.compile("[0-9]{3}");
                                digitsMatcher = digits.matcher(clock);
                                if(!digitsMatcher.matches())
                                {
                                        clock = serverMessage.substring(index +
                                                                        codec.length() + 1,
                                                                        index +
                                                                        codec.length() +3);
                                        digits = Pattern.compile("[0-9]{2}");
                                        digitsMatcher = digits.matcher(clock);
                                        if(!digitsMatcher.matches())
                                        {
                                                clock = serverMessage.substring(index +
                                                                        codec.length() +1,
                                                                        index +
                                                                        codec.length()+2);
                                                digits = Pattern.compile("[0-9]{1}");
                                                digitsMatcher = digits.matcher(clock);
                                                if(!digitsMatcher.matches())
                                                {
                                                        return 0;
                                                }
                                        }
                                }
                        }
                }
                return Integer.parseInt(clock);
        }


        public Socket getSocket()
        {
                return serverSocket;
        }


        public SDPParser getParser(String codec)
        {
                for(int i = 0; i < codecs.length; i++)
                        if (codecs[i].equals(codec))
                                return parser[i];
                return null;
        }
}


public class PlayerHandler extends Thread
{
        private ServerSocket gatewaySocket;
        private int gatewayPort;

        private Socket playerSocket;
        private InputStream playerInputStream;
        private OutputStream playerOutputStream;
        private byte[] playerPayload;
        private String playerMessage;


        public PlayerHandler()
        {
                try
                {
                        gatewaySocket = new ServerSocket(8554);
                        gatewayPort = 7000;
                }
                catch (IOException e)
                {
                        Log.e("RTSP Player Constructor",
                             "IOException\n" + e.getMessage());
                }
        }


         public void run()
         {
                try
                {
```

```
                playerSocket = gatewaySocket.accept();
                playerSocket.setKeepAlive(true);

                playerInputStream = playerSocket.getInputStream();
                playerOutputStream = playerSocket.getOutputStream();

                while(!Thread.currentThread().isInterrupted())
                {
                        if(playerInputStream.available() > 0)
                        {
                                playerPayload =
                                        new byte[playerInputStream.available()];
                                playerInputStream.read(playerPayload);
                                playerMessage = new String(playerPayload);
                                if (playerMessage.startsWith("SETUP"))
                                        editClientPorts();
                                serverHandler.send(playerMessage);
                        }
                }
        }
        catch (IOException e)
        {
                Log.e("RTSP Player Thread", "IOException\n" + e.getMessage());
        }
}


public void end()
{
        try
        {
                playerSocket.close();
                gatewaySocket.close();
        }
        catch (IOException e)
        {
                Log.e("RTSP Player End", "IOException\n" + e.getMessage());
        }
        this.interrupt();
}


public void send(String message)
{
        try
        {
                playerOutputStream.write(message.getBytes());
        }
        catch(IOException e)
        {
                Log.e("RTSP Player Send", "IOException\n" + e.getMessage());
        }
}


 private void editClientPorts()
{
        // Edit Port Numbers
        String clientPort = "client_port=";
        int idx = playerMessage.indexOf(clientPort);
        String sPlayerPort = playerMessage.substring(idx + clientPort.length(),
                                                idx + clientPort.length()+5);
        int iPlayerPort;

        Pattern fiveDigits = Pattern.compile("[0-9]{5}");
        Matcher fiveDigitsMatcher = fiveDigits.matcher(sPlayerPort);
        if(!fiveDigitsMatcher.matches())
                sPlayerPort = playerMessage.substring(idx + clientPort.length(),
                                                idx + clientPort.length() + 4);

        iPlayerPort = Integer.parseInt(sPlayerPort);
        playerMessage = playerMessage.substring(0, idx + clientPort.length()) +
                          gatewayPort + "-" + (gatewayPort + 1) +
                          playerMessage.substring(idx + clientPort.length() +
                          2 * sPlayerPort.length() + 1, playerMessage.length());

        // Create RTP Handler for this SETUP
        boolean containsTrack = false;
        for(int i = 0; i < tracks.length; i++)
        {
```

```
                        if(tracks[i] != null && playerMessage.contains(tracks[i]))
                        {
                                idx = i;
                                containsTrack = true;
                        }
                }
                if (containsTrack)
                        streamingHandler.createRTPHandler(iPlayerPort,
                                                          gatewayPort,
                                                          codecs[idx],
                                                          clocks[idx]);
                else
                        streamingHandler.createRTPHandler(iPlayerPort,
                                                          gatewayPort,
                                                          null,
                                                          0);
                gatewayPort += 2;
            }
        }
}
```

D.1.3 Die Klasse RTPHandler

```
public class RTPHandler
{
        private StreamingHandler streamingHandler;
        private ServerHandler serverHandler;
        private PlayerHandler playerHandler;

        private String codec;
        private int clock;
        private DatagramParser parser;


        RTPHandler(StreamingHandler streamingHandler,
                   int gatewayPort,
                   int playerPort,
                   String codec,
                   int clockReference)
        {
                this.streamingHandler = streamingHandler;
                serverHandler = new ServerHandler(gatewayPort);
                playerHandler = new PlayerHandler(playerPort);
                if(codec != null)
                {
                        this.codec = codec;
                        if(codec.equals("H264"))
                        {
                                parser = new H264Parser();

                                parser.setInputParameter(this.streamingHandler.getRTSPHandler
                                        ().getServerHandler().getParser("H264").getParameter());
                        }
                }
                this.clock = clockReference;
        }


        public void start()
        {
                serverHandler.start();
        }


        public void end()
        {
                serverHandler.end();
                playerHandler.end();
        }


        public ServerHandler getServerHandler()
        {
                return serverHandler;
        }
```

```
public PlayerHandler getPlayerHandler()
{
	return playerHandler;
}


public String getCodec()
{
	return codec;
}


public DatagramParser getParser()
{
	return parser;
}


public class ServerHandler extends Thread
{
	private int gatewayPort;
	private DatagramSocket gatewaySocket;

	private byte[] serverPayload;
	private DatagramPacket serverPacket;

	// Bitrate:
	long numPackets;
	long numBits;
	long bitrate;
	long beginTimestamp;

	// Jitter
	long jitter;
	long jitterMax = 0;
	long jitterMin = 1000;
	long arrivalTimestamp, previousArrivalTimestamp;
	long rtpTimestamp, previousRTPTimestamp;

	// Loss
	long singleLosses = 0;
	long burstLosses = -1;
	long avgBurstLossDuration;
	long outOfOrderPackets = 0;
	long sequenceNumber;
	long previousSequenceNumber;

	// Defragmentation for Android MediaPlayer - Android RC, bug fixed in RC 1.5
	private int fragmentOffset;
	private byte[] fragmentedNAL;


	public ServerHandler(int gatewayPort)
	{
		try
		{
			this.gatewayPort = gatewayPort;
			gatewaySocket = new DatagramSocket(this.gatewayPort);

			serverPayload = new byte[1472];
			serverPacket = new DatagramPacket(serverPayload,
                                                  serverPayload.length);
		}
		catch (SocketException e)
		{
			Log.e("RTP Server Constructor",
			     "SocketException\n" + e.getMessage());
		}
	}


	public void run()
	{
		while(!Thread.currentThread().isInterrupted())
		{
			try
			{
				serverPacket.setLength(1472);
				gatewaySocket.receive(serverPacket);
```

```
                        // Defragmentation for Android MediaPlayer - Android RC
                                                1.1, bug fixed in RC 1.5
                        // if(codec != null && codec.equals("H264") &&
                                (serverPacket.getData()[12] & 0x1f) == 28)
                        // {
                        //      byte[] defragmentedNAL =
                                        defragmentNAL(serverPacket);
                        //      if(defragmentedNAL != null)
                        //              playerHandler.send(defragmentedNAL);
                        //
                        // else
                                playerHandler.send(serverPacket);

                        // Bitrate
                        arrivalTimestamp = (new Date()).getTime();
                        calculateBitrate(serverPacket.getLength());

                        if(codec != null && clock != 0)
                        {
                                // Jitter
                                rtpTimestamp =
                                        parseRTPTimestamp(serverPacket.getData());
                                calculateJitter(rtpTimestamp, arrivalTimestamp);
                                // Loss
                                sequenceNumber =
                                   parseRTPSequenceNumber(serverPacket.getData());
                                calculateLoss(sequenceNumber);
                                // Deep Inspection
                                if(parser != null)
                                        parser.parse(serverPacket);
                        }
                }
                catch (IOException e)
                {
                        Log.e("RTP Server Thread",
                             "IOException\n" + e.getMessage());
                }
        }
}


public void end()
{
        gatewaySocket.close();
        this.interrupt();
}


// Defragmentation for Android MediaPlayer - Android 1.1, bug fixed in 1.5
public byte[] defragmentNAL(DatagramPacket rtpPacket)
{
        byte[] fragment = rtpPacket.getData();
        // if FU-A Start-Flag == 1
        if((rtpPacket.getData()[13] & 0x80) == 128)
        {
                fragmentOffset = 0;
                fragmentedNAL = new byte[16384];
        }

        if (fragmentedNAL != null)
        {
                // First Fragment + RTP-Header
                if(fragmentOffset == 0)
                {
                        for(int j = 0; j < rtpPacket.getLength() - 1; j++)
                        {
                                // RTP-Header
                                if(j < 12)
                                        fragmentedNAL[j] = fragment[j];
                                // NAL-Header
                                if(j == 12)
                                        fragmentedNAL[12] = (byte)((fragment[12] &
                                                0xe0) | (fragment[13] & 0x1f));
                                // NAL-Fragment
                                if(j > 12)
                                        fragmentedNAL[j] = fragment[j + 1];
                        }
                        fragmentOffset += (rtpPacket.getLength() - 1);
                }
                // Other Fragments
```

```
                else
                {
                        // NAL-Fragment
                        for(int j = 0; j < rtpPacket.getLength() - 14; j++)
                                fragmentedNAL[j + fragmentOffset] =
                                                                fragment[j + 14];
                        fragmentOffset += (rtpPacket.getLength() - 14);
                }
        }
        // if FU-A End-Flag == 1
        if((rtpPacket.getData()[13] & 0x40) == 64)
                return fragmentedNAL;
        return null;
}


public void calculateBitrate(int length)
{
        numBits += 8 * length;
        if(numPackets == 0)
                beginTimestamp = arrivalTimestamp;
        else if(((arrivalTimestamp - beginTimestamp) / 1000) > 0)
                bitrate = numBits / ((arrivalTimestamp - beginTimestamp) / 1000)
                                                                        / 1024;
        numPackets++;
}


public void calculateJitter(long rtpTimestamp, long arrivalTimestamp)
{
        if(previousArrivalTimestamp != 0 && previousRTPTimestamp != 0)
        {
                long d = ((previousRTPTimestamp - rtpTimestamp) * 1000) /
                           clock - (previousArrivalTimestamp - arrivalTimestamp);
                jitter += (Math.abs(d) - jitter) / 16;
                if(jitter > jitterMax && numPackets > 16)
                        jitterMax = jitter;
                if(jitter < jitterMin && numPackets > 16)
                        jitterMin = jitter;
        }
        previousRTPTimestamp = rtpTimestamp;
        previousArrivalTimestamp = arrivalTimestamp;
}


public void calculateLoss(long sequenceNumber)
{
        boolean singleLoss = false;
        boolean burstLoss = false;
        boolean outOfOrder = false;

        if(sequenceNumber == 0)
                previousSequenceNumber = -1;

        if(sequenceNumber - previousSequenceNumber == 2)
                singleLoss = true;
        if(sequenceNumber - previousSequenceNumber > 2)
                burstLoss = true;
        if(sequenceNumber - previousSequenceNumber < 0)
                outOfOrder = true;

        if(singleLoss)
                singleLosses++;
        if(burstLoss && burstLosses >= 0)
                avgBurstLossDuration = ((avgBurstLossDuration * burstLosses) +
                                        (sequenceNumber - previousSequenceNumber) /
                                        ++burstLosses;
        else if(burstLoss)
                burstLosses++;
        if(outOfOrder)
                outOfOrderPackets++;

        previousSequenceNumber = sequenceNumber;
}


public long parseRTPTimestamp(byte[] rtpPacket)
{
        return (long)(((rtpPacket[4] & 0xff) << 24) |
                        ((rtpPacket[5] & 0xff) << 16) |
```

```
                                ((rtpPacket[6] & 0xff) << 8) |
                                 (rtpPacket[7] & 0xff));
        }


        public long parseRTPSequenceNumber(byte[] rtpPacket)
        {
                return (long)(((rtpPacket[2] & 0xff) << 8) | (rtpPacket[3] & 0xff));
        }


        public long getBitrate()
        {
                return bitrate;
        }


        public long getNumPackets()
        {
                return numPackets;
        }


        public long getNumBits()
        {
                return numBits;
        }


        public long[] getJitter()
        {
                long[] jitterValues = new long[3];
                jitterValues[0] = jitter;
                jitterValues[1] = jitterMax;
                jitterValues[2] = jitterMin;
                return jitterValues;
        }


        public long getSingleLosses()
        {
                return singleLosses;
        }


        public long getBurstLosses()
        {
                return burstLosses;
        }


        public long getAvgBurstLossDuration()
        {
                return avgBurstLossDuration;
        }


        public long getOutOfOrder()
        {
                return outOfOrderPackets;
        }


        public DatagramSocket getSocket()
        {
                return gatewaySocket;
        }
}


public class PlayerHandler
{
        private InetAddress playerAddress;
        private int playerPort;
        private DatagramSocket playerSocket;
        private byte[] playerPayload;
        private DatagramPacket playerPacket;
```

```
                public PlayerHandler(int playerPort)
                {
                        try
                        {
                                this.playerAddress = InetAddress.getByName("localhost");
                                this.playerPort = playerPort;
                                playerSocket = new DatagramSocket();
                                playerPayload = new byte[1472];
                                playerPacket = new DatagramPacket(playerPayload,
                                                                  playerPayload.length,
                                                                  this.playerAddress,
                                                                  this.playerPort);
                        }
                        catch (UnknownHostException e)
                        {
                                Log.e("RTP Player Constructor",
                                     "UnknownException\n" + e.getMessage());
                        }
                        catch (SocketException e)
                        {
                                Log.e("RTP Player Constructor",
                                     "SocketException\n" + e.getMessage());
                        }
                }


                public void send(DatagramPacket serverPacket)
                {
                        try
                        {
                                playerPacket.setData(serverPacket.getData());
                                playerPacket.setLength(serverPacket.getLength());
                                playerSocket.send(playerPacket);
                        }
                        catch (IOException e)
                        {
                                Log.e("RTP Player Send", "IOException\n" + e.getMessage());
                        }
                }


                public void send(byte[] serverPacket)
                {
                        try
                        {
                                playerPacket.setData(serverPacket);
                                playerPacket.setLength(serverPacket.length);
                                playerSocket.send(playerPacket);
                        }
                        catch (IOException e)
                        {
                                Log.e("RTP Player Send", "IOException\n" + e.getMessage());
                        }
                }


                public void end()
                {
                        playerSocket.close();
                }


                public int getPlayerPortNumber()
                {
                        return playerHandler.playerPort;
                }
        }
}
```

D.1.4 Die Klasse RTCPHandler

```
public class RTCPHandler
{
        private StreamingHandler streamingHandler;
        private ServerHandler serverHandler;
```

```
private PlayerHandler playerHandler;


public RTCPHandler(StreamingHandler streamingHandler,
                   String serverAddress,
                   int serverPort)
{
        this.streamingHandler = streamingHandler;
        serverHandler = new ServerHandler(serverAddress, serverPort);
        playerHandler = new PlayerHandler(serverPort);
}


public void start()
{
        playerHandler.start();
}


public void end()
{
        playerHandler.end();
}


public class PlayerHandler extends Thread
{
        private int gatewayPort;
        private DatagramSocket gatewaySocket;

        private int rtcpPlayerPort;
        private byte[] playerPayload;
        private DatagramPacket playerPacket;


        public PlayerHandler(int gatewayPort)
        {
                try
                {
                        this.gatewayPort = gatewayPort;
                        gatewaySocket = new DatagramSocket(this.gatewayPort);

                        playerPayload = new byte[1460];
                        playerPacket = new DatagramPacket(playerPayload,
                                                          playerPayload.length);
                }
                catch (SocketException e)
                {
                        Log.e("RTCP Player Constructor",
                             "SocketException\n" + e.getMessage());
                }

        }


        public void run()
        {
                while(!Thread.currentThread().isInterrupted())
                {
                        try
                        {
                                playerPacket.setLength(1460);
                                gatewaySocket.receive(playerPacket);
                                rtcpPlayerPort = playerPacket.getPort();
                                serverHandler.send(playerPacket, rtcpPlayerPort);
                        }
                        catch(IOException e)
                        {
                                Log.e("RTCP Player Thread",
                                     "IOException\n" + e.getMessage());
                        }
                }
        }


        public void end()
        {
                gatewaySocket.close();
                this.interrupt();
        }
```

```
        }

        public class ServerHandler
        {
                private int serverPort;
                private byte[] serverPayload;
                private DatagramPacket serverPacket;


                public ServerHandler(String serverAddress, int serverPort)
                {
                        try
                        {
                        this.serverPort = serverPort;
                        serverPayload = new byte[1460];
                        serverPacket = new DatagramPacket(serverPayload,
                                                          serverPayload.length,
                                                          InetAddress.getByName(serverAddress),
                                                          this.serverPort);
                        }
                        catch (UnknownHostException e)
                        {
                                Log.e("RTCP Server Constructor",
                                     "UnknownHostException\n" + e.getMessage());
                        }
                }


                public void send(DatagramPacket playerPacket, int rtcpPlayerPort)
                {
                        try
                        {
                                serverPacket.setLength(playerPacket.getLength());
                                serverPacket.setData(playerPacket.getData());
                                findRTPServerSocket(rtcpPlayerPort).send(serverPacket);
                        }
                        catch (IOException e)
                        {
                                Log.e("RTCP Server Send", "IOException\n" + e.getMessage());
                        }
                }


                public DatagramSocket findRTPServerSocket(int rtcpPlayerPort)
                {
                for(RTPHandler rtpHandler : streamingHandler.getRTPHandlerList())
                {
                        if(rtpHandler.getPlayerHandler().getPlayerPortNumber() ==
                                                                        rtcpPlayerPort)
                        {
                                return rtpHandler.getServerHandler().getSocket();
                        }
                }
                return null;
                }
        }
}
```

D.1.5 Die Schnittstelle SDPParser

```
public interface SDPParser
{
        public void parse(String sdp);

        public ArrayList<String> getParameter();
}
```

D.1.6 Die Klasse ParameterSetParser

```
public class ParameterSetParser implements SDPParser
```

```
{
	private ArrayList<String> parameter;
	private ArrayList<String> sequenceParameterSet;
	private ArrayList<String> pictureParameterSet;
	private String packetizationMode;

	private ExpGolomb expGolomp;
	private Base64 base64;


	public ParameterSetParser()
	{
		parameter = new ArrayList<String>();
		sequenceParameterSet = new ArrayList<String>();
		pictureParameterSet = new ArrayList<String>();

		expGolomp = new ExpGolomb();
		base64 = new Base64();
	}


	@Override
	public ArrayList<String> getParameter()
	{
		return parameter;
	}


	@Override
	public void parse(String sdp)
	{
		int begin, end, endAttribut;
		String help;

		begin = sdp.indexOf("sprop-parameter-sets") + 21;
		help = sdp.substring(begin, sdp.length() - 1);
		endAttribut = help.indexOf("\n") - 1;
		help = help.substring(0, endAttribut);
		if(help.contains(";"))
		{
			endAttribut = help.indexOf(";");
			help = sdp.substring(begin, endAttribut);
		}

		while(help.contains(","))
		{
			end = help.indexOf(",");
			String parameterSet = help.substring(0, end);
			byte[] ps = base64.decode(parameterSet);
			if((ps[0] & 0x1f) == 7)
				parseSPS(ps);
			if((ps[0] & 0x1f) == 8)
				parsePPS(ps);
			help = help.substring(end + 1, help.length());
		}
		String parameterSet = help.substring(0, help.length());
		byte[] ps = base64.decode(parameterSet);
		if((ps[0] & 0x1f) == 7)
			parseSPS(ps);
		if((ps[0] & 0x1f) == 8)
			parsePPS(ps);

		parsePacketizationMode(sdp);

		addParameter();
	}


	public void parseSPS(byte[] sps)
	{
		sequenceParameterSet.add("<sequence_parameter_set>");

		// profile_idc
		expGolomp.jumpXBits(sps, 1, 0, 8);
		sequenceParameterSet.add("<profile_idc>"  + expGolomp.getValue());
		int profile = expGolomp.getValue();

		// constraint_flags
		expGolomp.jumpXBits(sps, 2, 0, 8);
		sequenceParameterSet.add("<constraint_flags>"  + expGolomp.getValue());
```

```
// level_idc
expGolomp.jumpXBits(sps, 3, 0, 8);
sequenceParameterSet.add("<level_idc>"  + expGolomp.getValue());

// sequence_parameter_set_id
expGolomp.parseUE(sps, 4, 0);
sequenceParameterSet.add("<sequence_parameter_set_id>" + expGolomp.getValue());


if(profile == 100 || profile == 110 || profile == 122 || profile == 144)
{
        // TODO ITU-T H.264 7.3.2.1 - High Profile Extensions
        // Default values are used for other profiles
}

// log2_max_frame_num_minus4 - indicates max. frame number, if frame number ==
                        max frame number: next NAL is IDR and frame number = 0
expGolomp.parseUE(sps,
                expGolomp.getNextByteOffset(),
                expGolomp.getNextBitOffset());
sequenceParameterSet.add("<log2_max_frame_num_minus4>" + expGolomp.getValue());

// pic_order_cnt_type - ITU-T H.264 8.2.1
expGolomp.parseUE(sps,
                expGolomp.getNextByteOffset(),
                expGolomp.getNextBitOffset());
sequenceParameterSet.add("<pic_order_cnt_type>" + expGolomp.getValue());
int pic_order_cnt_type = expGolomp.getValue();

if(pic_order_cnt_type == 0)
{
        // log2_max_pic_order_cnt_lsb_minus4 - ITU-T H.264 8.2.1.1
        expGolomp.parseUE(sps,
                        expGolomp.getNextByteOffset(),
                        expGolomp.getNextBitOffset());
        sequenceParameterSet.add("<log2_max_pic_order_cnt_lsb_minus4>" +
                                                        expGolomp.getValue());
}
if(pic_order_cnt_type == 1)
{
        // delta_pic_order_always_zero_flag
        expGolomp.jumpXBits(sps,
                          expGolomp.getNextByteOffset(),
                          expGolomp.getNextBitOffset(),
                          1);
        sequenceParameterSet.add("<delta_pic_order_always_zero_flag>"  +
                                                        expGolomp.getValue());

        // offset_for_non_ref_pic
        expGolomp.parseSE(sps,
                        expGolomp.getNextByteOffset(),
                        expGolomp.getNextBitOffset());
        pictureParameterSet.add("<offset_for_non_ref_pic>" +
                                                        expGolomp.getValue());

        // offset_for_top_to_bottom_field
        expGolomp.parseSE(sps,
                        expGolomp.getNextByteOffset(),
                        expGolomp.getNextBitOffset());
        pictureParameterSet.add("<offset_for_top_to_bottom_field>" +
                                                        expGolomp.getValue());

        // num_ref_frames_in_pic_order_cnt_cycle
        expGolomp.parseUE(sps,
                        expGolomp.getNextByteOffset(),
                        expGolomp.getNextBitOffset());
        sequenceParameterSet.add("<num_ref_frames_in_pic_order_cnt_cycle>" +
                                                        expGolomp.getValue());
        int num_ref_frames_in_pic_order_cnt_cycle = expGolomp.getValue();

        for(int i = 0; i < num_ref_frames_in_pic_order_cnt_cycle; i++)
        {
                // offset_for_ref_frame[i]
                expGolomp.parseUE(sps,
                                expGolomp.getNextByteOffset(),
                                expGolomp.getNextBitOffset());
                sequenceParameterSet.add("<offset_for_ref_frame["+i+"]>" +
                                                        expGolomp.getValue());
        }
```

```
}

// num_ref_pics - indicates max. size of reference lists l0 & l1
expGolomp.parseUE(sps,
                  expGolomp.getNextByteOffset(),
                  expGolomp.getNextBitOffset());
sequenceParameterSet.add("<num_ref_pics>" + expGolomp.getValue());

// gaps_in_frame_num_value_allowed_flag - indicates if gaps in frame number are
                                                                        allowed
expGolomp.jumpXBits(sps,
                    expGolomp.getNextByteOffset(),
                    expGolomp.getNextBitOffset(),
                    1);
sequenceParameterSet.add("<gaps_in_frame_num_value_allowed_flag>"  +
                                                        expGolomp.getValue());

// pic_width_in_mbs_minus1 - indicates pic width, luma width = 16 *
       (pic_width_in_mbs_minus1 + 1), chroma width depends on chroma_format_idc
expGolomp.parseUE(sps,
                  expGolomp.getNextByteOffset(),
                  expGolomp.getNextBitOffset());
sequenceParameterSet.add("<pic_width_in_mbs_minus1>" + expGolomp.getValue());

// pic_height_in_map_units_minus1 - indicates pic height, luma height = 16 *
                                          (pic_height_in_map_units_minus1 + 1)
expGolomp.parseUE(sps,
                  expGolomp.getNextByteOffset(),
                  expGolomp.getNextBitOffset());
sequenceParameterSet.add("<pic_height_in_map_units_minus1>" +
                                                        expGolomp.getValue());

// frame_mbs_only_flag - frames or fields (baseline profile:
                                                    frame_mbs_only_flag == 1)
expGolomp.jumpXBits(sps,
                    expGolomp.getNextByteOffset(),
                    expGolomp.getNextBitOffset(),
                    1);
sequenceParameterSet.add("<frame_mbs_only_flag>" + expGolomp.getValue());
int frame_mbs_only_flag = expGolomp.getValue();

if(frame_mbs_only_flag == 0)
{
        // mb_adaptive_frame_field_flag
        expGolomp.jumpXBits(sps,
                            expGolomp.getNextByteOffset(),
                            expGolomp.getNextBitOffset(),
                            1);
        sequenceParameterSet.add("<mb_adaptive_frame_field_flag>" +
                                                        expGolomp.getValue());
}

// direct_8x8_inference_flag - ITU-T H.264 8.4.1.2 (baseline profile:
                                                  direct_8x8_inference_flag == 0)
expGolomp.jumpXBits(sps,
                    expGolomp.getNextByteOffset(),
                    expGolomp.getNextBitOffset(),
                    1);
sequenceParameterSet.add("<direct_8x8_inference_flag>" + expGolomp.getValue());

// frame_cropping_flag - indicates if frame cropping parameters are present
expGolomp.jumpXBits(sps,
                    expGolomp.getNextByteOffset(),
                    expGolomp.getNextBitOffset(),
                    1);
sequenceParameterSet.add("<frame_cropping_flag>" + expGolomp.getValue());
int frame_cropping_flag = expGolomp.getValue();

if(frame_cropping_flag == 1)
{
        // frame_crop_left_offset
        expGolomp.parseUE(sps,
                          expGolomp.getNextByteOffset(),
                          expGolomp.getNextBitOffset());
        sequenceParameterSet.add("<frame_crop_left_offset>" +
                                                        expGolomp.getValue());

        // frame_crop_right_offset
        expGolomp.parseUE(sps,
                          expGolomp.getNextByteOffset(),
```

```
                        expGolomp.getNextBitOffset());
        sequenceParameterSet.add("<frame_crop_right_offset>" +
                                                        expGolomp.getValue());

        // frame_crop_top_offset
        expGolomp.parseUE(sps,
                        expGolomp.getNextByteOffset(),
                        expGolomp.getNextBitOffset());
        sequenceParameterSet.add("<frame_crop_top_offset>" +
                                                        expGolomp.getValue()
        // frame_crop_bottom_offset
        expGolomp.parseUE(sps,
                        expGolomp.getNextByteOffset(),
                        expGolomp.getNextBitOffset());
        sequenceParameterSet.add("<frame_crop_bottom_offset>" +
                                                        expGolomp.getValue());
}

// vui_parameters_present_flag
expGolomp.jumpXBits(sps,
                    expGolomp.getNextByteOffset(),
                    expGolomp.getNextBitOffset(),
                    1);
sequenceParameterSet.add("<vui_parameters_present_flag>" +
                                                        expGolomp.getValue());
int vui_parameters_present_flag = expGolomp.getValue();

if(vui_parameters_present_flag == 1)
{
        // aspect_ratio_info_present_flag
        expGolomp.jumpXBits(sps,
                        expGolomp.getNextByteOffset(),
                        expGolomp.getNextBitOffset(),
                        1);
        sequenceParameterSet.add("<aspect_ratio_info_present_flag>" +
                                                        expGolomp.getValue());
        int aspect_ratio_info_present_flag = expGolomp.getValue();

        if(aspect_ratio_info_present_flag == 1)
        {
                // aspect_ratio_idc
                expGolomp.jumpXBits(sps,
                                expGolomp.getNextByteOffset(),
                                expGolomp.getNextBitOffset(),
                                8);
                sequenceParameterSet.add("<aspect_ratio_idc>" +
                                                        expGolomp.getValue());
                int aspect_ratio_idc = expGolomp.getValue();

                if(aspect_ratio_idc == 255)
                {
                        // sar_width
                        expGolomp.jumpXBits(sps,
                                        expGolomp.getNextByteOffset(),
                                        expGolomp.getNextBitOffset(),
                                        16);
                        sequenceParameterSet.add("<sar_width>" +
                                                        expGolomp.getValue());

                        // sar_height
                        expGolomp.jumpXBits(sps,
                                        expGolomp.getNextByteOffset(),
                                        expGolomp.getNextBitOffset(),
                                        16);
                        sequenceParameterSet.add("<sar_height>" +
                                                        expGolomp.getValue());
                }
        }

        // overscan_info_present_flag
        expGolomp.jumpXBits(sps,
                        expGolomp.getNextByteOffset(),
                        expGolomp.getNextBitOffset(),
                        1);
        sequenceParameterSet.add("<overscan_info_present_flag>" +
                                                        expGolomp.getValue());
        int overscan_info_present_flag = expGolomp.getValue();

        if(overscan_info_present_flag == 1)
        {
```

```
        // overscan_appropriate_flag
        expGolomp.jumpXBits(sps,
                            expGolomp.getNextByteOffset(),
                            expGolomp.getNextBitOffset(),
                            1);
        sequenceParameterSet.add("<overscan_appropriate_flag>" +
                                                    expGolomp.getValue());
}


// video_signal_type_present_flag
expGolomp.jumpXBits(sps,
                    expGolomp.getNextByteOffset(),
                    expGolomp.getNextBitOffset(),
                    1);
sequenceParameterSet.add("<video_signal_type_present_flag>" +
                                                    expGolomp.getValue());
int video_signal_type_present_flag = expGolomp.getValue();

if(video_signal_type_present_flag == 1)
{
        // video_format - NTSC, PAL, ...
        expGolomp.jumpXBits(sps,
                            expGolomp.getNextByteOffset(),
                            expGolomp.getNextBitOffset(),
                            3);
        sequenceParameterSet.add("<video_format>" +
                                                   expGolomp.getValue());

        // video_full_range_flag
        expGolomp.jumpXBits(sps,
                            expGolomp.getNextByteOffset(),
                            expGolomp.getNextBitOffset(),
                            1);
        sequenceParameterSet.add("<video_full_range_flag>" +
                                                    expGolomp.getValue());

        // colour_description_present_flag
        expGolomp.jumpXBits(sps,
                            expGolomp.getNextByteOffset(),
                            expGolomp.getNextBitOffset(),
                            1);
        sequenceParameterSet.add("<colour_description_present_flag>" +
                                                    expGolomp.getValue());
        int colour_description_present_flag = expGolomp.getValue();

        if(colour_description_present_flag == 1)
        {
                // colour_primaries
                expGolomp.jumpXBits(sps,
                                    expGolomp.getNextByteOffset(),
                                    expGolomp.getNextBitOffset(),
                                    8);
                sequenceParameterSet.add("<colour_primaries>" +
                                                    expGolomp.getValue());

                // transfer_characteristics
                expGolomp.jumpXBits(sps,
                                    expGolomp.getNextByteOffset(),
                                    expGolomp.getNextBitOffset(),
                                    8);
                sequenceParameterSet.add("<transfer_characteristics>" +
                                                    expGolomp.getValue());

                // matrix_coefficients
                expGolomp.jumpXBits(sps,
                                    expGolomp.getNextByteOffset(),
                                    expGolomp.getNextBitOffset(),
                                    8);
                sequenceParameterSet.add("<matrix_coefficients>" +
                                                    expGolomp.getValue());
        }
}

// chroma_loc_info_present_flag
expGolomp.jumpXBits(sps,
                    expGolomp.getNextByteOffset(),
                    expGolomp.getNextBitOffset(),
                    1);
sequenceParameterSet.add("<chroma_loc_info_present_flag>" +
```

```
                                                        expGolomp.getValue());
int chroma_loc_info_present_flag = expGolomp.getValue();

if(chroma_loc_info_present_flag == 1)
{
        // chroma_sample_loc_type_top_field
        expGolomp.parseUE(sps,
                          expGolomp.getNextByteOffset(),
                          expGolomp.getNextBitOffset());

        sequenceParameterSet.add("<chroma_sample_loc_type_top_field>" +
                                                        expGolomp.getValue());

        // chroma_sample_loc_type_bottom_field
        expGolomp.parseUE(sps,
                          expGolomp.getNextByteOffset(),
                          expGolomp.getNextBitOffset());
        sequenceParameterSet.add("<chroma_sample_loc_type_bottom_field>"
                                                      + expGolomp.getValue());
}

// timing_info_present_flag
expGolomp.jumpXBits(sps,
                    expGolomp.getNextByteOffset(),
                    expGolomp.getNextBitOffset(),
                    1);
sequenceParameterSet.add("<timing_info_present_flag>" +
                                                        expGolomp.getValue());
int timing_info_present_flag = expGolomp.getValue();

if(timing_info_present_flag == 1)
{
        // num_units_in_tick
        expGolomp.jumpXBits(sps,
                            expGolomp.getNextByteOffset(),
                            expGolomp.getNextBitOffset(),
                            32);
        sequenceParameterSet.add("<num_units_in_tick>" +
                                                        expGolomp.getValue());

        // time_scale
        expGolomp.jumpXBits(sps,
                            expGolomp.getNextByteOffset(),
                            expGolomp.getNextBitOffset(),
                            32);
        sequenceParameterSet.add("<time_scale>" + expGolomp.getValue());

        // fixed_frame_rate_flag
        expGolomp.jumpXBits(sps,
                            expGolomp.getNextByteOffset(),
                            expGolomp.getNextBitOffset(),
                            1);
        sequenceParameterSet.add("<fixed_frame_rate_flag>" +
                                                        expGolomp.getValue());
}

// nal_hrd_parameters_present_flag
expGolomp.jumpXBits(sps,
                    expGolomp.getNextByteOffset(),
                    expGolomp.getNextBitOffset(),
                    1);
sequenceParameterSet.add("<nal_hrd_parameters_present_flag>" +
                                                        expGolomp.getValue());
int nal_hrd_parameters_present_flag = expGolomp.getValue();

if(nal_hrd_parameters_present_flag == 1)
{
        // cpb_cnt_minus1
        expGolomp.parseUE(sps,
                          expGolomp.getNextByteOffset(),
                          expGolomp.getNextBitOffset());
        sequenceParameterSet.add("<cpb_cnt_minus1>" +
                                                        expGolomp.getValue());
        int cpb_cnt_minus1 = expGolomp.getValue();

        // bit_rate_scale
        expGolomp.jumpXBits(sps,
                            expGolomp.getNextByteOffset(),
                            expGolomp.getNextBitOffset(),
                            4);
```

```
                sequenceParameterSet.add("<bit_rate_scale>" +
                                                        expGolomp.getValue());

                // cpb_size_scale
                expGolomp.jumpXBits(sps,
                                    expGolomp.getNextByteOffset(),
                                    expGolomp.getNextBitOffset(),
                                    4);
                sequenceParameterSet.add("<cpb_size_scale>" +
                                                        expGolomp.getValue());
                for(int i = 0; i <= cpb_cnt_minus1; i++)
                {
                        // bit_rate_value_minus1[i]
                        expGolomp.parseUE(sps,
                                          expGolomp.getNextByteOffset(),
                                          expGolomp.getNextBitOffset());
                        sequenceParameterSet.add("<bit_rate_value_minus1["+i+"]>"
                                                    + expGolomp.getValue());

                        // cpb_size_value_minus1[i]
                        expGolomp.parseUE(sps,
                                          expGolomp.getNextByteOffset(),
                                          expGolomp.getNextBitOffset());
                        sequenceParameterSet.add("<cpb_size_value_minus1["+i+"]>"
                                                    + expGolomp.getValue());

                        // cbr_flag[i]
                        expGolomp.jumpXBits(sps,
                                            expGolomp.getNextByteOffset(),
                                            expGolomp.getNextBitOffset(),
                                            1);
                        sequenceParameterSet.add("<cbr_flag["+i+"]>" +
                                                        expGolomp.getValue());
                }

                // initial_cpb_removal_delay_length_minus1
                expGolomp.jumpXBits(sps,
                                    expGolomp.getNextByteOffset(),
                                    expGolomp.getNextBitOffset(),
                                    5);

                sequenceParameterSet.add(
                                    "<initial_cpb_removal_delay_length_minus1>" +
                                                        expGolomp.getValue());

                // cpb_removal_delay_length_minus1
                expGolomp.jumpXBits(sps,
                                    expGolomp.getNextByteOffset(),
                                    expGolomp.getNextBitOffset(),
                                    5);
                sequenceParameterSet.add("<cpb_removal_delay_length_minus1>" +
                                                        expGolomp.getValue());

                // dpb_output_delay_length_minus1
                expGolomp.jumpXBits(sps,
                                    expGolomp.getNextByteOffset(),
                                    expGolomp.getNextBitOffset(),
                                    5);
                sequenceParameterSet.add("<dpb_output_delay_length_minus1>" +
                                                        expGolomp.getValue());

                // time_offset_length
                expGolomp.jumpXBits(sps,
                                    expGolomp.getNextByteOffset(),
                                    expGolomp.getNextBitOffset(),
                                    5);
                sequenceParameterSet.add("<time_offset_length>" +
                                                        expGolomp.getValue());
        }

        // vcl_hrd_parameters_present_flag
        expGolomp.jumpXBits(sps,
                            expGolomp.getNextByteOffset(),
                            expGolomp.getNextBitOffset(),
                            1);
        sequenceParameterSet.add("<nal_hrd_parameters_present_flag>" +
                                                        expGolomp.getValue());
        int vcl_hrd_parameters_present_flag = expGolomp.getValue();

        if(vcl_hrd_parameters_present_flag == 1)
```

```
{
        // cpb_cnt_minus1
        expGolomp.parseUE(sps,
                          expGolomp.getNextByteOffset(),
                          expGolomp.getNextBitOffset());
        sequenceParameterSet.add("<cpb_cnt_minus1>" +
                                                 expGolomp.getValue());
        int cpb_cnt_minus1 = expGolomp.getValue();


        // bit_rate_scale
        expGolomp.jumpXBits(sps,
                            expGolomp.getNextByteOffset(),
                            expGolomp.getNextBitOffset(),
                            4);
        sequenceParameterSet.add("<bit_rate_scale>" +
                                                 expGolomp.getValue());

        // cpb_size_scale
        expGolomp.jumpXBits(sps,
                            expGolomp.getNextByteOffset(),
                            expGolomp.getNextBitOffset(),
                            4);
        sequenceParameterSet.add("<cpb_size_scale>" +
                                                 expGolomp.getValue());

        for(int i = 0; i <= cpb_cnt_minus1; i++)
        {
                // bit_rate_value_minus1[i]
                expGolomp.parseUE(sps,
                                  expGolomp.getNextByteOffset(),
                                  expGolomp.getNextBitOffset());
                sequenceParameterSet.add("<bit_rate_value_minus1["+i+"]>"
                                                       + expGolomp.getValue());

                // cpb_size_value_minus1[i]
                expGolomp.parseUE(sps,
                                  expGolomp.getNextByteOffset(),
                                  expGolomp.getNextBitOffset());
                sequenceParameterSet.add("<cpb_size_value_minus1["+i+"]>"
                                                       + expGolomp.getValue());

                // cbr_flag[i]
                expGolomp.jumpXBits(sps,
                                    expGolomp.getNextByteOffset(),
                                    expGolomp.getNextBitOffset(),
                                    1);
                sequenceParameterSet.add("<cbr_flag["+i+"]>" +
                                                 expGolomp.getValue());
        }

        // initial_cpb_removal_delay_length_minus1
        expGolomp.jumpXBits(sps,
                            expGolomp.getNextByteOffset(),
                            expGolomp.getNextBitOffset(),
                            5);
        sequenceParameterSet.add(
                            "<initial_cpb_removal_delay_length_minus1>" +
                                                 expGolomp.getValue());

        // cpb_removal_delay_length_minus1
        expGolomp.jumpXBits(sps,
                            expGolomp.getNextByteOffset(),
                            expGolomp.getNextBitOffset(),
                            5);
        sequenceParameterSet.add("<cpb_removal_delay_length_minus1>" +
                                                 expGolomp.getValue());

        // dpb_output_delay_length_minus1
        expGolomp.jumpXBits(sps,
                            expGolomp.getNextByteOffset(),
                            expGolomp.getNextBitOffset(),
                            5);
        sequenceParameterSet.add("<dpb_output_delay_length_minus1>" +
                                                 expGolomp.getValue());

        // time_offset_length
        expGolomp.jumpXBits(sps,
                            expGolomp.getNextByteOffset(),
                            expGolomp.getNextBitOffset(),
```

```
                                5);
        sequenceParameterSet.add("<time_offset_length>" +
                                                        expGolomp.getValue());
}

if(vcl_hrd_parameters_present_flag == 1 ||
                                    nal_hrd_parameters_present_flag == 1)
{
        // low_delay_hrd_flag
        expGolomp.jumpXBits(sps,
                            expGolomp.getNextByteOffset(),
                            expGolomp.getNextBitOffset(),
                            1);
        sequenceParameterSet.add("<low_delay_hrd_flag>" +
                                                        expGolomp.getValue());
}

// pic_struct_present_flag
expGolomp.jumpXBits(sps,
                    expGolomp.getNextByteOffset(),
                    expGolomp.getNextBitOffset(),
                    1);
sequenceParameterSet.add("<pic_struct_present_flag>" +
                                                        expGolomp.getValue());

// bitstream_restriction_flag
expGolomp.jumpXBits(sps,
                    expGolomp.getNextByteOffset(),
                    expGolomp.getNextBitOffset(), 1);
sequenceParameterSet.add("<bitstream_restriction_flag>" +
                                                        expGolomp.getValue());
int bitstream_restriction_flag = expGolomp.getValue();

if(bitstream_restriction_flag == 1)
{
        // motion_vectors_over_pic_boundaries_flag
        expGolomp.jumpXBits(sps,
                            expGolomp.getNextByteOffset(),
                            expGolomp.getNextBitOffset(),
                            1);
        sequenceParameterSet.add(
                            "<motion_vectors_over_pic_boundaries_flag>" +
                                                        expGolomp.getValue());

        // max_bytes_per_pic_denom
        expGolomp.parseUE(sps,
                          expGolomp.getNextByteOffset(),
                          expGolomp.getNextBitOffset());
        sequenceParameterSet.add("<max_bytes_per_pic_denom>" +
                                                        expGolomp.getValue());

        // max_bits_per_mb_denom
        expGolomp.parseUE(sps,
                          expGolomp.getNextByteOffset(),
                          expGolomp.getNextBitOffset());
        sequenceParameterSet.add("<max_bits_per_mb_denom>" +
                                                        expGolomp.getValue());

        // log2_max_mv_length_horizontal
        expGolomp.parseUE(sps,
                          expGolomp.getNextByteOffset(),
                          expGolomp.getNextBitOffset());
        sequenceParameterSet.add("<log2_max_mv_length_horizontal>" +
                                                        expGolomp.getValue());

        // log2_max_mv_length_vertical
        expGolomp.parseUE(sps,
                          expGolomp.getNextByteOffset(),
                          expGolomp.getNextBitOffset());
        sequenceParameterSet.add("<log2_max_mv_length_vertical>" +
                                                        expGolomp.getValue());

        // num_reorder_frames
        expGolomp.parseUE(sps,
                          expGolomp.getNextByteOffset(),
                          expGolomp.getNextBitOffset());
        sequenceParameterSet.add("<num_reorder_frames>" +
                                                        expGolomp.getValue());

        // max_dec_frame_buffering
```

```
                        expGolomp.parseUE(sps,
                                        expGolomp.getNextByteOffset(),
                                        expGolomp.getNextBitOffset());
                        sequenceParameterSet.add("<max_dec_frame_buffering>" +
                                                        expGolomp.getValue());
                }
        }
}


void parsePPS(byte[] pps)
{
        pictureParameterSet.add("<picture_parameter_set>");

        // pic_parameter_set_id
        expGolomp.parseUE(pps, 1, 0);
        pictureParameterSet.add("<pic_parameter_set_id>" + expGolomp.getValue());

        // seq_parameter_set_id
        expGolomp.parseUE(pps,
                        expGolomp.getNextByteOffset(),
                        expGolomp.getNextBitOffset());
        pictureParameterSet.add("<seq_parameter_set_id>" + expGolomp.getValue());

        // entropy_coding_mode_flag
        expGolomp.jumpXBits(pps,
                        expGolomp.getNextByteOffset(),
                        expGolomp.getNextBitOffset(),
                        1);
        pictureParameterSet.add("<entropy_coding_mode_flag>" + expGolomp.getValue());

        // pic_order_present_flag
        expGolomp.jumpXBits(pps,
                        expGolomp.getNextByteOffset(),
                        expGolomp.getNextBitOffset(),
                        1);
        pictureParameterSet.add("<pic_order_present_flag>" + expGolomp.getValue());

        // num_slice_groups_minus1
        expGolomp.parseUE(pps,
                        expGolomp.getNextByteOffset(),
                        expGolomp.getNextBitOffset());
        pictureParameterSet.add("<num_slice_groups_minus1>" + expGolomp.getValue());
        int num_slice_groups_minus1 = expGolomp.getValue();

        if(num_slice_groups_minus1 > 0)
        {
                // slice_group_map_type
                expGolomp.parseUE(pps,
                                expGolomp.getNextByteOffset(),
                                expGolomp.getNextBitOffset());
                pictureParameterSet.add("<slice_group_map_type>" +
                                                        expGolomp.getValue());
                int slice_group_map_type = expGolomp.getValue();

                if(slice_group_map_type == 0)
                {
                        for(int i = 0; i <= num_slice_groups_minus1; i++ )
                        {
                                // run_length_minus1[i]
                                expGolomp.parseUE(pps,
                                                expGolomp.getNextByteOffset(),
                                                expGolomp.getNextBitOffset());
                                pictureParameterSet.add("<run_length_minus1["+i+"]>" +
                                                        expGolomp.getValue());
                        }
                }
                if(slice_group_map_type == 2)
                {
                        for(int i = 0; i <= num_slice_groups_minus1; i++ )
                        {
                                // top_left[i]
                                expGolomp.parseUE(pps,
                                                expGolomp.getNextByteOffset(),
                                                expGolomp.getNextBitOffset());
                                pictureParameterSet.add("<top_left["+i+"]>" +
                                                        expGolomp.getValue());
                                // bottom_right[i]
                                expGolomp.parseUE(pps,
                                                expGolomp.getNextByteOffset(),
```

```
                                        expGolomp.getNextBitOffset());
                        pictureParameterSet.add("<bottom_right["+i+"]>" +
                                                        expGolomp.getValue());
                }
        }
        if(slice_group_map_type == 3 ||
           slice_group_map_type == 4 ||
           slice_group_map_type == 5)
        {

                // slice_group_change_direction_flag
                expGolomp.jumpXBits(pps,
                                expGolomp.getNextByteOffset(),
                                expGolomp.getNextBitOffset(),
                                1);
                pictureParameterSet.add("<slice_group_change_direction_flag>" +
                                                        expGolomp.getValue());
                // slice_group_change_rate_minus1
                expGolomp.parseUE(pps,
                                expGolomp.getNextByteOffset(),
                                expGolomp.getNextBitOffset());
                pictureParameterSet.add("<slice_group_change_rate_minus1>" +
                                                        expGolomp.getValue());
        }
        if( slice_group_map_type == 6)
        {
                // pic_size_in_map_units_minus1
                expGolomp.parseUE(pps,
                                expGolomp.getNextByteOffset(),
                                expGolomp.getNextBitOffset());
                pictureParameterSet.add("<pic_size_in_map_units_minus1>" +
                                                        expGolomp.getValue());
                int pic_size_in_map_units_minus1 = expGolomp.getValue();

                for(int i = 0; i <= pic_size_in_map_units_minus1; i++ )
                {
                        // slice_group_id[i]
                        expGolomp.jumpXBits(pps,
                                        expGolomp.getNextByteOffset(),
                                        expGolomp.getNextBitOffset(),
                                        ExtendedMath.ceil(
                                           ExtendedMath.log2(
                                             (double)num_slice_groups_minus1 +
                                               1)));
                        pictureParameterSet.add("<slice_group_id["+i+"]>" +
                        expGolomp.getValue());
                }
        }
}

// num_ref_idx_l0_active_minus1
expGolomp.parseUE(pps,
                  expGolomp.getNextByteOffset(),
                  expGolomp.getNextBitOffset());
pictureParameterSet.add("<num_ref_idx_l0_active_minus1>" +
                                                        expGolomp.getValue());

// num_ref_idx_l1_active_minus1
expGolomp.parseUE(pps,
                  expGolomp.getNextByteOffset(),
                  expGolomp.getNextBitOffset());
pictureParameterSet.add("<num_ref_idx_l1_active_minus1>" +
                                                        expGolomp.getValue());

// weighted_pred_flag
expGolomp.jumpXBits(pps,
                    expGolomp.getNextByteOffset(),
                    expGolomp.getNextBitOffset(),
                    1);
pictureParameterSet.add("<weighted_pred_flag>" + expGolomp.getValue());

// weighted_bipred_idc
expGolomp.jumpXBits(pps,
                    expGolomp.getNextByteOffset(),
                    expGolomp.getNextBitOffset(),
                    2);
pictureParameterSet.add("<weighted_bipred_idc>" + expGolomp.getValue());

// pic_init_qp_minus26
expGolomp.parseSE(pps,
```

```
                        expGolomp.getNextByteOffset(),
                        expGolomp.getNextBitOffset());
        pictureParameterSet.add("<pic_init_qp_minus26>" + expGolomp.getValue());

        // pic_init_qs_minus26
        expGolomp.parseSE(pps,
                        expGolomp.getNextByteOffset(),
                        expGolomp.getNextBitOffset());
        pictureParameterSet.add("<pic_init_qs_minus26>" + expGolomp.getValue());

        // chroma_qp_index_offset
        expGolomp.parseSE(pps,
                        expGolomp.getNextByteOffset(),
                        expGolomp.getNextBitOffset());
        pictureParameterSet.add("<chroma_qp_index_offset>" + expGolomp.getValue());

        // deblocking_filter_control_present_flag
        expGolomp.jumpXBits(pps,
                          expGolomp.getNextByteOffset(),
                          expGolomp.getNextBitOffset(),
                          1);
        pictureParameterSet.add("<deblocking_filter_control_present_flag>" +
                                                              expGolomp.getValue());

        // constrained_intra_pred_flag
        expGolomp.jumpXBits(pps,
                          expGolomp.getNextByteOffset(),
                          expGolomp.getNextBitOffset(),
                          1);
        pictureParameterSet.add("<constrained_intra_pred_flag>" +
                                                              expGolomp.getValue());

        // redundant_pic_cnt_present_flag
        expGolomp.jumpXBits(pps,
                          expGolomp.getNextByteOffset(),
                          expGolomp.getNextBitOffset(),
                          1);
        pictureParameterSet.add("<redundant_pic_cnt_present_flag>" +
                                                              expGolomp.getValue());

        // TODO: if(more_rbsp_data) {...}
}


public void addParameter()
{
        // some parameters for H264 video
        boolean chroma_format_idc_present = false;
        for(int i = 0; i < sequenceParameterSet.size(); i++)
        {
                String param = sequenceParameterSet.get(i);
                if(param.contains("profile_idc") && param.contains("66"))
                        parameter.add(0, "Baseline");
                else if(param.contains("profile_idc") && param.contains("77"))
                        parameter.add(0, "Main");
                else if(param.contains("profile_idc") && param.contains("88"))
                        parameter.add(0, "Extended");
                if(param.contains("level_idc"))
                        parameter.add(1,
                                        param.substring(param.indexOf(">") + 1,
                                        param.length()));
                if(param.contains("log2_max_pic_order_cnt_lsb_minus4"))
                        parameter.add(2, (int)Math.pow(2,
                                 Double.parseDouble(param.substring(param.indexOf(">") +
                                 1, param.length())) + 4) / 2 + "");
                if(param.contains("num_ref_pics"))
                        parameter.add(3,
                                param.substring(param.indexOf(">") + 1, param.length()));
                if(param.contains("pic_width_in_mbs_minus1"))
                        parameter.add(4,
                                 ((Integer.parseInt(param.substring(param.indexOf(">") +
                                 1, param.length())) + 1) * 16) + "");
                if(param.contains("pic_height_in_map_units_minus1"))
                        parameter.add(5,
                                 ((Integer.parseInt(param.substring(param.indexOf(">") +
                                 1, param.length())) + 1) * 16) + "");
                if(param.contains("num_units_in_tick"))
                        parameter.add(6,
                                param.substring(param.indexOf(">") + 1, param.length()));
                else if(parameter.size() == 6)
```

```
                        parameter.add(6, "0");
                if(param.contains("time_scale"))
                        parameter.add(7,
                                param.substring(param.indexOf(">") + 1, param.length()));
                else if(parameter.size() == 7)
                        parameter.add(7, "0");
                if(param.contains("chroma_format_idc") &&
                    param.substring(param.indexOf(">") + 1, param.length()).equals("2"))
                {
                        parameter.add(8, "4:2:2");
                        chroma_format_idc_present = true;
                }
                if(param.contains("chroma_format_idc") &&
                    param.substring(param.indexOf(">") + 1, param.length()).equals("3"))
                {
                        parameter.add(8, "4:4:4");
                        chroma_format_idc_present = true;
                }
        }
        if(!chroma_format_idc_present)
        {
                parameter.add(8, "4:2:0");
                sequenceParameterSet.add("<chroma_format_idc>1");
        }


        for(int i = 0; i < pictureParameterSet.size(); i++)
        {
                String param = pictureParameterSet.get(i);
                if(param.contains("entropy_coding_mode_flag") && param.contains("0"))
                        parameter.add(9, "CAVLC");
                else if (param.contains("entropy_coding_mode_flag") &&
                                                                param.contains("1"))
                        parameter.add(9, "CABAC");
                if(param.contains("num_slice_groups_minus1"))
                        parameter.add(10,
                                (Integer.parseInt(param.substring(param.indexOf(">") + 1,
                                param.length())) + 1) + "");
                if(param.contains("pic_size_in_map_units_minus1"))
                        parameter.add(11, (Integer.parseInt(parameter.get(5)) *
                                (Integer.parseInt(param.substring(param.indexOf(">") + 1,
                                param.length())) + 1)) + "");
        }

        for(int i = 0; i < sequenceParameterSet.size(); i++)
                parameter.add(sequenceParameterSet.get(i));
        for(int i = 0; i < pictureParameterSet.size(); i++)
                parameter.add(pictureParameterSet.get(i));

        parameter.add(packetizationMode);
    }


    public void parsePacketizationMode(String sdp)
    {
        int index = sdp.indexOf("packetization-mode") + 19;
        packetizationMode = sdp.substring(index, index + 1);
    }
}
```

D.1.7 Die Schnittstelle DatagramParser

```
public interface DatagramParser
{
        public void parse(DatagramPacket packet);

        public ArrayList<long[]> getOutputParameter();

        public void setInputParameter(ArrayList<String> parameter);
}
```

D.1.8 Die Klasse H264Parser

```
public class H264Parser implements DatagramParser
{
	private int fragmentOffset;
	private byte[] fragmentedNAL;

	private ExpGolomb expGolomb;
	private CAVLC cavlc;
	private int headerSize;
	private int byteOffset;
	private int bitOffset;

	private ArrayList<SequenceParameterSet> sequenceParameterSets;
	private SequenceParameterSet sequenceParameterSet;
	private ArrayList<PictureParameterSet> pictureParameterSets;
	private PictureParameterSet pictureParameterSet;
	private int oldPictureParameterSetId;

	private ArrayList<String> inputParameter;
	private ArrayList<long[]> outputParameter;
	private long[] nalTypes;
	private long[] nalRefIdc;
	private long[] sliceTypes;
	private long[] packets;
	private long[] numSlicesPerFrame;
	private long[] macroblockTypes;
	private long[] subMacroblockTypes;
	private long[] refPics;

	// NAL-Header-Parameter
	private int nal_unit_type;
	private int nal_ref_idc;

	// Slice-Header-Parameter
	private int first_mb_in_slice;
	private int slice_type;
	private int pic_parameter_set_id;
	private int frame_num;
	private int field_pic_flag;
	private int bottom_field_flag;
	private int idr_pic_id;
	private int pic_order_cnt_lsb;
	private int delta_pic_order_cnt_bottom;
	private int delta_pic_order_cnt0;
	private int delta_pic_order_cnt1;
	private int redundant_pic_cnt;
	private int direct_spatial_mv_pred_flag;
	private int num_ref_idx_active_override_flag;
	private int num_ref_idx_l0_active_minus1;
	private int num_ref_idx_l1_active_minus1;
	private int ref_pic_list_reordering_flag_l0;
	private int ref_pic_list_reordering_flag_l1;
	private int reordering_of_pic_nums_idc;
	private int abs_diff_pic_num_minus1;
	private int long_term_pic_num;
	private int no_output_of_prior_pics_flag;
	private int long_term_reference_flag;
	private int adaptive_ref_pic_marking_mode_flag;
	private int memory_management_control_operation;
	private int difference_of_pic_nums_minus1;
	private int long_term_frame_idx;
	private int max_long_term_frame_idx_plus1;
	private int cabac_init_idc;
	private int slice_qp_delta;
	private int sp_for_switch_flag;
	private int slice_qs_delta;
	private int disable_deblocking_filter_idc;
	private int slice_alpha_c0_offset_div2;
	private int slice_beta_offset_div2;
	private int slice_group_change_cycle;

	// Slice-Data-Parameter
	private int mb_skip_run;
	private int mb_field_decoding_flag;

	// Macroblock-Parameter
```

```
private int mbaffFrameFlag;
private int currMbAddr;
private int mbWidthC;
private int mbHeightC;
private int mb_type;
private int[] sub_mb_types;
private int pcm_alignment_zero_bit;
private int[] pcm_sample_luma;
private int[] pcm_sample_chroma;
private int transform_size_8x8_flag;
private int prev_intra4x4_pred_mode_flag;
private int prev_intra8x8_pred_mode_flag;
private int coded_block_pattern;
private int codedBlockPatternLuma;
private int codedBlockPatternChroma;
private int mb_qp_delta;
private int trailing_ones_sign_flag;
private int[] pic_mb_types;
private int[][] totalCoeffLuma;
private int[][] totalCoeffChromaCb;
private int[][] totalCoeffChromaCr;
private int[] trailingOnes;

private final int ChromaDCLevel = 0;
private final int ChromaACLevel = 1;
private final int Intra16x16DCLevel = 2;
private final int Intra16x16ACLevel = 3;
private final int LumaLevel = 4;

// Macroblock-Types
private final int B_Direct_16x16 = 31;
private final int B_L0_16x16 = 32;
private final int B_L1_16x16 = 33;
private final int B_Bi_16x16 = 34;
private final int B_L0_L0_16x8 = 35;
private final int B_L0_L0_8x16 = 36;
private final int B_L1_L1_16x8 = 37;
private final int B_L1_L1_8x16 = 38;
private final int B_L0_L1_16x8 = 39;
private final int B_L0_L1_8x16 = 40;
private final int B_L1_L0_16x8 = 41;
private final int B_L1_L0_8x16 = 42;
private final int B_L0_Bi_16x8 = 43;
private final int B_L0_Bi_8x16 = 44;
private final int B_L1_Bi_16x8 = 45;
private final int B_L1_Bi_8x16 = 46;
private final int B_Bi_L0_16x8 = 47;
private final int B_Bi_L0_8x16 = 48;
private final int B_Bi_L1_16x8 = 49;
private final int B_Bi_L1_8x16 = 50;
private final int B_Bi_Bi_16x8 = 51;
private final int B_Bi_Bi_8x16 = 52;
private final int B_8x8 = 53;
private final int B_Skip = 54;

private final int P_L0_16x16 = 0;
private final int P_L0_L0_16x8 = 1;
private final int P_L0_L0_8x16 = 2;
private final int P_8x8 = 3;
private final int P_8x8ref0 = 4;
private final int P_Skip = 55;

private final int I_NxN = 5;
private final int I_16x16_0_0_0 = 6;
private final int I_16x16_1_0_0 = 7;
private final int I_16x16_2_0_0 = 8;
private final int I_16x16_3_0_0 = 9;
private final int I_16x16_0_1_0 = 10;
private final int I_16x16_1_1_0 = 11;
private final int I_16x16_2_1_0 = 12;
private final int I_16x16_3_1_0 = 13;
private final int I_16x16_0_2_0 = 14;
private final int I_16x16_1_2_0 = 15;
private final int I_16x16_2_2_0 = 16;
private final int I_16x16_3_2_0 = 17;
private final int I_16x16_0_0_1 = 18;
private final int I_16x16_1_0_1 = 19;
private final int I_16x16_2_0_1 = 20;
private final int I_16x16_3_0_1 = 21;
private final int I_16x16_0_1_1 = 22;
```

```
private final int I_16x16_1_1_1 = 23;
private final int I_16x16_2_1_1 = 24;
private final int I_16x16_3_1_1 = 25;
private final int I_16x16_0_2_1 = 26;
private final int I_16x16_1_2_1 = 27;
private final int I_16x16_2_2_1 = 28;
private final int I_16x16_3_2_1 = 29;
private final int I_PCM = 30;

// Sub-Macroblock-Types
private final int B_Direct_8x8 = 5;
private final int B_L0_8x8 = 6;
private final int B_L1_8x8 = 7;
private final int B_Bi_8x8 = 8;
private final int B_L0_8x4 = 9;
private final int B_L0_4x8 = 10;
private final int B_L1_8x4 = 11;
private final int B_L1_4x8 = 12;
private final int B_Bi_8x4 = 13;
private final int B_Bi_4x8 = 14;
private final int B_L0_4x4 = 15;
private final int B_L1_4x4 = 16;
private final int B_Bi_4x4 = 17;

private final int P_L0_8x8 = 0;
private final int P_L0_8x4 = 1;
private final int P_L0_4x8 = 2;
private final int P_L0_4x4 = 3;

// Macroblock-Prediction-Modes
private final int Direct = -2;
private final int BiPred = -1;
private final int Pred_L0 = 0;
private final int Pred_L1 = 1;
private final int Intra_4x4 = 2;
private final int Intra_8x8 = 3;
private final int Intra_16x16 = 4;



public H264Parser()
{
        expGolomb = new ExpGolomb();
        cavlc = new CAVLC();

        sequenceParameterSets = new ArrayList<SequenceParameterSet>();
        pictureParameterSets = new ArrayList<PictureParameterSet>();

        inputParameter = new ArrayList<String>();
        outputParameter = new ArrayList<long[]>();
        nalTypes = new long[32];
        nalRefIdc = new long[4];
        sliceTypes = new long[14];
        packets = new long[32];
        numSlicesPerFrame = new long[2];
        macroblockTypes = new long[56];
        subMacroblockTypes = new long[4];
        refPics = new long[2];

        outputParameter.add(nalTypes);
        outputParameter.add(nalRefIdc);
        outputParameter.add(sliceTypes);
        outputParameter.add(packets);
        outputParameter.add(numSlicesPerFrame);
        outputParameter.add(macroblockTypes);
        outputParameter.add(subMacroblockTypes);
        outputParameter.add(refPics);
}


@Override
public void parse(DatagramPacket packet)
{
        headerSize = parseRTPHeaderSize(packet.getData());
        int nalOrPacketType = parseNALType(packet.getData()[headerSize]);
        int nalOrPacketRefIdc = parseNALRefIdc(packet.getData()[headerSize]);
        packets[nalOrPacketType]++;

        switch(nalOrPacketType)
        {
```

```
// H264 NAL: Coded Slice
case 1:         nal_unit_type = nalOrPacketType;
                nal_ref_idc = nalOrPacketRefIdc;
                nalTypes[nal_unit_type]++;
                nalRefIdc[nal_ref_idc]++;
                parseSliceHeader(packet.getData(), headerSize + 1);
                // parseSliceData(packet.getData(), this.byteOffset,
                this.bitOffset);
                break;

// H264 NAL: Data Partitions - Extended Profile
case 2:         nal_unit_type = nalOrPacketType;
                nal_ref_idc = nalOrPacketRefIdc;
                nalTypes[nal_unit_type]++;
                nalRefIdc[nal_ref_idc]++;
                parseSliceHeader(packet.getData(), headerSize + 1);
                break;

case 3:
case 4:         nal_unit_type = nalOrPacketType;
                nal_ref_idc = nalOrPacketRefIdc;
                nalTypes[nal_unit_type]++;
                nalRefIdc[nal_ref_idc]++;
                // parseSliceData(packet.getData(), this.byteOffset,
                this.bitOffset);
                break;

// H264 NAL: IDR
case 5:         nal_unit_type = nalOrPacketType;
                nal_ref_idc = nalOrPacketRefIdc;
                nalTypes[nal_unit_type]++;
                nalRefIdc[nal_ref_idc]++;
                parseSliceHeader(packet.getData(), headerSize + 1);
                // parseSliceData(packet.getData(), this.byteOffset,
                this.bitOffset);
                break;

// H264 NAL: SEI
case 6:         nal_unit_type = nalOrPacketType;
                nal_ref_idc = nalOrPacketRefIdc;
                nalTypes[nal_unit_type]++;
                nalRefIdc[nal_ref_idc]++;
                break;

// H264 NAL: Parameter Sets - Transport over TCP
// case 7,8:

// H264 NAL: AUD
case 9:         nal_unit_type = nalOrPacketType;
                nal_ref_idc = nalOrPacketRefIdc;
                nalTypes[nal_unit_type]++;
                nalRefIdc[nal_ref_idc]++;
                break;

// H264 NAL: End of Sequence
case 10:        nal_unit_type = nalOrPacketType;
                nal_ref_idc = nalOrPacketRefIdc;
                nalTypes[nal_unit_type]++;
                nalRefIdc[nal_ref_idc]++;
                break;

// H264 NAL: End of Stream
case 11:        nal_unit_type = nalOrPacketType;
                nal_ref_idc = nalOrPacketRefIdc;
                nalTypes[nal_unit_type]++;
                nalRefIdc[nal_ref_idc]++;
                break;

// H264 NAL: Filler Data
case 12:        nal_unit_type = nalOrPacketType;
        nal_ref_idc = nalOrPacketRefIdc;
        nalTypes[nal_unit_type]++;
                        nalRefIdc[nal_ref_idc]++;
                        break;

// H264 NAL: Sequence Parameter Set Extension
case 13:        nal_unit_type = nalOrPacketType;
                nal_ref_idc = nalOrPacketRefIdc;
                nalTypes[nal_unit_type]++;
                nalRefIdc[nal_ref_idc]++;
```

```
                        break;

                // RTP STAP-A
                case 24:        parseSTAPA(packet);
                                break;

                // RTP STAP-B
                case 25:        parseSTAPB(packet);
                                break;

                // RTP MTAP-16
                case 26:        parseMTAP16(packet);
                                break;

                // RTP MTAP-24
                case 27:        parseMTAP24(packet);
                                break;

                // RTP FU-A
                case 28:        parseFUA(packet);
                                break;

                // RTP FU-B
                case 29:        parseFUB(packet);
                                break;
        }
}


@Override
public ArrayList<long[]>getOutputParameter()
{
        return outputParameter;
}


@Override
public void setInputParameter(ArrayList<String> parameter)
{
        inputParameter = parameter;
        SequenceParameterSet sps;
        PictureParameterSet pps;
        for(int i = 0; i< parameter.size(); i++)
        {
                if(parameter.get(i).equals("<sequence_parameter_set>"))
                {
                        sps = new SequenceParameterSet();
                        createSPS(sps, i);
                        sequenceParameterSets.add(sps);
                }
                if(parameter.get(i).equals("<picture_parameter_set>"))
                {
                        pps = new PictureParameterSet();
                        createPPS(pps, i);
                        pictureParameterSets.add(pps);
                }
        }
}


public void createSPS(SequenceParameterSet sps, int i)
{
sps.log2_max_frame_num_minus4 =
                Integer.parseInt(readParameter("log2_max_frame_num_minus4", i));
sps.frame_mbs_only_flag = Integer.parseInt(readParameter("frame_mbs_only_flag", i));
sps.pic_order_cnt_type = Integer.parseInt(readParameter("pic_order_cnt_type", i));
sps.log2_max_pic_order_cnt_lsb_minus4 =
                Integer.parseInt(readParameter("log2_max_pic_order_cnt_lsb_minus4", i));
sps.delta_pic_order_always_zero_flag =
                Integer.parseInt(readParameter("delta_pic_order_always_zero_flag", i));
sps.mb_adaptive_frame_field_flag =
                Integer.parseInt(readParameter("mb_adaptive_frame_field_flag", i));
sps.chroma_format_idc = Integer.parseInt(readParameter("chroma_format_idc", i));
sps.direct_8x8_inference_flag =
                Integer.parseInt(readParameter("direct_8x8_inference_flag", i));
sps.pic_width_in_mbs_minus1 =
                Integer.parseInt(readParameter("pic_width_in_mbs_minus1", i));
sps.pic_height_in_map_units_minus1 =
                Integer.parseInt(readParameter("pic_height_in_map_units_minus1", i));
}
```

```
public void createPPS(PictureParameterSet pps, int i)
{
        pps.picture_parameter_set_id =
                Integer.parseInt(readParameter("picture_parameter_set_id", i));
        pps.sequence_parameter_set_id =
                Integer.parseInt(readParameter("sequence_parameter_set_id", i));
        pps.pic_order_present_flag =
                Integer.parseInt(readParameter("pic_order_present_flag", i));
        pps.redundant_pic_cnt_present_flag =
                Integer.parseInt(readParameter("redundant_pic_cnt_present_flag", i));
        pps.entropy_coding_mode_flag =
                Integer.parseInt(readParameter("entropy_coding_mode_flag", i));
        pps.deblocking_filter_control_present_flag =
          Integer.parseInt(readParameter("deblocking_filter_control_present_flag", i));
        pps.num_slice_groups_minus1 =
                Integer.parseInt(readParameter("num_slice_groups_minus1", i));
        pps.slice_group_map_type =
                Integer.parseInt(readParameter("slice_group_map_type", i));
        pps.pic_size_in_map_units_minus1 =
                Integer.parseInt(readParameter("pic_size_in_map_units_minus1", i));
        pps.slice_group_change_rate_minus1 =
                Integer.parseInt(readParameter("slice_group_change_rate_minus1", i));
        pps.transform_8x8_mode_flag =
                Integer.parseInt(readParameter("transform_8x8_mode_flag", i));
}


public void setParameterSets(int picture_parameter_set_id)
{
        int sequence_parameter_set_id = -1;
        for(PictureParameterSet pps : pictureParameterSets)
                if(pps.picture_parameter_set_id == picture_parameter_set_id)
                {
                        pictureParameterSet = pps;
                        sequence_parameter_set_id = pps.sequence_parameter_set_id;
                }
        for(SequenceParameterSet sps: sequenceParameterSets)
                if(sps.sequence_parameter_set_id == sequence_parameter_set_id)
                        sequenceParameterSet = sps;
        setMbWidthHeight();

        pcm_sample_luma = new int[256];
        pcm_sample_chroma = new int[64];
        sub_mb_types = new int[4];
        totalCoeffLuma = new int[(sequenceParameterSet.pic_width_in_mbs_minus1 + 1) *
                        (sequenceParameterSet.pic_height_in_map_units_minus1 + 1)][16];
        totalCoeffChromaCb = new int[(sequenceParameterSet.pic_width_in_mbs_minus1 + 1)
                        * (sequenceParameterSet.pic_height_in_map_units_minus1 + 1)][4];
        totalCoeffChromaCr = new int[(sequenceParameterSet.pic_width_in_mbs_minus1 + 1)
                        * (sequenceParameterSet.pic_height_in_map_units_minus1 + 1)][4];
        trailingOnes = new int[(sequenceParameterSet.pic_width_in_mbs_minus1 + 1) *
                              (sequenceParameterSet.pic_height_in_map_units_minus1 + 1)];
        pic_mb_types = new int[(sequenceParameterSet.pic_width_in_mbs_minus1 + 1) *
                              (sequenceParameterSet.pic_height_in_map_units_minus1 + 1)];
}


public int parseRTPHeaderSize(byte[] packet)
{
        int crsc = (packet[0] & 0x0f);
        int x = (packet[0] & 0x10);
        if(x != 0)
        {
                int fixed = 12 + 4 * crsc;
                int var = (((packet[fixed + 2] & 0xff) << 8) |
                              (packet[fixed + 3] & 0xff));
                return fixed + var;
        }
        return 12 + 4 * crsc;
}


public int parseNALRefIdc(byte nalHeader)
{
        return ((nalHeader & 0x60) >> 5);
}
```

```
public int parseNALType(byte nalHeader)
{
        return (nalHeader & 0x1f);
}


public void parseSTAPA(DatagramPacket packet)
{
        // RTP-Header + 1 Byte STAP-A-Header
        int size, offset = headerSize + 1;
        byte[] rtpPacket = packet.getData();

        while(offset < packet.getLength())
        {
                nalTypes[24]++;
                // 2 Byte NAL-Size
                size = (((rtpPacket[offset] & 0xff) << 8) |
                          (rtpPacket[offset + 1] & 0xff));
                // 1 Byte NAL-Header
                nal_unit_type = parseNALType(rtpPacket[offset + 2]);
                nalTypes[nal_unit_type]++;
                nal_ref_idc = parseNALRefIdc(rtpPacket[offset + 2]);
                nalRefIdc[nal_ref_idc]++;
                if(nal_unit_type == 1 || nal_unit_type == 5)
                {
                        parseSliceHeader(rtpPacket, offset + 3);
                        // parseSliceData(packet.getData(), this.byteOffset,
                        this.bitOffset);
                }
                // Next NAL-Unit
                offset += (size + 2);
        }
}


public void parseSTAPB(DatagramPacket packet)
{
        // RTP-Header + 1 Byte STAP-B-Header + 2Byte DON
        int size, offset = headerSize + 3;
        byte[] rtpPacket = packet.getData();

        while(offset < packet.getLength())
        {
                nalTypes[25]++;
                // 2 Byte NAL-Size
                size = (((rtpPacket[offset] & 0xff) << 8) |
                          (rtpPacket[offset + 1] & 0xff));
                // 1 Byte NAL-Header
                nal_unit_type = parseNALType(rtpPacket[offset + 2]);
                nalTypes[nal_unit_type]++;
                nal_ref_idc = parseNALRefIdc(rtpPacket[offset + 2]);
                nalRefIdc[nal_ref_idc]++;
                if(nal_unit_type == 1 || nal_unit_type == 5)
                {
                        parseSliceHeader(rtpPacket, offset + 3);
                        // parseSliceData(packet.getData(), this.byteOffset,
                        this.bitOffset);
                }
                // Next NAL-Unit
                offset += (size + 2);
        }
}


public void parseMTAP16(DatagramPacket packet)
{
        // RTP-Header + 1 Byte MTAP16-Header + 2Byte DON
        int size, offset = headerSize + 3;
        byte[] rtpPacket = packet.getData();

        while(offset < packet.getLength())
        {
                nalTypes[26]++;
                // 2 Byte NAL-Size
                size = (((rtpPacket[offset] & 0xff) << 8) |
                          (rtpPacket[offset + 1] & 0xff));
                // 1 Byte NAL-Header
                nal_unit_type = parseNALType(rtpPacket[offset + 5]);
                nalTypes[nal_unit_type]++;
                nal_ref_idc = parseNALRefIdc(rtpPacket[offset + 5]);
```

```
                        nalRefIdc[nal_ref_idc]++;
                        if(nal_unit_type == 1 || nal_unit_type == 5)
                        {
                                parseSliceHeader(rtpPacket, offset + 6);
                                // parseSliceData(packet.getData(), this.byteOffset,
                                this.bitOffset);
                        }
                        // Next NAL-Unit
                        offset += (size + 5);
                }
        }


        public void parseMTAP24(DatagramPacket packet)
        {
                // RTP-Header + 1 Byte MTAP24-Header + 2Byte DON
                int size, offset = headerSize + 3;
                byte[] rtpPacket = packet.getData();

                while(offset < packet.getLength())
                {
                        nalTypes[27]++;
                        // 2 Byte NAL-Size
                        size = (((rtpPacket[offset] & 0xff) << 8) |
                                (rtpPacket[offset + 1] & 0xff));
                        // 1 Byte NAL-Header
                        nal_unit_type = parseNALType(rtpPacket[offset + 6]);
                        nalTypes[nal_unit_type]++;
                        nal_ref_idc = parseNALRefIdc(rtpPacket[offset + 6]);
                        nalRefIdc[nal_ref_idc]++;
                        if(nal_unit_type == 1 || nal_unit_type == 5)
                        {
                                parseSliceHeader(rtpPacket, offset + 7);
                                // parseSliceData(packet.getData(), this.byteOffset,
                                this.bitOffset);
                        }
                        // Next NAL-Unit
                        offset += (size + 6);
                }
        }


        public void parseFUA(DatagramPacket packet)
        {
                byte[] rtpPacket = packet.getData();

                // Start-Flag im FU-A-Header
                if((rtpPacket[headerSize + 1] & 0x80) == 128)
                {
                        nalTypes[28]++;
                        nal_unit_type = parseNALType(rtpPacket[headerSize + 1]);
                        nalTypes[nal_unit_type]++;
                        nal_ref_idc = parseNALRefIdc(rtpPacket[headerSize]);
                        nalRefIdc[nal_ref_idc]++;

                        parseSliceHeader(rtpPacket, headerSize + 2);
                }
//              byte[] defragmentedNAL = defragmentNAL(packet);
//              if(defragmentedNAL != null)
//              {
//                      nal_unit_type = parseNALType(defragmentedNAL[headerSize]);
//                      if(nal_unit_type == 1 || nal_unit_type == 5)
//                      {
//                              parseSliceHeader(defragmentedNAL, headerSize + 1);
//                              parseSliceData(defragmentedNAL, this.byteOffset ,
                                this.bitOffset);
//                      }
//              }
        }


        public void parseFUB(DatagramPacket packet)
        {
                byte[] rtpPacket = packet.getData();

                // Start-Flag im FU-B-Header
                if((rtpPacket[headerSize + 1] & 0x80) == 128)
                {
                        nalTypes[29]++;
                        nal_unit_type = parseNALType(rtpPacket[headerSize + 1]);
```

```
                        nalTypes[nal_unit_type]++;
                        nal_ref_idc = parseNALRefIdc(rtpPacket[headerSize]);
                        nalRefIdc[nal_ref_idc]++;

                        parseSliceHeader(rtpPacket, headerSize + 4);
                }
//              byte[] defragmentedNAL = defragmentNAL(packet);
//              if(defragmentedNAL != null)
//              {
//                      nal_unit_type = parseNALType(defragmentedNAL[headerSize]);
//                      if(nal_unit_type == 1 || nal_unit_type == 5)
//                      {
//                              parseSliceHeader(defragmentedNAL, headerSize + 1);
//                              parseSliceData(defragmentedNAL, this.byteOffset ,
                                this.bitOffset);
//                      }
//              }
        }


        public byte[] defragmentNAL(DatagramPacket rtpPacket)
        {
                byte[] fragment = rtpPacket.getData();
                // if FU-A Start-Flag == 1
                if((rtpPacket.getData()[headerSize + 1] & 0x80) == 128)
                {
                        fragmentOffset = 0;
                        fragmentedNAL = new byte[MAX_NAL_SIZE];
                }

                if (fragmentedNAL != null)
                {
                        // First Fragment + RTP-Header
                        if(fragmentOffset == 0)
                        {
                                for(int j = 0; j < rtpPacket.getLength() - 1; j++)
                                {
                                        // RTP-Header
                                        if(j < headerSize)
                                                fragmentedNAL[j] = fragment[j];
                                        // NAL-Header
                                        if(j == headerSize)
                                                fragmentedNAL[12] = (byte)
                                                                ((fragment[headerSize] & 0xe0) |
                                                                (fragment[headerSize + 1] & 0x1f));
                                        // NAL-Fragment
                                        if(j > headerSize)
                                                fragmentedNAL[j] = fragment[j + 1];
                                }
                                fragmentOffset += (rtpPacket.getLength() - 1);
                        }
                        // Other Fragments
                        else if (fragmentOffset + rtpPacket.getLength() < MAX_NAL_SIZE)
                        {
                                // NAL-Fragment
                                for(int j = 0; j < rtpPacket.getLength() - (headerSize+2); j++)
                                        fragmentedNAL[j + fragmentOffset] =
                                                                        fragment[j + headerSize + 2];
                                fragmentOffset += (rtpPacket.getLength() - (headerSize + 2));
                        }
                }
                // if FU-A End-Flag == 1
                if((rtpPacket.getData()[headerSize + 1] & 0x40) == 64)
                {
                        return fragmentedNAL;
                }
                return null;
        }


        public String readParameter(String parameter, int begin)
        {
                int end = 0;;
                for(int i = begin; i < inputParameter.size(); i++)
                        if(inputParameter.get(i).contains("parameter_set>") || i ==
                                                                        inputParameter.size() - 1)
                                end = i;
                for(int i = begin; i < end; i++)
                if(inputParameter.get(i).contains(parameter))
                        return
```

```
                inputParameter.get(i).substring(inputParameter.get(i).indexOf(">") + 1,
                inputParameter.get(i).length());
return "0";
}


public void parseSliceHeader(byte[] rtpPacket, int byteOffset)
{
        expGolomb.parseUE(rtpPacket, byteOffset, 0);
        first_mb_in_slice = expGolomb.getValue();
        if(first_mb_in_slice == 0)
                numSlicesPerFrame[0]++;
        numSlicesPerFrame[1]++;

        expGolomb.parseUE(rtpPacket,
                          expGolomb.getNextByteOffset(),
                          expGolomb.getNextBitOffset());
        slice_type = expGolomb.getValue();

        oldPictureParameterSetId = pic_parameter_set_id;
        expGolomb.parseUE(rtpPacket,
                          expGolomb.getNextByteOffset(),
                          expGolomb.getNextBitOffset());
        pic_parameter_set_id = expGolomb.getValue();
        if(oldPictureParameterSetId != pic_parameter_set_id ||
           pictureParameterSet == null)
                setParameterSets(pic_parameter_set_id);

        expGolomb.jumpXBits(rtpPacket,
                            expGolomb.getNextByteOffset(),
                            expGolomb.getNextBitOffset(),
                            sequenceParameterSet.log2_max_frame_num_minus4 + 4);
        frame_num = expGolomb.getValue();

        if(sequenceParameterSet.frame_mbs_only_flag == 0)
        {
                expGolomb.jumpXBits(rtpPacket,
                                    expGolomb.getNextByteOffset(),
                                    expGolomb.getNextBitOffset(),
                                    1);
                field_pic_flag = expGolomb.getValue();

                if(field_pic_flag == 1)
                {
                        expGolomb.jumpXBits(rtpPacket,
                                            expGolomb.getNextByteOffset(),
                                            expGolomb.getNextBitOffset(),
                                            1);
                        bottom_field_flag = expGolomb.getValue();
                }
        }

        if(nal_unit_type == 5)
        {
                expGolomb.parseUE(rtpPacket,
                                  expGolomb.getNextByteOffset(),
                                  expGolomb.getNextBitOffset());
                idr_pic_id = expGolomb.getValue();
        }

        if(sequenceParameterSet.pic_order_cnt_type == 0)
        {
                expGolomb.jumpXBits(rtpPacket,
                                    expGolomb.getNextByteOffset(),
                                    expGolomb.getNextBitOffset(),
                                sequenceParameterSet.log2_max_pic_order_cnt_lsb_minus4 + 4);
                pic_order_cnt_lsb = expGolomb.getValue();

                if(pictureParameterSet.pic_order_present_flag == 1 &&
                   field_pic_flag == 0)
                {
                        expGolomb.parseSE(rtpPacket,
                                          expGolomb.getNextByteOffset(),
                                          expGolomb.getNextBitOffset());
                        delta_pic_order_cnt_bottom = expGolomb.getValue();
                }
        }

        if(sequenceParameterSet.pic_order_cnt_type == 1 &&
           sequenceParameterSet.delta_pic_order_always_zero_flag == 0)
```

```
{
        expGolomb.parseSE(rtpPacket,
                          expGolomb.getNextByteOffset(),
                          expGolomb.getNextBitOffset());
        delta_pic_order_cnt0 = expGolomb.getValue();

        if(pictureParameterSet.pic_order_present_flag == 1 &&
           field_pic_flag == 0)
        {
                expGolomb.parseSE(rtpPacket,
                                  expGolomb.getNextByteOffset(),
                                  expGolomb.getNextBitOffset());
                delta_pic_order_cnt1 = expGolomb.getValue();
        }
}

if(pictureParameterSet.redundant_pic_cnt_present_flag == 1)
{
        expGolomb.parseUE(rtpPacket,
                          expGolomb.getNextByteOffset(),
                          expGolomb.getNextBitOffset());
        redundant_pic_cnt = expGolomb.getValue();
}

if(slice_type == 1 || slice_type == 6)
{
        expGolomb.jumpXBits(rtpPacket,
                            expGolomb.getNextByteOffset(),
                            expGolomb.getNextBitOffset(),
                            1);
        direct_spatial_mv_pred_flag = expGolomb.getValue();
}

if(slice_type == 0 || slice_type == 5 ||
   slice_type == 1 || slice_type == 6 ||
   slice_type == 3 || slice_type == 8)
{
        expGolomb.jumpXBits(rtpPacket,
                            expGolomb.getNextByteOffset(),
                            expGolomb.getNextBitOffset(),
                            1);
        num_ref_idx_active_override_flag = expGolomb.getValue();

        if(num_ref_idx_active_override_flag == 1)
        {
                expGolomb.parseUE(rtpPacket,
                                  expGolomb.getNextByteOffset(),
                                  expGolomb.getNextBitOffset());
                num_ref_idx_l0_active_minus1 = expGolomb.getValue();

                if(slice_type == 1 || slice_type == 6)
                {
                        expGolomb.parseUE(rtpPacket,
                                          expGolomb.getNextByteOffset(),
                                          expGolomb.getNextBitOffset());
                        num_ref_idx_l1_active_minus1 = expGolomb.getValue();
                }
        }
}

ref_pic_list_reordering(rtpPacket);

// TODO: weighted prediction options

if(nal_ref_idc != 0)
        dec_ref_pic_marking(rtpPacket);

if(pictureParameterSet.entropy_coding_mode_flag == 1 &&
   slice_type != 2 && slice_type != 7)
{
        expGolomb.parseUE(rtpPacket,
                          expGolomb.getNextByteOffset(),
                          expGolomb.getNextBitOffset());
        cabac_init_idc = expGolomb.getValue();
}

expGolomb.parseSE(rtpPacket,
                  expGolomb.getNextByteOffset(),
                  expGolomb.getNextBitOffset());
slice_qp_delta = expGolomb.getValue();
```

```
        if(slice_type == 3 || slice_type == 4 || slice_type == 8 || slice_type == 9)
        {
                if(slice_type == 3 || slice_type == 4)
                {
                        expGolomb.jumpXBits(rtpPacket,
                                            expGolomb.getNextByteOffset(),
                                            expGolomb.getNextBitOffset(),
                                            1);
                        sp_for_switch_flag = expGolomb.getValue();
                }
                expGolomb.parseSE(rtpPacket,
                                  expGolomb.getNextByteOffset(),
                                  expGolomb.getNextBitOffset());
                slice_qs_delta = expGolomb.getValue();
        }

        if(pictureParameterSet.deblocking_filter_control_present_flag == 1)
        {
        expGolomb.parseUE(rtpPacket,
                          expGolomb.getNextByteOffset(),
                          expGolomb.getNextBitOffset());
        disable_deblocking_filter_idc = expGolomb.getValue();

        if(disable_deblocking_filter_idc != 1)
        {
                expGolomb.parseSE(rtpPacket,
                                  expGolomb.getNextByteOffset(),
                                  expGolomb.getNextBitOffset());
                slice_alpha_c0_offset_div2 = expGolomb.getValue();

                expGolomb.parseSE(rtpPacket,
                                  expGolomb.getNextByteOffset(),
                                  expGolomb.getNextBitOffset());
                slice_beta_offset_div2 = expGolomb.getValue();
        }
        }

        if(pictureParameterSet.num_slice_groups_minus1 > 0 &&
           pictureParameterSet.slice_group_map_type >= 3 &&
           pictureParameterSet.slice_group_map_type <= 5)
        {
                expGolomb.jumpXBits(rtpPacket,
                                    expGolomb.getNextByteOffset(),
                                    expGolomb.getNextBitOffset(),
                                    ExtendedMath.ceil(
                                    ExtendedMath.log2(
                                    (pictureParameterSet.pic_size_in_map_units_minus1+1)
                                   / (pictureParameterSet.slice_group_change_rate_minus1
                                   + 1) + 1)));
                slice_group_change_cycle = expGolomb.getValue();
        }

        sliceTypes[slice_type]++;

        this.byteOffset = expGolomb.getNextByteOffset();
        this.bitOffset = expGolomb.getNextBitOffset();
}


public void ref_pic_list_reordering(byte[] rtpPacket)
{
        refPics[0]++;

        if(slice_type != 2 && slice_type != 7 && slice_type != 4 && slice_type != 9)
        {
                expGolomb.jumpXBits(rtpPacket,
                                    expGolomb.getNextByteOffset(),
                                    expGolomb.getNextBitOffset(),
                                    1);
                ref_pic_list_reordering_flag_l0 = expGolomb.getValue();

                if(ref_pic_list_reordering_flag_l0 == 1)
                {
                        do
                        {
                                expGolomb.parseUE(rtpPacket,
                                                  expGolomb.getNextByteOffset(),
                                                  expGolomb.getNextBitOffset());
                                reordering_of_pic_nums_idc = expGolomb.getValue();
```

```
                        if(reordering_of_pic_nums_idc == 0 ||
                           reordering_of_pic_nums_idc == 1)
                        {
                                expGolomb.parseUE(rtpPacket,
                                                expGolomb.getNextByteOffset(),
                                                expGolomb.getNextBitOffset());
                                abs_diff_pic_num_minus1 = expGolomb.getValue();
                        }
                        else if(reordering_of_pic_nums_idc == 2)
                        {
                                expGolomb.parseUE(rtpPacket,
                                                expGolomb.getNextByteOffset(),
                                                expGolomb.getNextBitOffset());
                                long_term_pic_num = expGolomb.getValue();

                                refPics[0]--;
                                refPics[1]++;
                        }
                }
                while(reordering_of_pic_nums_idc != 3);
        }
    }

    if(slice_type == 1 || slice_type == 6)
    {
            expGolomb.jumpXBits(rtpPacket,
                            expGolomb.getNextByteOffset(),
                            expGolomb.getNextBitOffset(), 1);
            ref_pic_list_reordering_flag_l1 = expGolomb.getValue();

            if(ref_pic_list_reordering_flag_l1 == 1)
            {
                    do
                    {
                            expGolomb.parseUE(rtpPacket,
                                            expGolomb.getNextByteOffset(),
                                            expGolomb.getNextBitOffset());
                            reordering_of_pic_nums_idc = expGolomb.getValue();

                            if(reordering_of_pic_nums_idc == 0 ||
                               reordering_of_pic_nums_idc == 1)
                            {
                                    expGolomb.parseUE(rtpPacket,
                                                    expGolomb.getNextByteOffset(),
                                                    expGolomb.getNextBitOffset());
                                    abs_diff_pic_num_minus1 = expGolomb.getValue();
                            }
                            else if(reordering_of_pic_nums_idc == 2)
                            {
                                    expGolomb.parseUE(rtpPacket,
                                                    expGolomb.getNextByteOffset(),
                                                    expGolomb.getNextBitOffset());
                                    long_term_pic_num = expGolomb.getValue();

                                    refPics[0]--;
                                    refPics[1]++;
                            }
                    }
                    while(reordering_of_pic_nums_idc != 3);
            }
    }
}


public void dec_ref_pic_marking(byte[] rtpPacket)
{
    if(nal_unit_type == 5)
    {
            expGolomb.jumpXBits(rtpPacket,
                            expGolomb.getNextByteOffset(),
                            expGolomb.getNextBitOffset(),
                            1);
            no_output_of_prior_pics_flag = expGolomb.getValue();

            expGolomb.jumpXBits(rtpPacket,
                            expGolomb.getNextByteOffset(),
                            expGolomb.getNextBitOffset(),
                            1);
            long_term_reference_flag = expGolomb.getValue();
```

```
        }
        else
        {
                expGolomb.jumpXBits(rtpPacket,
                                    expGolomb.getNextByteOffset(),
                                    expGolomb.getNextBitOffset(),
                                    1);
                adaptive_ref_pic_marking_mode_flag = expGolomb.getValue();

                if(adaptive_ref_pic_marking_mode_flag == 1)
                {
                        do
                        {
                                expGolomb.parseUE(rtpPacket,
                                                  expGolomb.getNextByteOffset(),
                                                  expGolomb.getNextBitOffset());
                                memory_management_control_operation =
                                                                    expGolomb.getValue();

                                if(memory_management_control_operation == 1 ||
                                   memory_management_control_operation == 3)
                                {
                                        expGolomb.parseUE(rtpPacket,
                                                          expGolomb.getNextByteOffset(),
                                                          expGolomb.getNextBitOffset());
                                        difference_of_pic_nums_minus1 =
                                                                    expGolomb.getValue();
                                }
                                if(memory_management_control_operation == 2)
                                {
                                        expGolomb.parseUE(rtpPacket,
                                                          expGolomb.getNextByteOffset(),
                                                          expGolomb.getNextBitOffset());
                                        long_term_pic_num = expGolomb.getValue();
                                }
                                if(memory_management_control_operation == 3 ||
                                   memory_management_control_operation == 6)
                                {
                                        expGolomb.parseUE(rtpPacket,
                                                          expGolomb.getNextByteOffset(),
                                                          expGolomb.getNextBitOffset());
                                        long_term_frame_idx = expGolomb.getValue();
                                }
                                if(memory_management_control_operation == 4)
                                {
                                        expGolomb.parseUE(rtpPacket,
                                                          expGolomb.getNextByteOffset(),
                                                          expGolomb.getNextBitOffset());
                                        max_long_term_frame_idx_plus1 =
                                                                    expGolomb.getValue();
                                }
                        } while(memory_management_control_operation != 0);
                }
        }
}


public void parseSliceData(byte[] rtpPacket, int byteOffset, int bitOffset)
{
        boolean moreDataFlag = true;
        int prevMbSkipped = 0;

        // TODO: if(entropy_coding_mode_flag)

        mbaffFrameFlag = sequenceParameterSet.mb_adaptive_frame_field_flag &
                         ((~field_pic_flag) & 0x01);
        currMbAddr = first_mb_in_slice * (1 + mbaffFrameFlag);

        do
        {
                if(slice_type != 2 && slice_type != 7 &&
                   slice_type != 4 && slice_type != 9)
                {
                        if(pictureParameterSet.entropy_coding_mode_flag == 0)
                        {
                                expGolomb.parseUE(rtpPacket,
                                                  this.byteOffset,
                                                  this.bitOffset);
                                mb_skip_run = expGolomb.getValue();
                                this.byteOffset = expGolomb.getNextByteOffset();
```

```
                                this.bitOffset = expGolomb.getNextBitOffset();

                                prevMbSkipped = mb_skip_run;
                                for(int i = 0; i < mb_skip_run; i++ )
                                {
                                        if(slice_type == 0 || slice_type == 5)
                                        {
                                                pic_mb_types[currMbAddr] = P_Skip;
                                                macroblockTypes[P_Skip]++;
                                        }
                                        if(slice_type == 1 || slice_type == 6)
                                        {
                                                pic_mb_types[currMbAddr] = B_Skip;
                                                macroblockTypes[B_Skip]++;
                                        }
                                        currMbAddr = nextMbAddress(currMbAddr);

                                }
                                moreDataFlag = more_rbsp_data(rtpPacket,
                                                              this.byteOffset,
                                                              this.bitOffset);
                        }
                        // TODO: else {...}
                }

                if(moreDataFlag)
                {
                        if(mbaffFrameFlag == 1 && (currMbAddr % 2 == 0 ||
                          (currMbAddr % 2 == 1 && prevMbSkipped > 0 )))
                        {
                                expGolomb.jumpXBits(rtpPacket,
                                                    this.byteOffset,
                                                    this.bitOffset,
                                                    1);
                                mb_field_decoding_flag = expGolomb.getValue();
                                this.byteOffset = expGolomb.getNextByteOffset();
                                this.bitOffset = expGolomb.getNextBitOffset();
                        }
                        macroblock_layer(rtpPacket);
                }

                if(pictureParameterSet.entropy_coding_mode_flag == 0)
                        moreDataFlag = more_rbsp_data(rtpPacket,
                                                      this.byteOffset,
                                                      this.bitOffset);
                // TODO: else {...}

                currMbAddr = nextMbAddress(currMbAddr);
        }
        while(moreDataFlag);
}


public void macroblock_layer(byte[] rtpPacket)
{
        expGolomb.parseUE(rtpPacket, this.byteOffset, this.bitOffset);
        mb_type = expGolomb.getValue();
        this.byteOffset = expGolomb.getNextByteOffset();
        this.bitOffset = expGolomb.getNextBitOffset();

        if(slice_type == 2 || slice_type == 7) // I Slice
        {
                mb_type += 5;
        }

        if(slice_type == 1 || slice_type == 6) // B Slice
        {
                if(mb_type >= 23)
                        mb_type -= 18;
                else
                        mb_type += 31;
        }

        // needed to avoid fault state
        mb_type = Math.abs(mb_type);
        mb_type %= 56;

        pic_mb_types[currMbAddr] = mb_type;
        macroblockTypes[mb_type]++;
```

```
if(mb_type == I_PCM)
{
        while(cavlc.getNextBitOffset() != 0)
        {
                expGolomb.jumpXBits(rtpPacket,
                                    this.byteOffset,
                                    this.bitOffset,
                                    1);
                pcm_alignment_zero_bit = cavlc.getValue();
                this.byteOffset = expGolomb.getNextByteOffset();
                this.bitOffset = expGolomb.getNextBitOffset();
        }
        for(int i = 0; i < 256; i++ )
        {
                expGolomb.jumpXBits(rtpPacket,
                                    this.byteOffset,
                                    this.bitOffset,
                                    8);
                pcm_sample_luma[i] = expGolomb.getValue();
                this.byteOffset = expGolomb.getNextByteOffset();
                this.bitOffset = expGolomb.getNextBitOffset();
        }
        for(int i = 0; i < 2 * mbWidthC * mbHeightC; i++ )
        {
                expGolomb.jumpXBits(rtpPacket,
                                    this.byteOffset,
                                    this.bitOffset,
                                    8);
                pcm_sample_chroma[i] = expGolomb.getValue();
                this.byteOffset = expGolomb.getNextByteOffset();
                this.bitOffset = expGolomb.getNextBitOffset();
        }
}
else
{
        int noSubMbPartSizeLessThan8x8Flag = 1;
        if(mb_type != I_NxN && mbPartPredMode(mb_type, 0) != Intra_16x16 &&
           numMbPart(mb_type) == 4)
        {
                sub_mb_pred(rtpPacket, mb_type);
                for(int mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)
                {
                        if(sub_mb_types[mbPartIdx] != B_Direct_8x8 )
                                if(numSubMbPart(sub_mb_types[mbPartIdx]) > 1)
                                        noSubMbPartSizeLessThan8x8Flag = 0;
                        else
                                if(sequenceParameterSet.direct_8x8_inference_flag
                                                                                == 0)
                                        noSubMbPartSizeLessThan8x8Flag = 0;
                }
        }
        else
        {
                if(pictureParameterSet.transform_8x8_mode_flag == 1 &&
                   mb_type == I_NxN)
                {
                        expGolomb.jumpXBits(rtpPacket,
                                            this.byteOffset,
                                            this.bitOffset,
                                            1);
                transform_size_8x8_flag = expGolomb.getValue();
                this.byteOffset = expGolomb.getNextByteOffset();
                this.bitOffset = expGolomb.getNextBitOffset();
                }
                mb_pred(rtpPacket);
        }

        if(mbPartPredMode(mb_type, 0) != Intra_16x16)
        {
                expGolomb.parseME(rtpPacket,
                                  this.byteOffset,
                                  this.bitOffset,
                                  mb_type);
                coded_block_pattern = expGolomb.getValue();
                this.byteOffset = expGolomb.getNextByteOffset();
                this.bitOffset = expGolomb.getNextBitOffset();
                codedBlockPatternLuma = coded_block_pattern % 16;
                codedBlockPatternChroma = coded_block_pattern / 16;
                if(codedBlockPatternLuma > 0 &&
                   pictureParameterSet.transform_8x8_mode_flag == 1 &&
```

```
                        mb_type != I_NxN &&
                        noSubMbPartSizeLessThan8x8Flag == 1 &&
                        (mb_type != B_Direct_16x16 ||
                         sequenceParameterSet.direct_8x8_inference_flag == 1))
                    {
                        // transform_size_8x8_flag
                        expGolomb.jumpXBits(rtpPacket,
                                            this.byteOffset,
                                            this.bitOffset,
                                            1);
                        this.byteOffset = expGolomb.getNextByteOffset();
                        this.bitOffset = expGolomb.getNextBitOffset();
                    }
                }
                else
                {
                        codedBlockPatternLuma = codedBlockPatternLuma(mb_type);
                        codedBlockPatternChroma = codedBlockPatternChroma(mb_type);
                }
        }

        if(codedBlockPatternLuma > 0 || codedBlockPatternChroma > 0 ||
           mbPartPredMode(mb_type, 0) == Intra_16x16)
        {
        expGolomb.parseSE(rtpPacket, this.byteOffset, this.bitOffset);
        mb_qp_delta = expGolomb.getValue();
        this.byteOffset = expGolomb.getNextByteOffset();
        this.bitOffset = expGolomb.getNextBitOffset();

        residual(rtpPacket);
        }
        else
        {
                for(int i = 0; i < 16; i++)
                        totalCoeffLuma[currMbAddr][i] = 0;
                for(int i = 0; i < 4; i++)
                        totalCoeffChromaCb[currMbAddr][i] = 0;
                for(int i = 0; i < 4; i++)
                        totalCoeffChromaCr[currMbAddr][i] = 0;
        }
}


public void sub_mb_pred(byte[] rtpPacket, int mb_type)
{
        for(int mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)
        {
                expGolomb.parseUE(rtpPacket, this.byteOffset, this.bitOffset);
                sub_mb_types[mbPartIdx] = expGolomb.getValue();
                this.byteOffset = expGolomb.getNextByteOffset();
                this.bitOffset = expGolomb.getNextBitOffset();

                if(slice_type == 1 || slice_type == 6) // B Slice
                        sub_mb_types[mbPartIdx] += 5;

                subMacroblockTypes[sub_mb_types[mbPartIdx]]++;
        }

        for(int mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)
        {
                if((num_ref_idx_l0_active_minus1 > 0 || mb_field_decoding_flag  == 1) &&
                   mb_type != P_8x8ref0 &&
                   sub_mb_types[mbPartIdx] != B_Direct_8x8 &&
                   subMbPredMode(sub_mb_types[mbPartIdx]) != Pred_L1)
                {
                        // ref_idx_l0
                        expGolomb.parseTE(rtpPacket,
                                          this.byteOffset,
                                          this.bitOffset,
                                          num_ref_idx_l0_active_minus1);
                        this.byteOffset = expGolomb.getNextByteOffset();
                        this.bitOffset = expGolomb.getNextBitOffset();
                }
        }
        for(int mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)
        {
                if((num_ref_idx_l1_active_minus1 > 0 || mb_field_decoding_flag == 1) &&
                   sub_mb_types[mbPartIdx] != B_Direct_8x8 &&
                   subMbPredMode(sub_mb_types[mbPartIdx]) != Pred_L0 )
                {
```

```
                        // ref_idx_l1
                        expGolomb.parseTE(rtpPacket,
                        this.byteOffset,
                        this.bitOffset,
                        num_ref_idx_l1_active_minus1);
                        this.byteOffset = expGolomb.getNextByteOffset();
                        this.bitOffset = expGolomb.getNextBitOffset();
                }
        }
        for(int mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)
        {
                if(sub_mb_types[mbPartIdx] != B_Direct_8x8 &&
                   subMbPredMode(sub_mb_types[mbPartIdx]) != Pred_L1 )
                {
                        for(int subMbPartIdx = 0; subMbPartIdx <
                                numSubMbPart(sub_mb_types[mbPartIdx]); subMbPartIdx++)
                        {
                                for(int compIdx = 0; compIdx < 2; compIdx++)
                                {
                                        // mvd_l0
                                        expGolomb.parseSE(rtpPacket,
                                                        this.byteOffset,
                                                        this.bitOffset);
                                        this.byteOffset = expGolomb.getNextByteOffset();
                                        this.bitOffset = expGolomb.getNextBitOffset();
                                }
                        }
                }
        }
        for(int mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)
        {
                if(sub_mb_types[mbPartIdx] != B_Direct_8x8 &&
                   subMbPredMode(sub_mb_types[mbPartIdx]) != Pred_L0 )
                {
                        for(int subMbPartIdx = 0; subMbPartIdx <
                                numSubMbPart(sub_mb_types[mbPartIdx]); subMbPartIdx++)
                        {
                                for(int compIdx = 0; compIdx < 2; compIdx++)
                                {
                                        // mvd_l1
                                        expGolomb.parseSE(rtpPacket,
                                                        this.byteOffset,
                                                        this.bitOffset);
                                this.byteOffset = expGolomb.getNextByteOffset();
                                this.bitOffset = expGolomb.getNextBitOffset();
                                }
                        }
                }
        }
}


public void mb_pred(byte[] rtpPacket)
{
        if(mbPartPredMode(mb_type, 0) == Intra_4x4 ||
           mbPartPredMode(mb_type, 0) == Intra_8x8 ||
           mbPartPredMode(mb_type, 0 ) == Intra_16x16)
        {
                if(mbPartPredMode(mb_type, 0) == Intra_4x4)
                {
                        for(int luma4x4BlkIdx=0; luma4x4BlkIdx<16; luma4x4BlkIdx++ )
                        {
                                // prev_intra4x4_pred_mode_flag
                                expGolomb.jumpXBits(rtpPacket,
                                                this.byteOffset,
                                                this.bitOffset,
                                                1);
                                prev_intra4x4_pred_mode_flag = expGolomb.getValue();
                                this.byteOffset = expGolomb.getNextByteOffset();
                                this.bitOffset = expGolomb.getNextBitOffset();

                                if(prev_intra4x4_pred_mode_flag == 0)
                                {
                                        // rem_intra4x4_pred_mode
                                        expGolomb.jumpXBits(rtpPacket,
                                                        this.byteOffset,
                                                        this.bitOffset,
                                                        3);
                                        this.byteOffset = expGolomb.getNextByteOffset();
                                        this.bitOffset = expGolomb.getNextBitOffset();
```

```
                        }
                    }
                }
                if(mbPartPredMode(mb_type, 0) == Intra_8x8)
                {
                    for(int luma4x4BlkIdx=0; luma4x4BlkIdx<16; luma4x4BlkIdx++ )
                    {
                        // prev_intra8x8_pred_mode_flag
                        expGolomb.jumpXBits(rtpPacket,
                                            this.byteOffset,
                                            this.bitOffset,
                                            1);
                        this.byteOffset = expGolomb.getNextByteOffset();
                        this.bitOffset = expGolomb.getNextBitOffset();

                        if(prev_intra8x8_pred_mode_flag == 0)
                        {
                            // rem_intra8x8_pred_mode
                            expGolomb.jumpXBits(rtpPacket,
                                                this.byteOffset,
                                                this.bitOffset,
                                                3);
                            this.byteOffset = expGolomb.getNextByteOffset();
                            this.bitOffset = expGolomb.getNextBitOffset();
                        }
                    }
                }
                if(sequenceParameterSet.chroma_format_idc != 0)
                {
                    // intra_chroma_pred_mode
                    expGolomb.parseUE(rtpPacket, this.byteOffset, this.bitOffset);
                    this.byteOffset = expGolomb.getNextByteOffset();
                    this.bitOffset = expGolomb.getNextBitOffset();
                }
            }
            else if(mbPartPredMode(mb_type, 0) != Direct)
            {
                for(int mbPartIdx = 0; mbPartIdx < numMbPart(mb_type); mbPartIdx++)
                {
                    if((num_ref_idx_l0_active_minus1 > 0 ||
                       mb_field_decoding_flag == 1) &&
                       mbPartPredMode(mb_type, mbPartIdx ) != Pred_L1)
                    {
                        // ref_idx_l0
                        expGolomb.parseTE(rtpPacket,
                                          this.byteOffset,
                                          this.bitOffset,
                                          num_ref_idx_l0_active_minus1);
                        this.byteOffset = expGolomb.getNextByteOffset();
                        this.bitOffset = expGolomb.getNextBitOffset();
                    }
                }
                for(int mbPartIdx = 0; mbPartIdx < numMbPart(mb_type); mbPartIdx++)
                {
                    if((num_ref_idx_l1_active_minus1 > 0 ||
                        mb_field_decoding_flag == 1) &&
                        mbPartPredMode(mb_type, mbPartIdx ) != Pred_L0)
                    {
                        // ref_idx_l1
                        expGolomb.parseTE(rtpPacket,
                                          this.byteOffset,
                                          this.bitOffset,
                                          num_ref_idx_l1_active_minus1);
                        this.byteOffset = expGolomb.getNextByteOffset();
                        this.bitOffset = expGolomb.getNextBitOffset();
                    }
                }
                for(int mbPartIdx = 0; mbPartIdx < numMbPart(mb_type); mbPartIdx++)
                {
                    if(mbPartPredMode (mb_type, mbPartIdx ) != Pred_L1 )
                    {
                        for(int compIdx = 0; compIdx < 2; compIdx++ )
                        {
                            // mvd_l0
                            expGolomb.parseSE(rtpPacket,
                                              this.byteOffset,
                                              this.bitOffset);
                            this.byteOffset = expGolomb.getNextByteOffset();
                            this.bitOffset = expGolomb.getNextBitOffset();
                        }
```

```
                        }
                }
                for(int mbPartIdx = 0; mbPartIdx < numMbPart(mb_type); mbPartIdx++)
                {
                        if(mbPartPredMode (mb_type, mbPartIdx ) != Pred_L0 )
                        {
                                for(int compIdx = 0; compIdx < 2; compIdx++ )
                                {
                                        // mvd_l1
                                        expGolomb.parseSE(rtpPacket,
                                                          this.byteOffset,
                                                          this.bitOffset);
                                        this.byteOffset = expGolomb.getNextByteOffset();
                                        this.bitOffset = expGolomb.getNextBitOffset();
                                }
                        }
                }
        }
}


public void residual(byte[] rtpPacket)
{
        if(mbPartPredMode(mb_type, 0) == Intra_16x16 )
                residual_block_cavlc(rtpPacket, Intra16x16DCLevel, 0, 0, 16);
        for(int i8x8 = 0; i8x8 < 4; i8x8++ )
        {
                if(transform_size_8x8_flag == 0 ||
                   pictureParameterSet.entropy_coding_mode_flag == 0)
                {
                        for(int i4x4 = 0; i4x4 < 4; i4x4++)
                        {
                                if((codedBlockPatternLuma & ( 1 << i8x8 )) != 0)
                                {
                                        if(mbPartPredMode(mb_type, 0) == Intra_16x16)
                                                residual_block_cavlc(rtpPacket,
                                                                     Intra16x16ACLevel,
                                                                     0,
                                                                     i8x8 * 4 + i4x4,
                                                                     15);
                                        else
                                                residual_block_cavlc(rtpPacket,
                                                                     LumaLevel,
                                                                     0,
                                                                     i8x8 * 4 + i4x4,
                                                                     16);
                                }
                                else
                                        totalCoeffLuma[currMbAddr][i8x8 * 4 + i4x4] = 0;
                        }
                }
                else if((codedBlockPatternLuma & ( 1 << i8x8 )) != 0)
                        residual_block_cavlc(rtpPacket, LumaLevel, 0, i8x8, 64);
        }
        if(sequenceParameterSet.chroma_format_idc != 0)
        {
                for(int iCbCr = 0; iCbCr < 2; iCbCr++ )
                        if((codedBlockPatternChroma & 0x03) != 0)
                                residual_block_cavlc(rtpPacket,
                                                     ChromaDCLevel,
                                                     iCbCr,
                                                     0,
                                                     4);
                for(int iCbCr = 0; iCbCr < 2; iCbCr++ )
                        for(int i8x8 = 0; i8x8 < 1; i8x8++ )
                                for(int i4x4 = 0; i4x4 < 4; i4x4++ )
                                        if((codedBlockPatternChroma & 0x02) != 0)
                                                residual_block_cavlc(rtpPacket,
                                                                     ChromaACLevel,
                                                                     iCbCr,
                                                                     i8x8 * 4 + i4x4,
                                                                     15);
                                        else
                                                if(iCbCr == 0)
                                                        totalCoeffChromaCb[currMbAddr]
                                                                          [i8x8 * 4 + i4x4] = 0;
                                                else
                                                        totalCoeffChromaCr[currMbAddr]
                                                                          [i8x8 * 4 + i4x4] = 0;
        }
```

```
}


public void residual_block_cavlc(byte[] rtpPacket,
                                 int coeffLevel,
                                 int cbcr,
                                 int blkIdx,
                                 int maxNumCoeff)
{
        int[] trainlingOnesAndTotalCoeff = new int[2];
        int level_prefix, level_suffix, suffixLength, levelSuffixSize;
        int zerosLeft, run_before = 0, levelCode, level;
        // coeff_token
        int nC = process9_2_1(coeffLevel, cbcr, blkIdx, maxNumCoeff);
        trainlingOnesAndTotalCoeff = cavlc.parseCoeffToken(rtpPacket,
                                                           this.byteOffset,
                                                           this.bitOffset,
                                                           nC);

        // needed to avoid fault state
        if(trainlingOnesAndTotalCoeff == null)
                return;

        this.byteOffset = cavlc.getNextByteOffset();
        this.bitOffset = cavlc.getNextBitOffset();

        if(coeffLevel == Intra16x16ACLevel || coeffLevel == LumaLevel)
        {
                trailingOnes[currMbAddr] = trainlingOnesAndTotalCoeff[0];
                totalCoeffLuma[currMbAddr][blkIdx] = trainlingOnesAndTotalCoeff[1];
        }
        if(coeffLevel == ChromaACLevel)
        {
                if(cbcr == 0)
                {
                        trailingOnes[currMbAddr] = trainlingOnesAndTotalCoeff[0];
                        totalCoeffChromaCb[currMbAddr][blkIdx] =
                                                        trainlingOnesAndTotalCoeff[1];
                }
                if(cbcr == 1)
                {
                        trailingOnes[currMbAddr] = trainlingOnesAndTotalCoeff[0];
                        totalCoeffChromaCr[currMbAddr][blkIdx] =
                                                        trainlingOnesAndTotalCoeff[1];
                }
        }

        if(trainlingOnesAndTotalCoeff[1] > 0)
        {
                if(trainlingOnesAndTotalCoeff[1] > 10 &&
                   trainlingOnesAndTotalCoeff[0] < 3)
                        suffixLength = 1;
                else
                        suffixLength = 0;

                for(int i = 0; i < trainlingOnesAndTotalCoeff[1]; i++)
                {
                        if( i < trainlingOnesAndTotalCoeff[0])
                        {
                                // trailing_ones_sign_flag
                                expGolomb.jumpXBits(rtpPacket,
                                                    this.byteOffset,
                                                    this.bitOffset,
                                                    1);
                                trailing_ones_sign_flag = expGolomb.getValue();
                                this.byteOffset = expGolomb.getNextByteOffset();
                                this.bitOffset = expGolomb.getNextBitOffset();

                                level = 1 - 2 * trailing_ones_sign_flag;
                        }
                        else
                        {
                                // level_prefix
                                level_prefix = cavlc.parseLevelPrefix(rtpPacket,
                                                                      this.byteOffset,
                                                                      this.bitOffset);
                                this.byteOffset = cavlc.getNextByteOffset();
                                this.bitOffset = cavlc.getNextBitOffset();

                                levelCode = (Math.min(15, level_prefix) << suffixLength);
```

```
                                if(suffixLength > 0 || level_prefix >= 14)
                                {
                                        if(level_prefix == 14 && suffixLength == 0)
                                                levelSuffixSize = 4;
                                        else if(level_prefix >= 15)
                                                levelSuffixSize = level_prefix - 3;
                                        else
                                                levelSuffixSize = suffixLength;

                                        // level_suffix
                                        expGolomb.jumpXBits(rtpPacket,
                                                            this.byteOffset,
                                                            this.bitOffset,
                                                            levelSuffixSize);
                                        level_suffix = expGolomb.getValue();
                                        this.byteOffset = expGolomb.getNextByteOffset();
                                        this.bitOffset = expGolomb.getNextBitOffset();

                                        levelCode += level_suffix;
                                }

                                if(level_prefix >= 15 && suffixLength == 0)
                                        levelCode += 15;
                                if(level_prefix >= 16)
                                        levelCode += ((1 << (level_prefix - 3)) - 4096);
                                if(i == trainlingOnesAndTotalCoeff[0] &&
                                   trainlingOnesAndTotalCoeff[0] < 3)
                                        levelCode += 2;
                                if(levelCode % 2 == 0)
                                        level = (levelCode + 2) >> 1;
                                else
                                        level = (-levelCode - 1) >> 1;
                                if(suffixLength == 0)
                                        suffixLength = 1;
                                if(Math.abs(level) > (3 << (suffixLength - 1)) &&
                                   suffixLength < 6)
                                        suffixLength++;
                        }
                }
                // total_zeros
                if(trainlingOnesAndTotalCoeff[1] < maxNumCoeff)
                {
                        zerosLeft = cavlc.parseTotalZeros(rtpPacket,
                                                          this.byteOffset,
                                                          this.bitOffset,
                                                          maxNumCoeff,
                                                          trainlingOnesAndTotalCoeff[1]);
                        this.byteOffset = cavlc.getNextByteOffset();
                        this.bitOffset = cavlc.getNextBitOffset();
                }
                else
                        zerosLeft = 0;


                // run_before
                for(int i = 0; i < trainlingOnesAndTotalCoeff[1] - 1; i++ )
                {
                        if(zerosLeft > 0)
                        {
                                run_before = cavlc.parseRunBefore(rtpPacket,
                                                                  this.byteOffset,
                                                                  this.bitOffset,
                                                                  zerosLeft);
                                this.byteOffset = cavlc.getNextByteOffset();
                                this.bitOffset = cavlc.getNextBitOffset();
                        }
                        zerosLeft -= run_before;
                }
        }
}


public int nextMbAddress(int n)
{
        return ++n;
}
```

```
public boolean more_rbsp_data(byte[] rtpPacket, int byteOffset, int bitOffset)
{
        if(currMbAddr >= ((sequenceParameterSet.pic_width_in_mbs_minus1 + 1) *
                          (sequenceParameterSet.pic_height_in_map_units_minus1 + 1) - 1))
                return false;
        else
                return true;
}


public boolean rbsp_trailing_bits(byte[] rtpPacket, int byteOffset, int bitOffset)
{
        int[] bitMasks = {0x80, 0x40, 0x20, 0x10, 0x08, 0x04, 0x02, 0x01};
        if((rtpPacket[byteOffset] & bitMasks[bitOffset++]) == 0)
                return false;
        else
                while(bitOffset != 8)
                        if((rtpPacket[byteOffset] & bitMasks[bitOffset++]) != 0)
                                return false;
        return true;
}


public void setMbWidthHeight()
{
        switch(sequenceParameterSet.chroma_format_idc)
        {
                case 1:
                        mbWidthC = 8;
                        mbHeightC = 8;
                        break;
                case 2:
                        mbWidthC = 8;
                        mbHeightC = 16;
                        break;
                case 3:
                        mbWidthC = 16;
                        mbHeightC = 16;
                        break;
        }
}


public int codedBlockPatternChroma(int mb_type)
{
        switch(mb_type)
        {
     case I_16x16_0_0_0: return 0;
     case I_16x16_1_0_0: return 0;
     case I_16x16_2_0_0: return 0;
     case I_16x16_3_0_0: return 0;
     case I_16x16_0_1_0: return 1;
     case I_16x16_1_1_0: return 1;
     case I_16x16_2_1_0: return 1;
     case I_16x16_3_1_0: return 1;
     case I_16x16_0_2_0: return 2;
     case I_16x16_1_2_0: return 2;
     case I_16x16_2_2_0: return 2;
     case I_16x16_3_2_0: return 2;
     case I_16x16_0_0_1: return 0;
     case I_16x16_1_0_1: return 0;
     case I_16x16_2_0_1: return 0;
     case I_16x16_3_0_1: return 0;
     case I_16x16_0_1_1: return 1;
     case I_16x16_1_1_1: return 1;
     case I_16x16_2_1_1: return 1;
     case I_16x16_3_1_1: return 1;
     case I_16x16_0_2_1: return 2;
     case I_16x16_1_2_1: return 2;
     case I_16x16_2_2_1: return 2;
     case I_16x16_3_2_1: return 2;
        }
        return 0;
}

public int codedBlockPatternLuma(int mb_type)
{
        switch(mb_type)
        {
     case I_16x16_0_0_0: return 0;
```

```
        case I_16x16_1_0_0: return 0;
        case I_16x16_2_0_0: return 0;
        case I_16x16_3_0_0: return 0;
        case I_16x16_0_1_0: return 0;
        case I_16x16_1_1_0: return 0;
        case I_16x16_2_1_0: return 0;
        case I_16x16_3_1_0: return 0;
        case I_16x16_0_2_0: return 0;
        case I_16x16_1_2_0: return 0;
        case I_16x16_2_2_0: return 0;
        case I_16x16_3_2_0: return 0;
        case I_16x16_0_0_1: return 15;
        case I_16x16_1_0_1: return 15;
        case I_16x16_2_0_1: return 15;
        case I_16x16_3_0_1: return 15;
        case I_16x16_0_1_1: return 15;
        case I_16x16_1_1_1: return 15;
        case I_16x16_2_1_1: return 15;
        case I_16x16_3_1_1: return 15;
        case I_16x16_0_2_1: return 15;
        case I_16x16_1_2_1: return 15;
        case I_16x16_2_2_1: return 15;
        case I_16x16_3_2_1: return 15;
           }
           return 0;
}


private int mbPartPredMode(int mb_type, int list)
{
        switch(mb_type)
        {
                case I_NxN: if(transform_size_8x8_flag == 0)
                                        return Intra_4x4;
                        if(transform_size_8x8_flag == 1)
                                return Intra_8x8;
                case I_16x16_0_0_0: return Intra_16x16;
                case I_16x16_1_0_0: return Intra_16x16;
                case I_16x16_2_0_0: return Intra_16x16;
                case I_16x16_3_0_0: return Intra_16x16;
                case I_16x16_0_1_0: return Intra_16x16;
                case I_16x16_1_1_0: return Intra_16x16;
                case I_16x16_2_1_0: return Intra_16x16;
                case I_16x16_3_1_0: return Intra_16x16;
                case I_16x16_0_2_0: return Intra_16x16;
                case I_16x16_1_2_0: return Intra_16x16;
                case I_16x16_2_2_0: return Intra_16x16;
                case I_16x16_3_2_0: return Intra_16x16;
                case I_16x16_0_0_1: return Intra_16x16;
                case I_16x16_1_0_1: return Intra_16x16;
                case I_16x16_2_0_1: return Intra_16x16;
                case I_16x16_3_0_1: return Intra_16x16;
                case I_16x16_0_1_1: return Intra_16x16;
                case I_16x16_1_1_1: return Intra_16x16;
                case I_16x16_2_1_1: return Intra_16x16;
                case I_16x16_3_1_1: return Intra_16x16;
                case I_16x16_0_2_1: return Intra_16x16;
                case I_16x16_1_2_1: return Intra_16x16;
                case I_16x16_2_2_1: return Intra_16x16;
                case I_16x16_3_2_1: return Intra_16x16;

                case P_L0_16x16:
                        if(list == 0)
                                return Pred_L0;
                case P_L0_L0_16x8:
                        if(list == 0)
                                return Pred_L0;
         if(list == 1)
                case P_L0_L0_8x16:
                        if(list == 0)
                                return Pred_L0;
         if(list == 1)
                return Pred_L0;

                case B_Direct_16x16:
                        if(list == 0)
                                return Direct;
                case B_L0_16x16:
                        if(list == 0)
```

```
                        return Pred_L0;
        case B_L1_16x16:
                if(list == 0)
                        return Pred_L1;
        case B_Bi_16x16:
                if(list == 0)
                        return BiPred;
        case B_L0_L0_16x8:
                if(list == 0)
                        return Pred_L0;
if(list == 1)
        return Pred_L0;
        case B_L0_L0_8x16:
                if(list == 0)
                        return Pred_L0;
if(list == 1)
        return Pred_L0;
        case B_L1_L1_16x8:
                if(list == 0)
                        return Pred_L1;
if(list == 1)
        return Pred_L1;
        case B_L1_L1_8x16:
                if(list == 0)
                        return Pred_L1;
if(list == 1)
        return Pred_L1;
        case B_L0_L1_16x8:
                if(list == 0)
                        return Pred_L0;
if(list == 1)
        return Pred_L1;
        case B_L0_L1_8x16:
                if(list == 0)
                        return Pred_L0;
if(list == 1)
        return Pred_L1;
        case B_L1_L0_16x8:
                if(list == 0)
                        return Pred_L1;
if(list == 1)
        return Pred_L0;
        case B_L1_L0_8x16:
                if(list == 0)
                        return Pred_L1;
if(list == 1)
        return Pred_L0;
        case B_L0_Bi_16x8:
                if(list == 0)
                        return Pred_L0;
if(list == 1)
        return BiPred;
        case B_L0_Bi_8x16:
                if(list == 0)
                        return Pred_L0;
if(list == 1)
        return BiPred;
        case B_L1_Bi_16x8:
                if(list == 0)
                        return Pred_L1;
if(list == 1)
        return BiPred;
        case B_L1_Bi_8x16:
                if(list == 0)
                        return Pred_L1;
if(list == 1)
        return BiPred;
        case B_Bi_L0_16x8:
                if(list == 0)
                        return BiPred;
if(list == 1)
        return Pred_L0;
        case B_Bi_L0_8x16:
                if(list == 0)
                        return BiPred;
if(list == 1)
        return Pred_L0;
        case B_Bi_L1_16x8:
                if(list == 0)
                        return BiPred;
```

```
         if(list == 1)
                return Pred_L1;
                case B_Bi_L1_8x16:
                        if(list == 0)
                                return BiPred;
         if(list == 1)
                return Pred_L1;
                case B_Bi_Bi_16x8:
                        if(list == 0)
                                return BiPred;
         if(list == 1)
                return BiPred;
                case B_Bi_Bi_8x16:
                        if(list == 0)
                                return BiPred;
         if(list == 1)
                return BiPred;
                case B_8x8:
                        if(list == 0)
                                return Direct;
        }
        return -1;
}


public int subMbPredMode(int sub_mb_type)
{
        switch (sub_mb_type)
        {
                case P_L0_8x8:
                case P_L0_8x4:
                case P_L0_4x8:
                case P_L0_4x4:
                case B_L0_8x8:
                case B_L0_8x4:
                case B_L0_4x8:
                case B_L0_4x4:
                        return Pred_L0;

                case B_L1_8x8:
                case B_L1_8x4:
                case B_L1_4x8:
                case B_L1_4x4:
                        return Pred_L1;

                case B_Bi_8x8:
                case B_Bi_8x4:
                case B_Bi_4x8:
                case B_Bi_4x4:
                        return BiPred;

                case B_Direct_8x8:
                        return Direct;
        }
        return -3;
}


public int numMbPart(int mb_type)
{
        switch(mb_type)
        {
                case P_L0_16x16:        return 1;
                case P_L0_L0_16x8:      return 2;
                case P_L0_L0_8x16:      return 2;
                case P_8x8:                     return 4;
                case P_8x8ref0:                 return 4;

                case B_L0_16x16:        return 1;
                case B_L1_16x16:        return 1;
                case B_Bi_16x16:        return 1;
                case B_L0_L0_16x8:      return 2;
                case B_L0_L0_8x16:      return 2;
                case B_L1_L1_16x8:      return 2;
                case B_L1_L1_8x16:      return 2;
                case B_L0_L1_16x8:      return 2;
                case B_L0_L1_8x16:      return 2;
                case B_L1_L0_16x8:      return 2;
                case B_L1_L0_8x16:      return 2;
                case B_L0_Bi_16x8:      return 2;
```

```
                case B_L0_Bi_8x16:    return 2;
                case B_L1_Bi_16x8:    return 2;
                case B_L1_Bi_8x16:    return 2;
                case B_Bi_L0_16x8:    return 2;
                case B_Bi_L0_8x16:    return 2;
                case B_Bi_L1_16x8:    return 2;
                case B_Bi_L1_8x16:    return 2;
                case B_Bi_Bi_16x8:    return 2;
                case B_Bi_Bi_8x16:    return 2;
                case B_8x8:                   return 4;
        }
        return 0;
}


public int numSubMbPart(int sub_mb_type)
{
        switch(sub_mb_type)
        {
                case P_L0_8x8:        return 1;
                case P_L0_8x4:        return 2;
                case P_L0_4x8:        return 2;
                case P_L0_4x4: return 4;

                case B_L0_8x8:        return 1;
                case B_L0_8x4:        return 2;
                case B_L0_4x8:        return 2;
                case B_L0_4x4: return 4;
                case B_L1_8x8:        return 1;
                case B_L1_8x4:        return 2;
                case B_L1_4x8:        return 2;
                case B_L1_4x4: return 4;
                case B_Bi_8x8:        return 1;
                case B_Bi_8x4:        return 2;
                case B_Bi_4x8:        return 2;
                case B_Bi_4x4: return 4;
                case B_Direct_8x8:    return 1;
        }
        return 0;
}



public int process9_2_1(int coeffLevel, int cbcr, int blkIdx, int maxNumCoeff)
{
        int nA = 0, nB = 0, nC = 0;
        int[] blkAB = new int[6];
        if(coeffLevel == ChromaDCLevel)
        nC = -1;
        else
        {
                if(coeffLevel == Intra16x16DCLevel)
                        blkIdx = 0;
                if(coeffLevel == Intra16x16DCLevel ||
                   coeffLevel == Intra16x16ACLevel ||
                   coeffLevel == LumaLevel)
                        blkAB = process6_4_8_3(blkIdx);
                if(coeffLevel == ChromaACLevel)
                        blkAB = process6_4_8_4(blkIdx);

                if(coeffLevel == Intra16x16DCLevel ||
                   coeffLevel == Intra16x16ACLevel ||
                   coeffLevel == LumaLevel)
                {
                if(blkAB[0] == 0 ||
                   pic_mb_types[blkAB[1]] == P_Skip ||
                   pic_mb_types[blkAB[1]] == B_Skip)
                        nA = 0;
                else
                        nA = totalCoeffLuma[blkAB[1]][blkAB[2]];
                if(blkAB[3] == 0 ||
                   pic_mb_types[blkAB[4]] == P_Skip ||
                   pic_mb_types[blkAB[4]] == B_Skip)
                        nB = 0;
                else
                        nB = totalCoeffLuma[blkAB[4]][blkAB[5]];
                }
                if(coeffLevel == ChromaACLevel)
                {
                        if(cbcr == 0)
```

```
                        {
                        if(blkAB[0] == 0 || pic_mb_types[blkAB[1]] == P_Skip)
                                nA = 0;
                        else
                                nA = totalCoeffChromaCb[blkAB[1]][blkAB[2]];
                        if(blkAB[3] == 0 || pic_mb_types[blkAB[4]] == P_Skip)
                                nB = 0;
                        else
                                nB = totalCoeffChromaCb[blkAB[4]][blkAB[5]];
                        }
                        if(cbcr == 1)
                        {
                        if(blkAB[0] == 0 || pic_mb_types[blkAB[1]] == P_Skip)
                                nA = 0;
                        else
                                nA = totalCoeffChromaCr[blkAB[1]][blkAB[2]];
                        if(blkAB[3] == 0 || pic_mb_types[blkAB[4]] == P_Skip)
                                nB = 0;
                        else
                                nB = totalCoeffChromaCr[blkAB[4]][blkAB[5]];
                        }
                }
                if(blkAB[0] == 1 && blkAB[3] == 1)
                        nC = ((nA + nB + 1) >> 1);
                else
                        nC = (nA + nB);
        }

        return nC;
}


public int[] process6_4_8_3(int luma4x4BlkIdx)
{
        int[] A = table6_2(1);
        int[] B = table6_2(2);

        int[] xy = process6_4_3(luma4x4BlkIdx);

        int xA = xy[0] + A[0];
        int yA = xy[1] + A[1];
        int xB = xy[0] + B[0];
        int yB = xy[1] + B[1];

        int[] AvailAddrxWyWA = process6_4_9(xA, yA, false);
        int luma4x4BlkIdxA = mapXYto4x4Idx(AvailAddrxWyWA[2], AvailAddrxWyWA[3]);

        int[] AvailAddrxWyWB = process6_4_9(xB, yB, false);
        int luma4x4BlkIdxB = mapXYto4x4Idx(AvailAddrxWyWB[2], AvailAddrxWyWB[3]);

        int[] output = new int[6];
        output[0] = AvailAddrxWyWA[0];
        output[1] = AvailAddrxWyWA[1];
        output[2] = luma4x4BlkIdxA;
        output[3] = AvailAddrxWyWB[0];
        output[4] = AvailAddrxWyWB[1];
        output[5] = luma4x4BlkIdxB;

        return output;
}


public int[] process6_4_8_4(int chroma4x4BlkIdx)
{
        int x = 0, y = 0;
        int[] A = table6_2(1);
        int[] B = table6_2(2);

        if(sequenceParameterSet.chroma_format_idc == 1 ||
           sequenceParameterSet.chroma_format_idc == 2)
        {
                x = inverseRasterScan(chroma4x4BlkIdx, 4, 4, 8, 0);
                y = inverseRasterScan(chroma4x4BlkIdx, 4, 4, 8, 1);
        }
        // TODO: if(chroma_format_idc == 3) - High Profile

        int xA = x + A[0];
        int yA = y + A[1];
        int xB = x + B[0];
        int yB = y + B[1];
```

```
        int[] AvailAddrxWyWA = process6_4_9(xA, yA, true);
        int chroma4x4BlkIdxA = mapXYto4x4Idx(AvailAddrxWyWA[2], AvailAddrxWyWA[3]);

        int[] AvailAddrxWyWB = process6_4_9(xB, yB, true);
        int chroma4x4BlkIdxB = mapXYto4x4Idx(AvailAddrxWyWB[2], AvailAddrxWyWB[3]);

        int[] output = new int[6];
        output[0] = AvailAddrxWyWA[0];
        output[1] = AvailAddrxWyWA[1];
        output[2] = chroma4x4BlkIdxA;
        output[3] = AvailAddrxWyWB[0];
        output[4] = AvailAddrxWyWB[1];
        output[5] = chroma4x4BlkIdxB;

        return output;
}


public int[] process6_4_3(int luma4x4BlkIdx)
{
        int[] xy = new int[2];
        // x
        xy[0] = inverseRasterScan(luma4x4BlkIdx/4, 8, 8, 16, 0) +
                                        inverseRasterScan(luma4x4BlkIdx%4, 4, 4, 8, 0);
        // y
        xy[1] = inverseRasterScan(luma4x4BlkIdx/4, 8, 8, 16, 1) +
                                        inverseRasterScan(luma4x4BlkIdx%4, 4, 4, 8, 1);

        return xy;
}


public int[] process6_4_9(int xN, int yN, boolean is4x4chroma)
{
        int maxW = 16, maxH = 16;
        if(is4x4chroma)
        {
                maxW = mbWidthC;
                maxH = mbHeightC;
        }

        if(mbaffFrameFlag == 0)
                return process6_4_9_1(xN, yN, maxW, maxH);
        // TODO: else return process6_4_9_2(xN, yN, maxW, maxH);
        return null;
}


public int[] process6_4_9_1(int xN, int yN, int maxW, int maxH)
{
        int[] availAddrxWyW = new int[4];
        int[] availabilityAndAddressMbAddrAMbAddrB = process6_4_6();

        if(xN < 0 && yN >= 0 && yN <= (maxH - 1))
        {
                // MbAddrA
                availAddrxWyW[0] = availabilityAndAddressMbAddrAMbAddrB[0];
                availAddrxWyW[1] = availabilityAndAddressMbAddrAMbAddrB[1];
        }
        if(xN >= 0 && xN <= (maxW - 1) && yN < 0)
        {
                // MbAddrB
                availAddrxWyW[0] = availabilityAndAddressMbAddrAMbAddrB[2];
                availAddrxWyW[1] = availabilityAndAddressMbAddrAMbAddrB[3];
        }
        if(xN >= 0 && xN <= (maxW - 1) && yN >= 0 && yN <= (maxH - 1))
        {
                // CurrMbAddr
                availAddrxWyW[0] = 1;
                availAddrxWyW[1] = currMbAddr;
        }

        int xW = (xN + maxW) % maxW;
        int yW = (yN + maxH) % maxH;
        availAddrxWyW[2] = xW;
        availAddrxWyW[3] = yW;

        return availAddrxWyW;
}
```

```
public int[] process6_4_6()
{
        int PicWidthInMbs = sequenceParameterSet.pic_width_in_mbs_minus1 + 1;
        int[] AvailibilityAndAddress = new int[4];
        // MbAddrA
        AvailibilityAndAddress[0] = process6_4_5(currMbAddr - 1);
        AvailibilityAndAddress[1] = currMbAddr - 1;
        // MbAddrB
        AvailibilityAndAddress[2] = process6_4_5(currMbAddr - PicWidthInMbs);
        AvailibilityAndAddress[3] = currMbAddr - PicWidthInMbs;

        return AvailibilityAndAddress;
}


public int process6_4_5(int mbAddr)
{
        int PicWidthInMbs = sequenceParameterSet.pic_width_in_mbs_minus1 + 1;
        if(currMbAddr % PicWidthInMbs == 0 &&
          (mbAddr == currMbAddr - 1 || mbAddr == currMbAddr - PicWidthInMbs - 1))
                return 0;
        if(mbAddr < 0)
                return 0;
        if(mbAddr > currMbAddr)
                return 0;
        return 1;
}


public int[] table6_2(int AB)
{
        if(AB == 1)
        {
                int[] A = {-1, 0};
                return A;
        }
        if(AB == 2)
        {
                int[] B = {0, -1};
                return B;
        }
        return null;
}


public int inverseRasterScan(int a, int b, int c, int d, int e)
{
        if(e == 0)
                return ((a%(d/b))*b);
        if(e == 1)
                return ((a/(d/b))*c);
        return -1;
}


public int mapXYto4x4Idx(int x, int y)
{
        x /= 4;
        y /= 4;
        if(x == 0 && y == 0)
                return 0;
        if(x == 0 && y == 1)
                return 2;
        if(x == 0 && y == 2)
                return 8;
        if(x == 0 && y == 3)
                return 10;
        if(x == 1 && y == 0)
                return 1;
        if(x == 1 && y == 1)
                return 3;
        if(x == 1 && y == 2)
                return 9;
        if(x == 1 && y == 3)
                return 11;
        if(x == 2 && y == 0)
                return 4;
        if(x == 2 && y == 1)
```

```
                return 6;
        if(x == 2 && y == 2)
                return 12;
        if(x == 2 && y == 3)
                return 14;
        if(x == 3 && y == 0)
                return 5;
        if(x == 3 && y == 1)
                return 7;
        if(x == 3 && y == 2)
                return 13;
        if(x == 3 && y == 3)
                return 15;
        return -1;
    }


    public class SequenceParameterSet
    {
        int sequence_parameter_set_id;
        int log2_max_frame_num_minus4;
        int frame_mbs_only_flag;
        int pic_order_cnt_type;
        int log2_max_pic_order_cnt_lsb_minus4;
        int delta_pic_order_always_zero_flag;
        int mb_adaptive_frame_field_flag;
        int chroma_format_idc;
        int direct_8x8_inference_flag;
        int pic_width_in_mbs_minus1;
        int pic_height_in_map_units_minus1;
    }

    public class PictureParameterSet
    {
        int picture_parameter_set_id;
        int sequence_parameter_set_id;
        int pic_order_present_flag;
        int redundant_pic_cnt_present_flag;
        int entropy_coding_mode_flag;
        int deblocking_filter_control_present_flag;
        int num_slice_groups_minus1;
        int slice_group_map_type;
        int pic_size_in_map_units_minus1;
        int slice_group_change_rate_minus1;
        int transform_8x8_mode_flag;
    }
}
```

D.1.9 Die Klasse ExpGolomb

```
public class ExpGolomb
{
    private int value;
    private int nextByteOffset;
    private int nextBitOffset;


    public ExpGolomb()
    {
        nextByteOffset = 0;
        nextBitOffset = 0;
    }


    public void parseUE(byte[] data, int byteOffset, int bitOffset)
    {
        // needed to avoid fault state
        if(byteOffset == data.length - 1)
            return;

        this.value = 0;
        int bit, valueBits, leadingZeroBits = -1;
        int[] bitMasks = {0x80, 0x40, 0x20, 0x10, 0x08, 0x04, 0x02, 0x01};

        byteOffset = emulationPreventionDetection(data, byteOffset);
```

```
		do
		{
			leadingZeroBits++;
			bit = (data[byteOffset] & bitMasks[bitOffset]);
			bitOffset++;
			bitOffset %= 8;
			if(bitOffset == 0)
			{
				byteOffset++;
				byteOffset = emulationPreventionDetection(data, byteOffset);
			}
		}
		while (bit == 0 && byteOffset < data.length - 1);
		valueBits = leadingZeroBits;
		while (valueBits > 0 && byteOffset < data.length - 1)
		{
			valueBits--;
			bit = (data[byteOffset] & bitMasks[bitOffset]);
			this.value = (this.value << 1) | (bit >> (7 - bitOffset));
			bitOffset++;
			bitOffset %= 8;
			if(bitOffset == 0)
			{
				byteOffset++;
				byteOffset = emulationPreventionDetection(data, byteOffset);
			}
		}

		this.value += (Math.pow(2, leadingZeroBits) - 1);
		this.nextByteOffset = byteOffset;
		this.nextBitOffset = bitOffset;
	}


	public void parseSE(byte[] data, int byteOffset, int bitOffset)
	{
		parseUE(data, byteOffset, bitOffset);
		this.value = (int)Math.pow(-1, this.value + 1) *
				ExtendedMath.ceil(((double)this.value) / 2);
	}


	public void parseME(byte[] data, int byteOffset, int bitOffset, int mb_type)
	{
		parseUE(data, byteOffset, bitOffset);
		if(mb_type > 4 && mb_type < 31)
			this.value = mapCodedBlockPattern(this.value, 0);
		else
			this.value = mapCodedBlockPattern(this.value, 1);
	}


	public void parseTE(byte[] data, int byteOffset, int bitOffset, int range)
	{
		// needed to avoid fault state
		if(byteOffset == data.length - 1)
			return;

		int bit;
		int[] bitMasks = {0x80, 0x40, 0x20, 0x10, 0x08, 0x04, 0x02, 0x01};

		if(range == 1)
		{
			byteOffset = emulationPreventionDetection(data, byteOffset);
			bit = (data[byteOffset] & bitMasks[bitOffset]);
			this.value = (bit >> (7 - bitOffset));
			bitOffset++;
			bitOffset %= 8;
			if(bitOffset == 0)
			{
				byteOffset++;
				byteOffset = emulationPreventionDetection(data, byteOffset);
			}
			if(value == 1)
				value = 0;
			if(value == 0)
				value = 1;

			this.nextByteOffset = byteOffset;
			this.nextBitOffset = bitOffset;
```

```
        }
        else
                parseUE(data, byteOffset, bitOffset);
}


public void jumpXBits(byte[] data, int byteOffset, int bitOffset, int x)
{
        this.value = 0;
        int bit;
        int[] bitMasks = {0x80, 0x40, 0x20, 0x10, 0x08, 0x04, 0x02, 0x01};

        byteOffset = emulationPreventionDetection(data, byteOffset);
        do
        {
                bit = (data[byteOffset] & bitMasks[bitOffset]);
                this.value = (this.value << 1) | (bit >> (7 - bitOffset));
                bitOffset++;
                bitOffset %= 8;
                if(bitOffset == 0)
                {
                        byteOffset++;
                        byteOffset = emulationPreventionDetection(data, byteOffset);
                }
        }
        while(--x > 0);

        this.nextByteOffset = byteOffset;
        this.nextBitOffset = bitOffset;
}


// Ignore Start Code Emulation Prevention Byte
public int emulationPreventionDetection(byte[] data, int byteOffset)
{
        if(byteOffset >= 2)
                if((data[byteOffset] & 0xff) == 3 &&
                   (data[byteOffset - 1] & 0xff) == 0 &&
                   (data[byteOffset - 2] & 0xff) == 0)
                       byteOffset++;
        return byteOffset;
}


public int mapCodedBlockPattern(int value, int intraOrInter)
{
        if(intraOrInter == 0)
                switch(value)
                {
                        case 0: return 47;
                        case 1: return 31;
                        case 2: return 15;
                        case 3: return 0;
                        case 4: return 23;
                        case 5: return 27;
                        case 6: return 29;
                        case 7: return 30;
                        case 8: return 7;
                        case 9: return 11;
                        case 10: return 13;
                        case 11: return 14;
                        case 12: return 39;
                        case 13: return 43;
                        case 14: return 45;
                        case 15: return 46;
                        case 16: return 16;
                        case 17: return 3;
                        case 18: return 5;
                        case 19: return 10;
                        case 20: return 12;
                        case 21: return 19;
                        case 22: return 21;
                        case 23: return 26;
                        case 24: return 28;
                        case 25: return 35;
                        case 26: return 37;
                        case 27: return 42;
                        case 28: return 44;
                        case 29: return 1;
                        case 30: return 2;
```

```
                        case 31: return 4;
                        case 32: return 8;
                        case 33: return 17;
                        case 34: return 18;
                        case 35: return 20;
                        case 36: return 24;
                        case 37: return 6;
                        case 38: return 9;
                        case 39: return 22;
                        case 40: return 25;
                        case 41: return 32;
                        case 42: return 33;
                        case 43: return 34;
                        case 44: return 36;
                        case 45: return 40;
                        case 46: return 38;
                        case 47: return 41;
                }
        if(intraOrInter == 1)
                switch(value)
                {
                        case 0: return 0;
                        case 1: return 16;
                        case 2: return 1;
                        case 3: return 2;
                        case 4: return 4;
                        case 5: return 8;
                        case 6: return 32;
                        case 7: return 3;
                        case 8: return 5;
                        case 9: return 10;
                        case 10: return 12;
                        case 11: return 15;
                        case 12: return 47;
                        case 13: return 7;
                        case 14: return 11;
                        case 15: return 13;
                        case 16: return 14;
                        case 17: return 6;
                        case 18: return 9;
                        case 19: return 31;
                        case 20: return 35;
                        case 21: return 37;
                        case 22: return 42;
                        case 23: return 44;
                        case 24: return 33;
                        case 25: return 34;
                        case 26: return 36;
                        case 27: return 40;
                        case 28: return 39;
                        case 29: return 43;
                        case 30: return 45;
                        case 31: return 46;
                        case 32: return 17;
                        case 33: return 18;
                        case 34: return 20;
                        case 35: return 24;
                        case 36: return 19;
                        case 37: return 21;
                        case 38: return 26;
                        case 39: return 28;
                        case 40: return 23;
                        case 41: return 27;
                        case 42: return 29;
                        case 43: return 30;
                        case 44: return 22;
                        case 45: return 25;
                        case 46: return 38;
                        case 47: return 41;
                }
        return -1;
}


public int getValue()
{
        return value;
}
public void setValue(int value)
{
```

```
                this.value = value;
        }


        public int getNextByteOffset()
        {
                return nextByteOffset;
        }


        public void setNextByteOffset(int nextByteOffset)
        {
                this.nextByteOffset = nextByteOffset;
        }


        public int getNextBitOffset()
        {
                return nextBitOffset;
        }


        public void setNextBitOffset(int nextBitOffset)
        {
                this.nextBitOffset = nextBitOffset;
        }
}
```

D.1.10 Die Klasse CAVLC

```
public class CAVLC
{
        private int value;
        private int nextByteOffset;
        private int nextBitOffset;

        // mapping of ITU H.264 Table 9-5 row index to TotalCoeff and TrailingOnes
        private int[][] trailingOnesAndTotalCoeff;

        // ct_coeffToken.length_nC.values
        private int[][] ct_1_nC01;
        private int[][] ct_2_nC01;
        private int[][] ct_3_nC01;
        private int[][] ct_5_nC01;
        private int[][] ct_6_nC01;
        private int[][] ct_7_nC01;
        private int[][] ct_8_nC01;
        private int[][] ct_9_nC01;
        private int[][] ct_10_nC01;
        private int[][] ct_11_nC01;
        private int[][] ct_13_nC01;
        private int[][] ct_14_nC01;
        private int[][] ct_15_nC01;
        private int[][] ct_16_nC01;
        private int[][] ct_2_nC23;
        private int[][] ct_3_nC23;
        private int[][] ct_4_nC23;
        private int[][] ct_5_nC23;
        private int[][] ct_6_nC23;
        private int[][] ct_7_nC23;
        private int[][] ct_8_nC23;
        private int[][] ct_9_nC23;
        private int[][] ct_11_nC23;
        private int[][] ct_12_nC23;
        private int[][] ct_13_nC23;
        private int[][] ct_14_nC23;
        private int[][] ct_4_nC4567;
        private int[][] ct_5_nC4567;
        private int[][] ct_6_nC4567;
        private int[][] ct_7_nC4567;
        private int[][] ct_8_nC4567;
        private int[][] ct_9_nC4567;
        private int[][] ct_10_nC4567;
        private int[][] ct_6_nCgt8;
        private int[][] ct_1_nCminus1;
```

```
private int[][] ct_2_nCminus1;
private int[][] ct_3_nCminus1;
private int[][] ct_6_nCminus1;
private int[][] ct_7_nCminus1;
private int[][] ct_8_nCminus1;
private int[][] ct_1_nCminus2;
private int[][] ct_2_nCminus2;
private int[][] ct_3_nCminus2;
private int[][] ct_5_nCminus2;
private int[][] ct_6_nCminus2;
private int[][] ct_7_nCminus2;
private int[][] ct_9_nCminus2;
private int[][] ct_10_nCminus2;
private int[][] ct_11_nCminus2;
private int[][] ct_12_nCminus2;
private int[][] ct_13_nCminus2;

// tz_totalCoeff_maxNumCoeff.value
private int[][] tz_1_maxNumCoeffsX;
private int[][] tz_2_maxNumCoeffsX;
private int[][] tz_3_maxNumCoeffsX;
private int[][] tz_4_maxNumCoeffsX;
private int[][] tz_5_maxNumCoeffsX;
private int[][] tz_6_maxNumCoeffsX;
private int[][] tz_7_maxNumCoeffsX;
private int[][] tz_8_maxNumCoeffsX;
private int[][] tz_9_maxNumCoeffsX;
private int[][] tz_10_maxNumCoeffsX;
private int[][] tz_11_maxNumCoeffsX;
private int[][] tz_12_maxNumCoeffsX;
private int[][] tz_13_maxNumCoeffsX;
private int[][] tz_14_maxNumCoeffsX;
private int[][] tz_15_maxNumCoeffsX;
private int[][] tz_1_maxNumCoeffs4;
private int[][] tz_2_maxNumCoeffs4;
private int[][] tz_3_maxNumCoeffs4;

// rb_zerosLeft
private int[][] rb_1;
private int[][] rb_2;
private int[][] rb_3;
private int[][] rb_4;
private int[][] rb_5;
private int[][] rb_6;
private int[][] rb_gt6;


public CAVLC()
{
        nextByteOffset = 0;
        nextBitOffset = 0;

        createTrailingOnesAndTotalCoeffField();
        createCoeffTokenFields();
        createTotalZerosFields();
        createRunBeforeField();
}


public int[] parseCoeffToken(byte[] data, int byteOffset, int bitOffset, int nC)
{

        // needed to avoid fault state
        if(byteOffset == data.length - 1)
                return null;

        this.value = 0;
        int bit, length = 0, idx = 0;
        int[] bitMasks = {0x80, 0x40, 0x20, 0x10, 0x08, 0x04, 0x02, 0x01};

        if(nC >= 8)
        {
                do
                {
                        length++;
                        bit = (data[byteOffset] & bitMasks[bitOffset]);
                        this.value = (this.value << 1) | (bit >> (7 - bitOffset));
                        bitOffset++;
                        bitOffset %= 8;
                        if(bitOffset == 0)
```

```
                {
                        byteOffset++;
                        byteOffset = emulationPreventionDetection(data,
                                                                byteOffset);
                }
        }
        while (length != 6 && byteOffset < data.length - 1);
        idx = matchCoeffToken(length, nC);
}
else if(nC == -1)
{
        byteOffset = emulationPreventionDetection(data, byteOffset);
        do
        {
                length++;
                bit = (data[byteOffset] & bitMasks[bitOffset]);
                this.value = (this.value << 1) | (bit >> (7 - bitOffset));
                bitOffset++;
                bitOffset %= 8;
                if(bitOffset == 0)
                {
                        byteOffset++;
                        byteOffset = emulationPreventionDetection(data,
                                                                byteOffset);
                }
        }
        while (bit == 0 &&
               !(length == 7 && this.value == 0) &&
               byteOffset < data.length - 1);
        if(length == 7 && this.value == 0)
                idx = matchCoeffToken(length, nC);
        else
        {
                while ((idx = matchCoeffToken(length, nC)) < 0
                        && byteOffset < data.length - 1)
                {
                        length++;
                        bit = (data[byteOffset] & bitMasks[bitOffset]);
                        this.value = (this.value << 1) |
                                     (bit >> (7 - bitOffset));
                        bitOffset++;
                        bitOffset %= 8;
                        if(bitOffset == 0)
                        {
                                byteOffset++;
                                byteOffset = emulationPreventionDetection(data,
                                                                byteOffset);
                        }
                }
        }
}
else
{
        byteOffset = emulationPreventionDetection(data, byteOffset);
        do
        {
                length++;
                bit = (data[byteOffset] & bitMasks[bitOffset]);
                bitOffset++;
                bitOffset %= 8;
                if(bitOffset == 0)
                {
                        byteOffset++;
                        byteOffset = emulationPreventionDetection(data,
                                                                byteOffset);
                }
        }
        while (bit == 0 && byteOffset < data.length - 1);
        this.value = 1;
        while ((idx = matchCoeffToken(length, nC)) < 0 &&
                byteOffset < data.length - 1)
        {
                length++;
                bit = (data[byteOffset] & bitMasks[bitOffset]);
                this.value = (this.value << 1) | (bit >> (7 - bitOffset));
                bitOffset++;
                bitOffset %= 8;
                if(bitOffset == 0)
                {
                        byteOffset++;
```

```
                                byteOffset = emulationPreventionDetection(data,
                                                                          byteOffset);
                        }
                    }
                }

                this.nextByteOffset = byteOffset;
                this.nextBitOffset = bitOffset;

                return mapTotalCoeffAndTrailingOnes(idx);
        }


        public int emulationPreventionDetection(byte[] data, int byteOffset)
        {
                if(byteOffset >= 2)
                        if((data[byteOffset] & 0xff) == 3 &&
                           (data[byteOffset - 1] & 0xff) == 0 &&
                           (data[byteOffset - 2] & 0xff) == 0)
                               byteOffset++;
                return byteOffset;
        }


        public int matchCoeffToken(int length, int nC)
        {
                int[][] ct = null;
                switch(nC)
                {
                        case -2:
                                switch(length)
                                {
                                        case 1:
                                                ct = ct_1_nCminus2;
                                                break;
                                        case 2:
                                                ct = ct_2_nCminus2;
                                                break;
                                        case 3:
                                                ct = ct_3_nCminus2;
                                                break;
                                        case 5:
                                                ct = ct_5_nCminus2;
                                                break;
                                        case 6:
                                                ct = ct_6_nCminus2;
                                                break;
                                        case 7:
                                                ct = ct_7_nCminus2;
                                                break;
                                        case 9:
                                                ct = ct_9_nCminus2;
                                                break;
                                        case 10:
                                                ct = ct_10_nCminus2;
                                                break;
                                        case 11:
                                                ct = ct_11_nCminus2;
                                                break;
                                        case 12:
                                                ct = ct_12_nCminus2;
                                                break;
                                        case 13:
                                                ct = ct_13_nCminus2;
                                                break;
                                }
                                break;
                        case -1:
                                switch(length)
                                {
                                        case 1:
                                                ct = ct_1_nCminus1;
                                                break;
                                        case 2:
                                                ct = ct_2_nCminus1;
                                                break;
                                        case 3:
                                                ct = ct_3_nCminus1;
                                                break;
                                        case 6:
```

```
                        ct = ct_6_nCminus1;
                        break;
                case 7:
                        ct = ct_7_nCminus1;
                        break;
                case 8:
                        ct = ct_8_nCminus1;
                        break;
        }
        break;
case 0:
case 1:
        switch(length)
        {
                case 1:
                        ct = ct_1_nC01;
                        break;
                case 2:
                        ct = ct_2_nC01;
                        break;
                case 3:
                        ct = ct_3_nC01;
                        break;
                case 5:
                        ct = ct_5_nC01;
                        break;
                case 6:
                        ct = ct_6_nC01;
                        break;
                case 7:
                        ct = ct_7_nC01;
                        break;
                case 8:
                        ct = ct_8_nC01;
                        break;
                case 9:
                        ct = ct_9_nC01;
                        break;
                case 10:
                        ct = ct_10_nC01;
                        break;
                case 11:
                        ct = ct_11_nC01;
                        break;
                case 13:
                        ct = ct_13_nC01;
                        break;
                case 14:
                        ct = ct_14_nC01;
                        break;
                case 15:
                        ct = ct_15_nC01;
                        break;
                case 16:
                        ct = ct_16_nC01;
                        break;
        }
        break;
case 2:
case 3:
        switch(length)
        {
                case 2:
                        ct = ct_2_nC23;
                        break;
                case 3:
                        ct = ct_3_nC23;
                        break;
                case 4:
                        ct = ct_4_nC23;
                        break;
                case 5:
                        ct = ct_5_nC23;
                        break;
                case 6:
                        ct = ct_6_nC23;
                        break;
                case 7:
                        ct = ct_7_nC23;
                        break;
```

```
                                case 8:
                                        ct = ct_8_nC23;
                                        break;
                                case 9:
                                        ct = ct_9_nC23;
                                        break;
                                case 11:
                                        ct = ct_11_nC23;
                                        break;
                                case 12:
                                        ct = ct_12_nC23;
                                        break;
                                case 13:
                                        ct = ct_13_nC23;
                                        break;
                                case 14:
                                        ct = ct_14_nC23;
                                        break;
                        }
                        break;
                case 4:
                case 5:
                case 6:
                case 7:
                        switch(length)
                        {
                                case 4:
                                        ct = ct_4_nC4567;
                                        break;
                                case 5:
                                        ct = ct_5_nC4567;
                                        break;
                                case 6:
                                        ct = ct_6_nC4567;
                                        break;
                                case 7:
                                        ct = ct_7_nC4567;
                                        break;
                                case 8:
                                        ct = ct_8_nC4567;
                                        break;
                                case 9:
                                        ct = ct_9_nC4567;
                                        break;
                                case 10:
                                        ct = ct_10_nC4567;
                                        break;

                        }
                        break;
                default:
                        ct = ct_6_nCgt8;
        }

        if(ct != null)
                for(int i = 0; i < ct.length; i++)
                        if(ct[i][0] == this.value)
                                return ct[i][1];
        return -1;
}


public int[] mapTotalCoeffAndTrailingOnes(int idx)
{
        if(trailingOnesAndTotalCoeff != null && idx >= 0)
                return trailingOnesAndTotalCoeff[idx];
        return null;
}


public int parseLevelPrefix(byte[] data, int byteOffset, int bitOffset)
{
        int bit, levelPrefix = -1;
        int[] bitMasks = {0x80, 0x40, 0x20, 0x10, 0x08, 0x04, 0x02, 0x01};

        byteOffset = emulationPreventionDetection(data, byteOffset);
        do
        {
                levelPrefix++;
                bit = (data[byteOffset] & bitMasks[bitOffset]);
```

```
                bitOffset++;
                bitOffset %= 8;
                if(bitOffset == 0)
                {
                        byteOffset++;
                        byteOffset = emulationPreventionDetection(data, byteOffset);
                }
        }
        while (bit == 0 && byteOffset < data.length - 1);

        this.value = levelPrefix;
        this.nextByteOffset = byteOffset;
        this.nextBitOffset = bitOffset;

        return levelPrefix;
}


public int parseTotalZeros(byte[] data,
                        int byteOffset,
                        int bitOffset,
                        int maxNumCoeff,
                        int totalCoeff)
{
        this.value = 0;
        int bit, length = 0, totalZeros;
        int[] bitMasks = {0x80, 0x40, 0x20, 0x10, 0x08, 0x04, 0x02, 0x01};

        byteOffset = emulationPreventionDetection(data, byteOffset);
        do
        {
                length++;
                bit = (data[byteOffset] & bitMasks[bitOffset]);
                this.value = (this.value << 1) | (bit >> (7 - bitOffset));
                bitOffset++;
                bitOffset %= 8;
                if(bitOffset == 0)
                {
                        byteOffset++;
                        byteOffset = emulationPreventionDetection(data, byteOffset);
                }
        }
        while ((totalZeros = matchTotalZeros(length, maxNumCoeff, totalCoeff)) < 0 &&
                byteOffset < data.length - 1);

        this.value = totalZeros;
        this.nextByteOffset = byteOffset;
        this.nextBitOffset = bitOffset;

        return totalZeros;
}


public int matchTotalZeros(int length, int maxNumCoeff, int totalCoeff)
{
        int[][] tz = null;
        if(maxNumCoeff == 4)
        {
        switch(totalCoeff)
        {
                case 1:
                        tz = tz_1_maxNumCoeffs4;
                        break;
                case 2:
                        tz = tz_2_maxNumCoeffs4;
                        break;
                case 3:
                        tz = tz_3_maxNumCoeffs4;
                        break;
        }
        }
        else
        {
                switch(totalCoeff)
        {
                case 1:
                        tz = tz_1_maxNumCoeffsX;
                        break;
                case 2:
                        tz = tz_2_maxNumCoeffsX;
```

```
                        break;
                case 3:
                        tz = tz_3_maxNumCoeffsX;
                        break;
                case 4:
                        tz = tz_4_maxNumCoeffsX;
                        break;
                case 5:
                        tz = tz_5_maxNumCoeffsX;
                        break;
                case 6:
                        tz = tz_6_maxNumCoeffsX;
                        break;
                case 7:
                        tz = tz_7_maxNumCoeffsX;
                        break;
                case 8:
                        tz = tz_8_maxNumCoeffsX;
                        break;
                case 9:
                        tz = tz_9_maxNumCoeffsX;
                        break;
                case 10:
                        tz = tz_10_maxNumCoeffsX;
                        break;
                case 11:
                        tz = tz_11_maxNumCoeffsX;
                        break;
                case 12:
                        tz = tz_12_maxNumCoeffsX;
                        break;
                case 13:
                        tz = tz_13_maxNumCoeffsX;
                        break;
                case 14:
                        tz = tz_14_maxNumCoeffsX;
                        break;
                case 15:
                        tz = tz_15_maxNumCoeffsX;
                        break;
        }
        }
        if(tz != null)
                for(int i = 0; i < tz.length; i++)
                        if(tz[i][1] == length && tz[i][0] == this.value)
                                return i;
        return -1;
}


public int parseRunBefore(byte[] data, int byteOffset, int bitOffset, int zerosLeft)
{

        this.value = 0;
        int bit, length = 0, runBefore;
        int[] bitMasks = {0x80, 0x40, 0x20, 0x10, 0x08, 0x04, 0x02, 0x01};

        byteOffset = emulationPreventionDetection(data, byteOffset);
        do
        {
                length++;
                bit = (data[byteOffset] & bitMasks[bitOffset]);
                this.value = (this.value << 1) | (bit >> (7 - bitOffset));
                bitOffset++;
                bitOffset %= 8;
                if(bitOffset == 0)
                {
                        byteOffset++;
                        byteOffset = emulationPreventionDetection(data, byteOffset);
                }
        }
        while ((runBefore = matchRunBefore(length, zerosLeft)) < 0 &&
                byteOffset < data.length - 1);

        this.value = runBefore;
        this.nextByteOffset = byteOffset;
        this.nextBitOffset = bitOffset;

        return runBefore;
}
```

```
public int matchRunBefore(int length, int zerosLeft)
{
        int[][] rb = null;
        switch(zerosLeft)
        {
                case 1:
                        rb = rb_1;
                        break;
                case 2:
                        rb = rb_2;
                        break;
                case 3:
                        rb = rb_3;
                        break;
                case 4:
                        rb = rb_4;
                        break;
                case 5:
                        rb = rb_5;
                        break;
                case 6:
                        rb = rb_6;
                        break;
                default:
                        rb = rb_gt6;
                        break;
        }

        if(rb != null)
                for(int i = 0; i < rb.length; i++)
                        if(rb[i][1] == length && rb[i][0] == this.value)
                                return i;
        return -1;
}


public void createTrailingOnesAndTotalCoeffField()
{
        // tc_to[x][0] = TotalCoeffs
        // tc_to[x][1] = TrailingOnes
        trailingOnesAndTotalCoeff = new int[62][2];
        trailingOnesAndTotalCoeff[0][0] = 0;
        trailingOnesAndTotalCoeff[0][1] = 0;
        trailingOnesAndTotalCoeff[1][0] = 0;
        trailingOnesAndTotalCoeff[1][1] = 1;
        trailingOnesAndTotalCoeff[2][0] = 1;
        trailingOnesAndTotalCoeff[2][1] = 1;
        trailingOnesAndTotalCoeff[3][0] = 0;
        trailingOnesAndTotalCoeff[3][1] = 2;
        trailingOnesAndTotalCoeff[4][0] = 1;
        trailingOnesAndTotalCoeff[4][1] = 2;
        trailingOnesAndTotalCoeff[5][0] = 2;
        trailingOnesAndTotalCoeff[5][1] = 2;
        trailingOnesAndTotalCoeff[6][0] = 0;
        trailingOnesAndTotalCoeff[6][1] = 3;
        trailingOnesAndTotalCoeff[7][0] = 1;
        trailingOnesAndTotalCoeff[7][1] = 3;
        trailingOnesAndTotalCoeff[8][0] = 2;
        trailingOnesAndTotalCoeff[8][1] = 3;
        trailingOnesAndTotalCoeff[9][0] = 3;
        trailingOnesAndTotalCoeff[9][1] = 3;
        trailingOnesAndTotalCoeff[10][0] = 0;
        trailingOnesAndTotalCoeff[10][1] = 4;
        trailingOnesAndTotalCoeff[11][0] = 1;
        trailingOnesAndTotalCoeff[11][1] = 4;
        trailingOnesAndTotalCoeff[12][0] = 2;
        trailingOnesAndTotalCoeff[12][1] = 4;
        trailingOnesAndTotalCoeff[13][0] = 3;
        trailingOnesAndTotalCoeff[13][1] = 4;
        trailingOnesAndTotalCoeff[14][0] = 0;
        trailingOnesAndTotalCoeff[14][1] = 5;
        trailingOnesAndTotalCoeff[15][0] = 1;
        trailingOnesAndTotalCoeff[15][1] = 5;
        trailingOnesAndTotalCoeff[16][0] = 2;
        trailingOnesAndTotalCoeff[16][1] = 5;
        trailingOnesAndTotalCoeff[17][0] = 3;
        trailingOnesAndTotalCoeff[17][1] = 5;
        trailingOnesAndTotalCoeff[18][0] = 0;
```

```
trailingOnesAndTotalCoeff[18][1] = 6;
trailingOnesAndTotalCoeff[19][0] = 1;
trailingOnesAndTotalCoeff[19][1] = 6;
trailingOnesAndTotalCoeff[20][0] = 2;
trailingOnesAndTotalCoeff[20][1] = 6;
trailingOnesAndTotalCoeff[21][0] = 3;
trailingOnesAndTotalCoeff[21][1] = 6;
trailingOnesAndTotalCoeff[22][0] = 0;
trailingOnesAndTotalCoeff[22][1] = 7;
trailingOnesAndTotalCoeff[23][0] = 1;
trailingOnesAndTotalCoeff[23][1] = 7;
trailingOnesAndTotalCoeff[24][0] = 2;
trailingOnesAndTotalCoeff[24][1] = 7;
trailingOnesAndTotalCoeff[25][0] = 3;
trailingOnesAndTotalCoeff[25][1] = 7;
trailingOnesAndTotalCoeff[26][0] = 0;
trailingOnesAndTotalCoeff[26][1] = 8;
trailingOnesAndTotalCoeff[27][0] = 1;
trailingOnesAndTotalCoeff[27][1] = 8;
trailingOnesAndTotalCoeff[28][0] = 2;
trailingOnesAndTotalCoeff[28][1] = 8;
trailingOnesAndTotalCoeff[29][0] = 3;
trailingOnesAndTotalCoeff[29][1] = 8;
trailingOnesAndTotalCoeff[30][0] = 0;
trailingOnesAndTotalCoeff[30][1] = 9;
trailingOnesAndTotalCoeff[31][0] = 1;
trailingOnesAndTotalCoeff[31][1] = 9;
trailingOnesAndTotalCoeff[32][0] = 2;
trailingOnesAndTotalCoeff[32][1] = 9;
trailingOnesAndTotalCoeff[33][0] = 3;
trailingOnesAndTotalCoeff[33][1] = 9;
trailingOnesAndTotalCoeff[34][0] = 0;
trailingOnesAndTotalCoeff[34][1] = 10;
trailingOnesAndTotalCoeff[35][0] = 1;
trailingOnesAndTotalCoeff[35][1] = 10;
trailingOnesAndTotalCoeff[36][0] = 2;
trailingOnesAndTotalCoeff[36][1] = 10;
trailingOnesAndTotalCoeff[37][0] = 3;
trailingOnesAndTotalCoeff[37][1] = 10;
trailingOnesAndTotalCoeff[38][0] = 0;
trailingOnesAndTotalCoeff[38][1] = 11;
trailingOnesAndTotalCoeff[39][0] = 1;
trailingOnesAndTotalCoeff[39][1] = 11;
trailingOnesAndTotalCoeff[40][0] = 2;
trailingOnesAndTotalCoeff[40][1] = 11;
trailingOnesAndTotalCoeff[41][0] = 3;
trailingOnesAndTotalCoeff[41][1] = 11;
trailingOnesAndTotalCoeff[42][0] = 0;
trailingOnesAndTotalCoeff[42][1] = 12;
trailingOnesAndTotalCoeff[43][0] = 1;
trailingOnesAndTotalCoeff[43][1] = 12;
trailingOnesAndTotalCoeff[44][0] = 2;
trailingOnesAndTotalCoeff[44][1] = 12;
trailingOnesAndTotalCoeff[45][0] = 3;
trailingOnesAndTotalCoeff[45][1] = 12;
trailingOnesAndTotalCoeff[46][0] = 0;
trailingOnesAndTotalCoeff[46][1] = 13;
trailingOnesAndTotalCoeff[47][0] = 1;
trailingOnesAndTotalCoeff[47][1] = 13;
trailingOnesAndTotalCoeff[48][0] = 2;
trailingOnesAndTotalCoeff[48][1] = 13;
trailingOnesAndTotalCoeff[49][0] = 3;
trailingOnesAndTotalCoeff[49][1] = 13;
trailingOnesAndTotalCoeff[50][0] = 0;
trailingOnesAndTotalCoeff[50][1] = 14;
trailingOnesAndTotalCoeff[51][0] = 1;
trailingOnesAndTotalCoeff[51][1] = 14;
trailingOnesAndTotalCoeff[52][0] = 2;
trailingOnesAndTotalCoeff[52][1] = 14;
trailingOnesAndTotalCoeff[53][0] = 3;
trailingOnesAndTotalCoeff[53][1] = 14;
trailingOnesAndTotalCoeff[54][0] = 0;
trailingOnesAndTotalCoeff[54][1] = 15;
trailingOnesAndTotalCoeff[55][0] = 1;
trailingOnesAndTotalCoeff[55][1] = 15;
trailingOnesAndTotalCoeff[56][0] = 2;
trailingOnesAndTotalCoeff[56][1] = 15;
trailingOnesAndTotalCoeff[57][0] = 3;
trailingOnesAndTotalCoeff[57][1] = 15;
trailingOnesAndTotalCoeff[58][0] = 0;
```

```
        trailingOnesAndTotalCoeff[58][1] = 16;
        trailingOnesAndTotalCoeff[59][0] = 1;
        trailingOnesAndTotalCoeff[59][1] = 16;
        trailingOnesAndTotalCoeff[60][0] = 2;
        trailingOnesAndTotalCoeff[60][1] = 16;
        trailingOnesAndTotalCoeff[61][0] = 3;
        trailingOnesAndTotalCoeff[61][1] = 16;
}


public void createCoeffTokenFields()
{
        // nC == 0 || nC == 1
        ct_1_nC01 = new int[1][2];
        ct_2_nC01 = new int[1][2];
        ct_3_nC01 = new int[1][2];
        ct_5_nC01 = new int[1][2];
        ct_6_nC01 = new int[3][2];
        ct_7_nC01 = new int[2][2];
        ct_8_nC01 = new int[4][2];
        ct_9_nC01 = new int[4][2];
        ct_10_nC01 = new int[4][2];
        ct_11_nC01 = new int[4][2];
        ct_13_nC01 = new int[8][2];
        ct_14_nC01 = new int[8][2];
        ct_15_nC01 = new int[9][2];
        ct_16_nC01 = new int[12][2];

        // ct_coeffToken.length_nC.values[x][0] = coeffToken.value
        // ct_coeffToken.length_nC.values[x][1] = ITU H.264 Table 9-5 row index
        ct_1_nC01[0][0] = 1;
        ct_1_nC01[0][1] = 0;

        ct_2_nC01[0][0] = 1;
        ct_2_nC01[0][1] = 2;

        ct_3_nC01[0][0] = 1;
        ct_3_nC01[0][1] = 5;

        ct_5_nC01[0][0] = 3;
        ct_5_nC01[0][1] = 9;

        ct_6_nC01[0][0] = 5;
        ct_6_nC01[0][1] = 1;
        ct_6_nC01[1][0] = 4;
        ct_6_nC01[1][1] = 4;
        ct_6_nC01[2][0] = 3;
        ct_6_nC01[2][1] = 13;

        ct_7_nC01[0][0] = 5;
        ct_7_nC01[0][1] = 8;
        ct_7_nC01[1][0] = 4;
        ct_7_nC01[1][1] = 17;

        ct_8_nC01[0][0] = 6;
        ct_8_nC01[0][1] = 7;
        ct_8_nC01[1][0] = 5;
        ct_8_nC01[1][1] = 12;
        ct_8_nC01[2][0] = 4;
        ct_8_nC01[2][1] = 21;
        ct_8_nC01[3][0] = 7;
        ct_8_nC01[3][1] = 3;

        ct_9_nC01[0][0] = 7;
        ct_9_nC01[0][1] = 6;
        ct_9_nC01[1][0] = 6;
        ct_9_nC01[1][1] = 11;
        ct_9_nC01[2][0] = 5;
        ct_9_nC01[2][1] = 16;
        ct_9_nC01[3][0] = 4;
        ct_9_nC01[3][1] = 25;

        ct_10_nC01[0][0] = 7;
        ct_10_nC01[0][1] = 10;
        ct_10_nC01[1][0] = 6;
        ct_10_nC01[1][1] = 15;
        ct_10_nC01[2][0] = 5;
        ct_10_nC01[2][1] = 20;
        ct_10_nC01[3][0] = 4;
        ct_10_nC01[3][1] = 29;
```

```
ct_11_nC01[0][0] = 7;
ct_11_nC01[0][1] = 14;
ct_11_nC01[1][0] = 6;
ct_11_nC01[1][1] = 19;
ct_11_nC01[2][0] = 5;
ct_11_nC01[2][1] = 24;
ct_11_nC01[3][0] = 4;
ct_11_nC01[3][1] = 33;

ct_13_nC01[0][0] = 15;
ct_13_nC01[0][1] = 18;
ct_13_nC01[1][0] = 11;
ct_13_nC01[1][1] = 22;
ct_13_nC01[2][0] = 14;
ct_13_nC01[2][1] = 23;
ct_13_nC01[3][0] = 8;
ct_13_nC01[3][1] = 26;
ct_13_nC01[4][0] = 10;
ct_13_nC01[4][1] = 27;
ct_13_nC01[5][0] = 13;
ct_13_nC01[5][1] = 28;
ct_13_nC01[6][0] = 9;
ct_13_nC01[6][1] = 32;
ct_13_nC01[7][0] = 12;
ct_13_nC01[7][1] = 37;

ct_14_nC01[0][0] = 15;
ct_14_nC01[0][1] = 30;
ct_14_nC01[1][0] = 14;
ct_14_nC01[1][1] = 31;
ct_14_nC01[2][0] = 10;
ct_14_nC01[2][1] = 35;
ct_14_nC01[3][0] = 9;
ct_14_nC01[3][1] = 40;
ct_14_nC01[4][0] = 8;
ct_14_nC01[4][1] = 45;
ct_14_nC01[5][0] = 11;
ct_14_nC01[5][1] = 34;
ct_14_nC01[6][0] = 12;
ct_14_nC01[6][1] = 41;
ct_14_nC01[7][0] = 13;
ct_14_nC01[7][1] = 36;

ct_15_nC01[0][0] = 15;
ct_15_nC01[0][1] = 38;
ct_15_nC01[1][0] = 14;
ct_15_nC01[1][1] = 39;
ct_15_nC01[2][0] = 10;
ct_15_nC01[2][1] = 43;
ct_15_nC01[3][0] = 13;
ct_15_nC01[3][1] = 44;
ct_15_nC01[4][0] = 11;
ct_15_nC01[4][1] = 42;
ct_15_nC01[5][0] = 1;
ct_15_nC01[5][1] = 47;
ct_15_nC01[6][0] = 9;
ct_15_nC01[6][1] = 48;
ct_15_nC01[7][0] = 12;
ct_15_nC01[7][1] = 49;
ct_15_nC01[8][0] = 8;
ct_15_nC01[8][1] = 53;

ct_16_nC01[0][0] = 15;
ct_16_nC01[0][1] = 46;
ct_16_nC01[1][0] = 11;
ct_16_nC01[1][1] = 50;
ct_16_nC01[2][0] = 14;
ct_16_nC01[2][1] = 51;
ct_16_nC01[3][0] = 13;
ct_16_nC01[3][1] = 52;
ct_16_nC01[4][0] = 7;
ct_16_nC01[4][1] = 54;
ct_16_nC01[5][0] = 10;
ct_16_nC01[5][1] = 55;
ct_16_nC01[6][0] = 9;
ct_16_nC01[6][1] = 56;
ct_16_nC01[7][0] = 12;
ct_16_nC01[7][1] = 57;
ct_16_nC01[8][0] = 4;
```

```
ct_16_nC01[8][1] = 58;
ct_16_nC01[9][0] = 6;
ct_16_nC01[9][1] = 59;
ct_16_nC01[10][0] = 5;
ct_16_nC01[10][1] = 60;
ct_16_nC01[11][0] = 8;
ct_16_nC01[11][1] = 61;


// nC == 2 || nC == 3
ct_2_nC23 = new int[2][2];
ct_3_nC23 = new int[1][2];
ct_4_nC23 = new int[2][2];
ct_5_nC23 = new int[2][2];
ct_6_nC23 = new int[8][2];
ct_7_nC23 = new int[4][2];
ct_8_nC23 = new int[4][2];
ct_9_nC23 = new int[4][2];
ct_11_nC23 = new int[8][2];
ct_12_nC23 = new int[8][2];
ct_13_nC23 = new int[11][2];
ct_14_nC23 = new int[8][2];

// ct_coeffToken.length_nC.values[x][0] = coeffToken.value
// ct_coeffToken.length_nC.values[x][1] = ITU H.264 Table 9-5 row number
ct_2_nC23[0][0] = 3;
ct_2_nC23[0][1] = 0;
ct_2_nC23[1][0] = 2;
ct_2_nC23[1][1] = 2;

ct_3_nC23[0][0] = 3;
ct_3_nC23[0][1] = 5;

ct_4_nC23[0][0] = 5;
ct_4_nC23[0][1] = 9;
ct_4_nC23[1][0] = 4;
ct_4_nC23[1][1] = 13;

ct_5_nC23[0][0] = 7;
ct_5_nC23[0][1] = 4;
ct_5_nC23[1][0] = 6;
ct_5_nC23[1][1] = 17;

ct_6_nC23[0][0] = 11;
ct_6_nC23[0][1] = 1;
ct_6_nC23[1][0] = 7;
ct_6_nC23[1][1] = 3;
ct_6_nC23[2][0] = 10;
ct_6_nC23[2][1] = 7;
ct_6_nC23[3][0] = 9;
ct_6_nC23[3][1] = 8;
ct_6_nC23[4][0] = 6;
ct_6_nC23[4][1] = 11;
ct_6_nC23[5][0] = 5;
ct_6_nC23[5][1] = 12;
ct_6_nC23[6][0] = 8;
ct_6_nC23[6][1] = 21;
ct_6_nC23[7][0] = 4;
ct_6_nC23[7][1] = 25;

ct_7_nC23[0][0] = 7;
ct_7_nC23[0][1] = 6;
ct_7_nC23[1][0] = 6;
ct_7_nC23[1][1] = 15;
ct_7_nC23[2][0] = 5;
ct_7_nC23[2][1] = 16;
ct_7_nC23[3][0] = 4;
ct_7_nC23[3][1] = 29;

ct_8_nC23[0][0] = 7;
ct_8_nC23[0][1] = 10;
ct_8_nC23[1][0] = 4;
ct_8_nC23[1][1] = 14;
ct_8_nC23[2][0] = 6;
ct_8_nC23[2][1] = 19;
ct_8_nC23[3][0] = 5;
ct_8_nC23[3][1] = 20;

ct_9_nC23[0][0] = 7;
ct_9_nC23[0][1] = 18;
```

```
ct_9_nC23[1][0] = 6;
ct_9_nC23[1][1] = 23;
ct_9_nC23[2][0] = 5;
ct_9_nC23[2][1] = 24;
ct_9_nC23[3][0] = 4;
ct_9_nC23[3][1] = 33;

ct_11_nC23[0][0] = 15;
ct_11_nC23[0][1] = 22;
ct_11_nC23[1][0] = 11;
ct_11_nC23[1][1] = 26;
ct_11_nC23[2][0] = 14;
ct_11_nC23[2][1] = 27;
ct_11_nC23[3][0] = 13;
ct_11_nC23[3][1] = 28;
ct_11_nC23[4][0] = 10;
ct_11_nC23[4][1] = 31;
ct_11_nC23[5][0] = 9;
ct_11_nC23[5][1] = 32;
ct_11_nC23[6][0] = 12;
ct_11_nC23[6][1] = 37;
ct_11_nC23[7][0] = 8;
ct_11_nC23[7][1] = 41;

ct_12_nC23[0][0] = 15;
ct_12_nC23[0][1] = 30;
ct_12_nC23[1][0] = 11;
ct_12_nC23[1][1] = 34;
ct_12_nC23[2][0] = 14;
ct_12_nC23[2][1] = 35;
ct_12_nC23[3][0] = 13;
ct_12_nC23[3][1] = 36;
ct_12_nC23[4][0] = 8;
ct_12_nC23[4][1] = 38;
ct_12_nC23[5][0] = 10;
ct_12_nC23[5][1] = 39;
ct_12_nC23[6][0] = 9;
ct_12_nC23[6][1] = 40;
ct_12_nC23[7][0] = 12;
ct_12_nC23[7][1] = 45;

ct_13_nC23[0][0] = 15;
ct_13_nC23[0][1] = 42;
ct_13_nC23[1][0] = 14;
ct_13_nC23[1][1] = 43;
ct_13_nC23[2][0] = 13;
ct_13_nC23[2][1] = 44;
ct_13_nC23[3][0] = 11;
ct_13_nC23[3][1] = 46;
ct_13_nC23[4][0] = 10;
ct_13_nC23[4][1] = 47;
ct_13_nC23[5][0] = 9;
ct_13_nC23[5][1] = 48;
ct_13_nC23[6][0] = 12;
ct_13_nC23[6][1] = 49;
ct_13_nC23[7][0] = 7;
ct_13_nC23[7][1] = 50;
ct_13_nC23[8][0] = 6;
ct_13_nC23[8][1] = 52;
ct_13_nC23[9][0] = 8;
ct_13_nC23[9][1] = 53;
ct_13_nC23[10][0] = 1;
ct_13_nC23[10][1] = 57;

ct_14_nC23[0][0] = 11;
ct_14_nC23[0][1] = 51;
ct_14_nC23[1][0] = 9;
ct_14_nC23[1][1] = 54;
ct_14_nC23[2][0] = 8;
ct_14_nC23[2][1] = 55;
ct_14_nC23[3][0] = 10;
ct_14_nC23[3][1] = 56;
ct_14_nC23[4][0] = 7;
ct_14_nC23[4][1] = 58;
ct_14_nC23[5][0] = 6;
ct_14_nC23[5][1] = 59;
ct_14_nC23[6][0] = 5;
ct_14_nC23[6][1] = 60;
ct_14_nC23[7][0] = 4;
ct_14_nC23[7][1] = 61;
```

```
// nC == 4 || nC == 5 || nC == 6 || nC == 7
ct_4_nC4567 = new int[8][2];
ct_5_nC4567 = new int[8][2];
ct_6_nC4567 = new int[8][2];
ct_7_nC4567 = new int[8][2];
ct_8_nC4567 = new int[8][2];
ct_9_nC4567 = new int[9][2];
ct_10_nC4567 = new int[13][2];

// ct_coeffToken.length_nC.values[x][0] = coeffToken.value
// ct_coeffToken.length_nC.values[x][1] = ITU H.264 Table 9-5 row number
ct_4_nC4567[0][0] = 15;
ct_4_nC4567[0][1] = 0;
ct_4_nC4567[1][0] = 14;
ct_4_nC4567[1][1] = 2;
ct_4_nC4567[2][0] = 13;
ct_4_nC4567[2][1] = 5;
ct_4_nC4567[3][0] = 12;
ct_4_nC4567[3][1] = 9;
ct_4_nC4567[4][0] = 11;
ct_4_nC4567[4][1] = 13;
ct_4_nC4567[5][0] = 10;
ct_4_nC4567[5][1] = 17;
ct_4_nC4567[6][0] = 9;
ct_4_nC4567[6][1] = 21;
ct_4_nC4567[7][0] = 8;
ct_4_nC4567[7][1] = 25;

ct_5_nC4567[0][0] = 15;
ct_5_nC4567[0][1] = 4;
ct_5_nC4567[1][0] = 12;
ct_5_nC4567[1][1] = 7;
ct_5_nC4567[2][0] = 14;
ct_5_nC4567[2][1] = 8;
ct_5_nC4567[3][0] = 10;
ct_5_nC4567[3][1] = 11;
ct_5_nC4567[4][0] = 11;
ct_5_nC4567[4][1] = 12;
ct_5_nC4567[5][0] = 8;
ct_5_nC4567[5][1] = 15;
ct_5_nC4567[6][0] = 9;
ct_5_nC4567[6][1] = 16;
ct_5_nC4567[7][0] = 13;
ct_5_nC4567[7][1] = 29;

ct_6_nC4567[0][0] = 15;
ct_6_nC4567[0][1] = 1;
ct_6_nC4567[1][0] = 11;
ct_6_nC4567[1][1] = 3;
ct_6_nC4567[2][0] = 8;
ct_6_nC4567[2][1] = 6;
ct_6_nC4567[3][0] = 14;
ct_6_nC4567[3][1] = 19;
ct_6_nC4567[4][0] = 13;
ct_6_nC4567[4][1] = 20;
ct_6_nC4567[5][0] = 10;
ct_6_nC4567[5][1] = 23;
ct_6_nC4567[6][0] = 9;
ct_6_nC4567[6][1] = 24;
ct_6_nC4567[7][0] = 12;
ct_6_nC4567[7][1] = 33;

ct_7_nC4567[0][0] = 15;
ct_7_nC4567[0][1] = 10;
ct_7_nC4567[1][0] = 11;
ct_7_nC4567[1][1] = 14;
ct_7_nC4567[2][0] = 9;
ct_7_nC4567[2][1] = 18;
ct_7_nC4567[3][0] = 8;
ct_7_nC4567[3][1] = 22;
ct_7_nC4567[4][0] = 14;
ct_7_nC4567[4][1] = 27;
ct_7_nC4567[5][0] = 13;
ct_7_nC4567[5][1] = 28;
ct_7_nC4567[6][0] = 10;
ct_7_nC4567[6][1] = 32;
ct_7_nC4567[7][0] = 12;
ct_7_nC4567[7][1] = 37;
```

```
ct_8_nC4567[0][0] = 11;
ct_8_nC4567[0][1] = 30;
ct_8_nC4567[1][0] = 14;
ct_8_nC4567[1][1] = 31;
ct_8_nC4567[2][0] = 15;
ct_8_nC4567[2][1] = 26;
ct_8_nC4567[3][0] = 10;
ct_8_nC4567[3][1] = 35;
ct_8_nC4567[4][0] = 13;
ct_8_nC4567[4][1] = 36;
ct_8_nC4567[5][0] = 9;
ct_8_nC4567[5][1] = 40;
ct_8_nC4567[6][0] = 12;
ct_8_nC4567[6][1] = 41;
ct_8_nC4567[7][0] = 8;
ct_8_nC4567[7][1] = 45;

ct_9_nC4567[0][0] = 15;
ct_9_nC4567[0][1] = 34;
ct_9_nC4567[1][0] = 11;
ct_9_nC4567[1][1] = 38;
ct_9_nC4567[2][0] = 8;
ct_9_nC4567[2][1] = 42;
ct_9_nC4567[3][0] = 10;
ct_9_nC4567[3][1] = 43;
ct_9_nC4567[4][0] = 13;
ct_9_nC4567[4][1] = 44;
ct_9_nC4567[5][0] = 7;
ct_9_nC4567[5][1] = 47;
ct_9_nC4567[6][0] = 9;
ct_9_nC4567[6][1] = 48;
ct_9_nC4567[7][0] = 12;
ct_9_nC4567[7][1] = 49;
ct_9_nC4567[8][0] = 14;
ct_9_nC4567[8][1] = 39;

ct_10_nC4567[0][0] = 13;
ct_10_nC4567[0][1] = 46;
ct_10_nC4567[1][0] = 9;
ct_10_nC4567[1][1] = 50;
ct_10_nC4567[2][0] = 12;
ct_10_nC4567[2][1] = 51;
ct_10_nC4567[3][0] = 11;
ct_10_nC4567[3][1] = 52;
ct_10_nC4567[4][0] = 10;
ct_10_nC4567[4][1] = 53;
ct_10_nC4567[5][0] = 5;
ct_10_nC4567[5][1] = 54;
ct_10_nC4567[6][0] = 8;
ct_10_nC4567[6][1] = 55;
ct_10_nC4567[7][0] = 7;
ct_10_nC4567[7][1] = 56;
ct_10_nC4567[8][0] = 6;
ct_10_nC4567[8][1] = 57;
ct_10_nC4567[9][0] = 1;
ct_10_nC4567[9][1] = 58;
ct_10_nC4567[10][0] = 4;
ct_10_nC4567[10][1] = 59;
ct_10_nC4567[11][0] = 3;
ct_10_nC4567[11][1] = 60;
ct_10_nC4567[12][0] = 2;
ct_10_nC4567[12][1] = 61;


// nC >= 8
ct_6_nCgt8 = new int[62][2];

// ct_coeffToken.length_nC.values[x][0] = coeffToken.value
// ct_coeffToken.length_nC.values[x][1] = ITU H.264 Table 9-5 row number
ct_6_nCgt8[0][0] = 3;
ct_6_nCgt8[0][1] = 0;
ct_6_nCgt8[1][0] = 0;
ct_6_nCgt8[1][1] = 1;
ct_6_nCgt8[2][0] = 1;
ct_6_nCgt8[2][1] = 2;
ct_6_nCgt8[3][0] = 4;
ct_6_nCgt8[3][1] = 3;
ct_6_nCgt8[4][0] = 5;
ct_6_nCgt8[4][1] = 4;
```

```
ct_6_nCgt8[5][0] = 6;
ct_6_nCgt8[5][1] = 5;
ct_6_nCgt8[6][0] = 8;
ct_6_nCgt8[6][1] = 6;
ct_6_nCgt8[7][0] = 9;
ct_6_nCgt8[7][1] = 7;
ct_6_nCgt8[8][0] = 10;
ct_6_nCgt8[8][1] = 8;
ct_6_nCgt8[9][0] = 11;
ct_6_nCgt8[9][1] = 9;
ct_6_nCgt8[10][0] = 12;
ct_6_nCgt8[10][1] = 10;
ct_6_nCgt8[11][0] = 13;
ct_6_nCgt8[11][1] = 11;
ct_6_nCgt8[12][0] = 14;
ct_6_nCgt8[12][1] = 12;
ct_6_nCgt8[13][0] = 15;
ct_6_nCgt8[13][1] = 13;
ct_6_nCgt8[14][0] = 16;
ct_6_nCgt8[14][1] = 14;
ct_6_nCgt8[15][0] = 17;
ct_6_nCgt8[15][1] = 15;
ct_6_nCgt8[16][0] = 18;
ct_6_nCgt8[16][1] = 16;
ct_6_nCgt8[17][0] = 19;
ct_6_nCgt8[17][1] = 17;
ct_6_nCgt8[18][0] = 20;
ct_6_nCgt8[18][1] = 18;
ct_6_nCgt8[19][0] = 21;
ct_6_nCgt8[19][1] = 19;
ct_6_nCgt8[20][0] = 22;
ct_6_nCgt8[20][1] = 20;
ct_6_nCgt8[21][0] = 23;
ct_6_nCgt8[21][1] = 21;
ct_6_nCgt8[22][0] = 24;
ct_6_nCgt8[22][1] = 22;
ct_6_nCgt8[23][0] = 25;
ct_6_nCgt8[23][1] = 23;
ct_6_nCgt8[24][0] = 26;
ct_6_nCgt8[24][1] = 24;
ct_6_nCgt8[25][0] = 27;
ct_6_nCgt8[25][1] = 25;
ct_6_nCgt8[26][0] = 28;
ct_6_nCgt8[26][1] = 26;
ct_6_nCgt8[27][0] = 29;
ct_6_nCgt8[27][1] = 27;
ct_6_nCgt8[28][0] = 30;
ct_6_nCgt8[28][1] = 28;
ct_6_nCgt8[29][0] = 31;
ct_6_nCgt8[29][1] = 29;
ct_6_nCgt8[30][0] = 32;
ct_6_nCgt8[30][1] = 30;
ct_6_nCgt8[31][0] = 33;
ct_6_nCgt8[31][1] = 31;
ct_6_nCgt8[32][0] = 34;
ct_6_nCgt8[32][1] = 32;
ct_6_nCgt8[33][0] = 35;
ct_6_nCgt8[33][1] = 33;
ct_6_nCgt8[34][0] = 36;
ct_6_nCgt8[34][1] = 34;
ct_6_nCgt8[35][0] = 37;
ct_6_nCgt8[35][1] = 35;
ct_6_nCgt8[36][0] = 38;
ct_6_nCgt8[36][1] = 36;
ct_6_nCgt8[37][0] = 39;
ct_6_nCgt8[37][1] = 37;
ct_6_nCgt8[38][0] = 40;
ct_6_nCgt8[38][1] = 38;
ct_6_nCgt8[39][0] = 41;
ct_6_nCgt8[39][1] = 39;
ct_6_nCgt8[40][0] = 42;
ct_6_nCgt8[40][1] = 40;
ct_6_nCgt8[41][0] = 43;
ct_6_nCgt8[41][1] = 41;
ct_6_nCgt8[42][0] = 44;
ct_6_nCgt8[42][1] = 42;
ct_6_nCgt8[43][0] = 45;
ct_6_nCgt8[43][1] = 43;
ct_6_nCgt8[44][0] = 46;
ct_6_nCgt8[44][1] = 44;
```

```
ct_6_nCgt8[45][0] = 47;
ct_6_nCgt8[45][1] = 45;
ct_6_nCgt8[46][0] = 48;
ct_6_nCgt8[46][1] = 46;
ct_6_nCgt8[47][0] = 49;
ct_6_nCgt8[47][1] = 47;
ct_6_nCgt8[48][0] = 50;
ct_6_nCgt8[48][1] = 48;
ct_6_nCgt8[49][0] = 51;
ct_6_nCgt8[49][1] = 49;
ct_6_nCgt8[50][0] = 52;
ct_6_nCgt8[50][1] = 50;
ct_6_nCgt8[51][0] = 53;
ct_6_nCgt8[51][1] = 51;
ct_6_nCgt8[52][0] = 54;
ct_6_nCgt8[52][1] = 52;
ct_6_nCgt8[53][0] = 55;
ct_6_nCgt8[53][1] = 53;
ct_6_nCgt8[54][0] = 56;
ct_6_nCgt8[54][1] = 54;
ct_6_nCgt8[55][0] = 57;
ct_6_nCgt8[55][1] = 55;
ct_6_nCgt8[56][0] = 58;
ct_6_nCgt8[56][1] = 56;
ct_6_nCgt8[57][0] = 59;
ct_6_nCgt8[57][1] = 57;
ct_6_nCgt8[58][0] = 60;
ct_6_nCgt8[58][1] = 58;
ct_6_nCgt8[59][0] = 61;
ct_6_nCgt8[59][1] = 59;
ct_6_nCgt8[60][0] = 62;
ct_6_nCgt8[60][1] = 60;
ct_6_nCgt8[61][0] = 63;
ct_6_nCgt8[61][1] = 61;


// nC == -1
ct_1_nCminus1 = new int[1][2];
ct_2_nCminus1 = new int[1][2];
ct_3_nCminus1 = new int[1][2];
ct_6_nCminus1 = new int[6][6];
ct_7_nCminus1 = new int[3][2];
ct_8_nCminus1 = new int[2][2];

// ct_coeffToken.length_nC.values[x][0] = coeffToken.value
// ct_coeffToken.length_nC.values[x][1] = ITU H.264 Table 9-5 row number
ct_1_nCminus1[0][0] = 1;
ct_1_nCminus1[0][1] = 2;

ct_2_nCminus1[0][0] = 1;
ct_2_nCminus1[0][1] = 0;

ct_3_nCminus1[0][0] = 1;
ct_3_nCminus1[0][1] = 5;

ct_6_nCminus1[0][0] = 7;
ct_6_nCminus1[0][1] = 1;
ct_6_nCminus1[1][0] = 4;
ct_6_nCminus1[1][1] = 3;
ct_6_nCminus1[2][0] = 6;
ct_6_nCminus1[2][1] = 4;
ct_6_nCminus1[3][0] = 3;
ct_6_nCminus1[3][1] = 6;
ct_6_nCminus1[4][0] = 5;
ct_6_nCminus1[4][1] = 9;
ct_6_nCminus1[5][0] = 2;
ct_6_nCminus1[5][1] = 10;

ct_7_nCminus1[0][0] = 3;
ct_7_nCminus1[0][1] = 7;
ct_7_nCminus1[1][0] = 2;
ct_7_nCminus1[1][1] = 8;
ct_7_nCminus1[2][0] = 0;
ct_7_nCminus1[2][1] = 13;

ct_8_nCminus1[0][0] = 3;
ct_8_nCminus1[0][1] = 11;
ct_8_nCminus1[1][0] = 2;
ct_8_nCminus1[1][1] = 12;
```

```
// nC == -2
ct_1_nCminus2 = new int[1][2];
ct_2_nCminus2 = new int[1][2];
ct_3_nCminus2 = new int[1][2];
ct_5_nCminus2 = new int[1][2];
ct_6_nCminus2 = new int[1][2];
ct_7_nCminus2 = new int[8][2];
ct_9_nCminus2 = new int[4][2];
ct_10_nCminus2 = new int[4][2];
ct_11_nCminus2 = new int[4][2];
ct_12_nCminus2 = new int[4][2];
ct_13_nCminus2 = new int[1][2];

// ct_coeffToken.length_nC.values[x][0] = coeffToken.value
// ct_coeffToken.length_nC.values[x][1] = ITU H.264 Table 9-5 row number
ct_1_nCminus2[0][0] = 1;
ct_1_nCminus2[0][1] = 0;

ct_2_nCminus2[0][0] = 1;
ct_2_nCminus2[0][1] = 2;

ct_3_nCminus2[0][0] = 1;
ct_3_nCminus2[0][1] = 5;

ct_5_nCminus2[0][0] = 1;
ct_5_nCminus2[0][1] = 9;

ct_6_nCminus2[0][0] = 1;
ct_6_nCminus2[0][1] = 13;

ct_7_nCminus2[0][0] = 15;
ct_7_nCminus2[0][1] = 1;
ct_7_nCminus2[1][0] = 14;
ct_7_nCminus2[1][1] = 3;
ct_7_nCminus2[2][0] = 13;
ct_7_nCminus2[2][1] = 4;
ct_7_nCminus2[3][0] = 12;
ct_7_nCminus2[3][1] = 7;
ct_7_nCminus2[4][0] = 11;
ct_7_nCminus2[4][1] = 8;
ct_7_nCminus2[5][0] = 10;
ct_7_nCminus2[5][1] = 12;
ct_7_nCminus2[6][0] = 9;
ct_7_nCminus2[6][1] = 17;
ct_7_nCminus2[7][0] = 8;
ct_7_nCminus2[7][1] = 21;

ct_9_nCminus2[0][0] = 7;
ct_9_nCminus2[0][1] = 6;
ct_9_nCminus2[1][0] = 6;
ct_9_nCminus2[1][1] = 10;
ct_9_nCminus2[2][0] = 5;
ct_9_nCminus2[2][1] = 11;
ct_9_nCminus2[3][0] = 4;
ct_9_nCminus2[3][1] = 16;

ct_10_nCminus2[0][0] = 7;
ct_10_nCminus2[0][1] = 14;
ct_10_nCminus2[1][0] = 6;
ct_10_nCminus2[1][1] = 15;
ct_10_nCminus2[2][0] = 5;
ct_10_nCminus2[2][1] = 20;
ct_10_nCminus2[3][0] = 4;
ct_10_nCminus2[3][1] = 25;

ct_11_nCminus2[0][0] = 7;
ct_11_nCminus2[0][1] = 18;
ct_11_nCminus2[1][0] = 6;
ct_11_nCminus2[1][1] = 19;
ct_11_nCminus2[2][0] = 5;
ct_11_nCminus2[2][1] = 24;
ct_11_nCminus2[3][0] = 4;
ct_11_nCminus2[3][1] = 29;

ct_12_nCminus2[0][0] = 7;
ct_12_nCminus2[0][1] = 22;
ct_12_nCminus2[1][0] = 6;
ct_12_nCminus2[1][1] = 23;
ct_12_nCminus2[2][0] = 5;
```

```
        ct_12_nCminus2[2][1] = 27;
        ct_12_nCminus2[3][0] = 4;
        ct_12_nCminus2[3][1] = 28;

        ct_13_nCminus2[0][0] = 7;
        ct_13_nCminus2[0][1] = 26;
}


private void createTotalZerosFields()
{
        // luma || chroma(4:4:4)
        tz_1_maxNumCoeffsX = new int[16][2];
        tz_2_maxNumCoeffsX = new int[15][2];
        tz_3_maxNumCoeffsX = new int[14][2];
        tz_4_maxNumCoeffsX = new int[13][2];
        tz_5_maxNumCoeffsX = new int[12][2];
        tz_6_maxNumCoeffsX = new int[11][2];
        tz_7_maxNumCoeffsX = new int[10][2];
        tz_8_maxNumCoeffsX = new int[9][2];
        tz_9_maxNumCoeffsX = new int[8][2];
        tz_10_maxNumCoeffsX = new int[7][2];
        tz_11_maxNumCoeffsX = new int[6][2];
        tz_12_maxNumCoeffsX = new int[5][2];
        tz_13_maxNumCoeffsX = new int[4][2];
        tz_14_maxNumCoeffsX = new int[3][2];
        tz_15_maxNumCoeffsX = new int[2][2];

        // tz_1_maxNumCoeffsX[x][0] = totalZeros.value
        // tz_1_maxNumCoeffsX[x][1] = totalZeros.length
        tz_1_maxNumCoeffsX[0][0] = 1;
        tz_1_maxNumCoeffsX[0][1] = 1;
        tz_1_maxNumCoeffsX[1][0] = 3;
        tz_1_maxNumCoeffsX[1][1] = 3;
        tz_1_maxNumCoeffsX[2][0] = 2;
        tz_1_maxNumCoeffsX[2][1] = 3;
        tz_1_maxNumCoeffsX[3][0] = 3;
        tz_1_maxNumCoeffsX[3][1] = 4;
        tz_1_maxNumCoeffsX[4][0] = 2;
        tz_1_maxNumCoeffsX[4][1] = 4;
        tz_1_maxNumCoeffsX[5][0] = 3;
        tz_1_maxNumCoeffsX[5][1] = 5;
        tz_1_maxNumCoeffsX[6][0] = 2;
        tz_1_maxNumCoeffsX[6][1] = 5;
        tz_1_maxNumCoeffsX[7][0] = 3;
        tz_1_maxNumCoeffsX[7][1] = 6;
        tz_1_maxNumCoeffsX[8][0] = 2;
        tz_1_maxNumCoeffsX[8][1] = 6;
        tz_1_maxNumCoeffsX[9][0] = 3;
        tz_1_maxNumCoeffsX[9][1] = 7;
        tz_1_maxNumCoeffsX[10][0] = 2;
        tz_1_maxNumCoeffsX[10][1] = 7;
        tz_1_maxNumCoeffsX[11][0] = 3;
        tz_1_maxNumCoeffsX[11][1] = 8;
        tz_1_maxNumCoeffsX[12][0] = 2;
        tz_1_maxNumCoeffsX[12][1] = 8;
        tz_1_maxNumCoeffsX[13][0] = 3;
        tz_1_maxNumCoeffsX[13][1] = 9;
        tz_1_maxNumCoeffsX[14][0] = 2;
        tz_1_maxNumCoeffsX[14][1] = 9;
        tz_1_maxNumCoeffsX[15][0] = 1;
        tz_1_maxNumCoeffsX[15][1] = 9;

        // tz_2_maxNumCoeffsX[x][0] = totalZeros.value
        // tz_2_maxNumCoeffsX[x][1] = totalZeros.length
        tz_2_maxNumCoeffsX[0][0] = 7;
        tz_2_maxNumCoeffsX[0][1] = 3;
        tz_2_maxNumCoeffsX[1][0] = 6;
        tz_2_maxNumCoeffsX[1][1] = 3;
        tz_2_maxNumCoeffsX[2][0] = 5;
        tz_2_maxNumCoeffsX[2][1] = 3;
        tz_2_maxNumCoeffsX[3][0] = 4;
        tz_2_maxNumCoeffsX[3][1] = 3;
        tz_2_maxNumCoeffsX[4][0] = 3;
        tz_2_maxNumCoeffsX[4][1] = 3;
        tz_2_maxNumCoeffsX[5][0] = 5;
        tz_2_maxNumCoeffsX[5][1] = 4;
        tz_2_maxNumCoeffsX[6][0] = 4;
        tz_2_maxNumCoeffsX[6][1] = 4;
        tz_2_maxNumCoeffsX[7][0] = 3;
```

```
tz_2_maxNumCoeffsX[7][1] = 4;
tz_2_maxNumCoeffsX[8][0] = 2;
tz_2_maxNumCoeffsX[8][1] = 4;
tz_2_maxNumCoeffsX[9][0] = 3;
tz_2_maxNumCoeffsX[9][1] = 5;
tz_2_maxNumCoeffsX[10][0] = 2;
tz_2_maxNumCoeffsX[10][1] = 5;
tz_2_maxNumCoeffsX[11][0] = 3;
tz_2_maxNumCoeffsX[11][1] = 6;
tz_2_maxNumCoeffsX[12][0] = 2;
tz_2_maxNumCoeffsX[12][1] = 6;
tz_2_maxNumCoeffsX[13][0] = 1;
tz_2_maxNumCoeffsX[13][1] = 6;
tz_2_maxNumCoeffsX[14][0] = 0;
tz_2_maxNumCoeffsX[14][1] = 6;

// tz_3_maxNumCoeffsX[x][0] = totalZeros.value
// tz_3_maxNumCoeffsX[x][1] = totalZeros.length
tz_3_maxNumCoeffsX[0][0] = 5;
tz_3_maxNumCoeffsX[0][1] = 4;
tz_3_maxNumCoeffsX[1][0] = 7;
tz_3_maxNumCoeffsX[1][1] = 3;
tz_3_maxNumCoeffsX[2][0] = 6;
tz_3_maxNumCoeffsX[2][1] = 3;
tz_3_maxNumCoeffsX[3][0] = 5;
tz_3_maxNumCoeffsX[3][1] = 3;
tz_3_maxNumCoeffsX[4][0] = 4;
tz_3_maxNumCoeffsX[4][1] = 4;
tz_3_maxNumCoeffsX[5][0] = 3;
tz_3_maxNumCoeffsX[5][1] = 4;
tz_3_maxNumCoeffsX[6][0] = 4;
tz_3_maxNumCoeffsX[6][1] = 3;
tz_3_maxNumCoeffsX[7][0] = 3;
tz_3_maxNumCoeffsX[7][1] = 3;
tz_3_maxNumCoeffsX[8][0] = 2;
tz_3_maxNumCoeffsX[8][1] = 4;
tz_3_maxNumCoeffsX[9][0] = 3;
tz_3_maxNumCoeffsX[9][1] = 5;
tz_3_maxNumCoeffsX[10][0] = 2;
tz_3_maxNumCoeffsX[10][1] = 5;
tz_3_maxNumCoeffsX[11][0] = 1;
tz_3_maxNumCoeffsX[11][1] = 6;
tz_3_maxNumCoeffsX[12][0] = 1;
tz_3_maxNumCoeffsX[12][1] = 5;
tz_3_maxNumCoeffsX[13][0] = 0;
tz_3_maxNumCoeffsX[13][1] = 6;

// tz_4_maxNumCoeffsX[x][0] = totalZeros.value
// tz_4_maxNumCoeffsX[x][1] = totalZeros.length
tz_4_maxNumCoeffsX[0][0] = 3;
tz_4_maxNumCoeffsX[0][1] = 5;
tz_4_maxNumCoeffsX[1][0] = 7;
tz_4_maxNumCoeffsX[1][1] = 3;
tz_4_maxNumCoeffsX[2][0] = 5;
tz_4_maxNumCoeffsX[2][1] = 4;
tz_4_maxNumCoeffsX[3][0] = 4;
tz_4_maxNumCoeffsX[3][1] = 4;
tz_4_maxNumCoeffsX[4][0] = 6;
tz_4_maxNumCoeffsX[4][1] = 3;
tz_4_maxNumCoeffsX[5][0] = 5;
tz_4_maxNumCoeffsX[5][1] = 3;
tz_4_maxNumCoeffsX[6][0] = 4;
tz_4_maxNumCoeffsX[6][1] = 3;
tz_4_maxNumCoeffsX[7][0] = 3;
tz_4_maxNumCoeffsX[7][1] = 4;
tz_4_maxNumCoeffsX[8][0] = 3;
tz_4_maxNumCoeffsX[8][1] = 3;
tz_4_maxNumCoeffsX[9][0] = 2;
tz_4_maxNumCoeffsX[9][1] = 4;
tz_4_maxNumCoeffsX[10][0] = 2;
tz_4_maxNumCoeffsX[10][1] = 5;
tz_4_maxNumCoeffsX[11][0] = 1;
tz_4_maxNumCoeffsX[11][1] = 5;
tz_4_maxNumCoeffsX[12][0] = 0;
tz_4_maxNumCoeffsX[12][1] = 5;

// tz_5_maxNumCoeffsX[x][0] = totalZeros.value
// tz_5_maxNumCoeffsX[x][1] = totalZeros.length
tz_5_maxNumCoeffsX[0][0] = 5;
tz_5_maxNumCoeffsX[0][1] = 4;
```

```
tz_5_maxNumCoeffsX[1][0] = 4;
tz_5_maxNumCoeffsX[1][1] = 4;
tz_5_maxNumCoeffsX[2][0] = 3;
tz_5_maxNumCoeffsX[2][1] = 4;
tz_5_maxNumCoeffsX[3][0] = 7;
tz_5_maxNumCoeffsX[3][1] = 3;
tz_5_maxNumCoeffsX[4][0] = 6;
tz_5_maxNumCoeffsX[4][1] = 3;
tz_5_maxNumCoeffsX[5][0] = 5;
tz_5_maxNumCoeffsX[5][1] = 3;
tz_5_maxNumCoeffsX[6][0] = 4;
tz_5_maxNumCoeffsX[6][1] = 3;
tz_5_maxNumCoeffsX[7][0] = 3;
tz_5_maxNumCoeffsX[7][1] = 3;
tz_5_maxNumCoeffsX[8][0] = 2;
tz_5_maxNumCoeffsX[8][1] = 4;
tz_5_maxNumCoeffsX[9][0] = 1;
tz_5_maxNumCoeffsX[9][1] = 5;
tz_5_maxNumCoeffsX[10][0] = 1;
tz_5_maxNumCoeffsX[10][1] = 4;
tz_5_maxNumCoeffsX[11][0] = 0;
tz_5_maxNumCoeffsX[11][1] = 5;

// tz_6_maxNumCoeffsX[x][0] = totalZeros.value
// tz_6_maxNumCoeffsX[x][1] = totalZeros.length
tz_6_maxNumCoeffsX[0][0] = 1;
tz_6_maxNumCoeffsX[0][1] = 6;
tz_6_maxNumCoeffsX[1][0] = 1;
tz_6_maxNumCoeffsX[1][1] = 5;
tz_6_maxNumCoeffsX[2][0] = 7;
tz_6_maxNumCoeffsX[2][1] = 3;
tz_6_maxNumCoeffsX[3][0] = 6;
tz_6_maxNumCoeffsX[3][1] = 3;
tz_6_maxNumCoeffsX[4][0] = 5;
tz_6_maxNumCoeffsX[4][1] = 3;
tz_6_maxNumCoeffsX[5][0] = 4;
tz_6_maxNumCoeffsX[5][1] = 3;
tz_6_maxNumCoeffsX[6][0] = 3;
tz_6_maxNumCoeffsX[6][1] = 3;
tz_6_maxNumCoeffsX[7][0] = 2;
tz_6_maxNumCoeffsX[7][1] = 3;
tz_6_maxNumCoeffsX[8][0] = 1;
tz_6_maxNumCoeffsX[8][1] = 4;
tz_6_maxNumCoeffsX[9][0] = 1;
tz_6_maxNumCoeffsX[9][1] = 3;
tz_6_maxNumCoeffsX[10][0] = 0;
tz_6_maxNumCoeffsX[10][1] = 6;

// tz_7_maxNumCoeffsX[x][0] = totalZeros.value
// tz_7_maxNumCoeffsX[x][1] = totalZeros.length
tz_7_maxNumCoeffsX[0][0] = 1;
tz_7_maxNumCoeffsX[0][1] = 6;
tz_7_maxNumCoeffsX[1][0] = 1;
tz_7_maxNumCoeffsX[1][1] = 5;
tz_7_maxNumCoeffsX[2][0] = 5;
tz_7_maxNumCoeffsX[2][1] = 3;
tz_7_maxNumCoeffsX[3][0] = 4;
tz_7_maxNumCoeffsX[3][1] = 3;
tz_7_maxNumCoeffsX[4][0] = 3;
tz_7_maxNumCoeffsX[4][1] = 3;
tz_7_maxNumCoeffsX[5][0] = 3;
tz_7_maxNumCoeffsX[5][1] = 2;
tz_7_maxNumCoeffsX[6][0] = 2;
tz_7_maxNumCoeffsX[6][1] = 3;
tz_7_maxNumCoeffsX[7][0] = 1;
tz_7_maxNumCoeffsX[7][1] = 4;
tz_7_maxNumCoeffsX[8][0] = 1;
tz_7_maxNumCoeffsX[8][1] = 3;
tz_7_maxNumCoeffsX[9][0] = 0;
tz_7_maxNumCoeffsX[9][1] = 6;

// tz_8_maxNumCoeffsX[x][0] = totalZeros.value
// tz_8_maxNumCoeffsX[x][1] = totalZeros.length
tz_8_maxNumCoeffsX[0][0] = 1;
tz_8_maxNumCoeffsX[0][1] = 6;
tz_8_maxNumCoeffsX[1][0] = 1;
tz_8_maxNumCoeffsX[1][1] = 4;
tz_8_maxNumCoeffsX[2][0] = 1;
tz_8_maxNumCoeffsX[2][1] = 5;
tz_8_maxNumCoeffsX[3][0] = 3;
```

```
tz_8_maxNumCoeffsX[3][1] = 3;
tz_8_maxNumCoeffsX[4][0] = 3;
tz_8_maxNumCoeffsX[4][1] = 2;
tz_8_maxNumCoeffsX[5][0] = 2;
tz_8_maxNumCoeffsX[5][1] = 2;
tz_8_maxNumCoeffsX[6][0] = 2;
tz_8_maxNumCoeffsX[6][1] = 3;
tz_8_maxNumCoeffsX[7][0] = 1;
tz_8_maxNumCoeffsX[7][1] = 3;
tz_8_maxNumCoeffsX[8][0] = 0;
tz_8_maxNumCoeffsX[8][1] = 6;

// tz_9_maxNumCoeffsX[x][0] = totalZeros.value
// tz_9_maxNumCoeffsX[x][1] = totalZeros.length
tz_9_maxNumCoeffsX[0][0] = 1;
tz_9_maxNumCoeffsX[0][1] = 6;
tz_9_maxNumCoeffsX[1][0] = 0;
tz_9_maxNumCoeffsX[1][1] = 6;
tz_9_maxNumCoeffsX[2][0] = 1;
tz_9_maxNumCoeffsX[2][1] = 4;
tz_9_maxNumCoeffsX[3][0] = 3;
tz_9_maxNumCoeffsX[3][1] = 2;
tz_9_maxNumCoeffsX[4][0] = 2;
tz_9_maxNumCoeffsX[4][1] = 2;
tz_9_maxNumCoeffsX[5][0] = 1;
tz_9_maxNumCoeffsX[5][1] = 3;
tz_9_maxNumCoeffsX[6][0] = 1;
tz_9_maxNumCoeffsX[6][1] = 2;
tz_9_maxNumCoeffsX[7][0] = 1;
tz_9_maxNumCoeffsX[7][1] = 5;

// tz_10_maxNumCoeffsX[x][0] = totalZeros.value
// tz_10_maxNumCoeffsX[x][1] = totalZeros.length
tz_10_maxNumCoeffsX[0][0] = 1;
tz_10_maxNumCoeffsX[0][1] = 5;
tz_10_maxNumCoeffsX[1][0] = 0;
tz_10_maxNumCoeffsX[1][1] = 5;
tz_10_maxNumCoeffsX[2][0] = 1;
tz_10_maxNumCoeffsX[2][1] = 3;
tz_10_maxNumCoeffsX[3][0] = 3;
tz_10_maxNumCoeffsX[3][1] = 2;
tz_10_maxNumCoeffsX[4][0] = 2;
tz_10_maxNumCoeffsX[4][1] = 2;
tz_10_maxNumCoeffsX[5][0] = 1;
tz_10_maxNumCoeffsX[5][1] = 2;
tz_10_maxNumCoeffsX[6][0] = 1;
tz_10_maxNumCoeffsX[6][1] = 4;

// tz_11_maxNumCoeffsX[x][0] = totalZeros.value
// tz_11_maxNumCoeffsX[x][1] = totalZeros.length
tz_11_maxNumCoeffsX[0][0] = 0;
tz_11_maxNumCoeffsX[0][1] = 4;
tz_11_maxNumCoeffsX[1][0] = 1;
tz_11_maxNumCoeffsX[1][1] = 4;
tz_11_maxNumCoeffsX[2][0] = 1;
tz_11_maxNumCoeffsX[2][1] = 3;
tz_11_maxNumCoeffsX[3][0] = 2;
tz_11_maxNumCoeffsX[3][1] = 3;
tz_11_maxNumCoeffsX[4][0] = 1;
tz_11_maxNumCoeffsX[4][1] = 1;
tz_11_maxNumCoeffsX[5][0] = 3;
tz_11_maxNumCoeffsX[5][1] = 3;

// tz_12_maxNumCoeffsX[x][0] = totalZeros.value
// tz_12_maxNumCoeffsX[x][1] = totalZeros.length
tz_12_maxNumCoeffsX[0][0] = 0;
tz_12_maxNumCoeffsX[0][1] = 4;
tz_12_maxNumCoeffsX[1][0] = 1;
tz_12_maxNumCoeffsX[1][1] = 4;
tz_12_maxNumCoeffsX[2][0] = 1;
tz_12_maxNumCoeffsX[2][1] = 2;
tz_12_maxNumCoeffsX[3][0] = 1;
tz_12_maxNumCoeffsX[3][1] = 1;
tz_12_maxNumCoeffsX[4][0] = 1;
tz_12_maxNumCoeffsX[4][1] = 3;

// tz_13_maxNumCoeffsX[x][0] = totalZeros.value
// tz_13_maxNumCoeffsX[x][1] = totalZeros.length
tz_13_maxNumCoeffsX[0][0] = 0;
tz_13_maxNumCoeffsX[0][1] = 3;
```

```
        tz_13_maxNumCoeffsX[1][0] = 1;
        tz_13_maxNumCoeffsX[1][1] = 3;
        tz_13_maxNumCoeffsX[2][0] = 1;
        tz_13_maxNumCoeffsX[2][1] = 1;
        tz_13_maxNumCoeffsX[3][0] = 1;
        tz_13_maxNumCoeffsX[3][1] = 2;

        // tz_14_maxNumCoeffsX[x][0] = totalZeros.value
        // tz_14_maxNumCoeffsX[x][1] = totalZeros.length
        tz_14_maxNumCoeffsX[0][0] = 0;
        tz_14_maxNumCoeffsX[0][1] = 2;
        tz_14_maxNumCoeffsX[1][0] = 1;
        tz_14_maxNumCoeffsX[1][1] = 2;
        tz_14_maxNumCoeffsX[2][0] = 1;
        tz_14_maxNumCoeffsX[2][1] = 1;

        // tz_15_maxNumCoeffsX[x][0] = totalZeros.value
        // tz_15_maxNumCoeffsX[x][1] = totalZeros.length
        tz_15_maxNumCoeffsX[0][0] = 0;
        tz_15_maxNumCoeffsX[0][1] = 1;
        tz_15_maxNumCoeffsX[1][0] = 1;
        tz_15_maxNumCoeffsX[1][1] = 1;


        // chroma(4:2:0)
        tz_1_maxNumCoeffs4 = new int[4][2];
        tz_2_maxNumCoeffs4 = new int[3][2];
        tz_3_maxNumCoeffs4 = new int[2][2];

        // tz_1_maxNumCoeffs4[x][0] = totalZeros.value
        // tz_1_maxNumCoeffs4[x][1] = totalZeros.length
        tz_1_maxNumCoeffs4[0][0] = 1;
        tz_1_maxNumCoeffs4[0][1] = 1;
        tz_1_maxNumCoeffs4[1][0] = 1;
        tz_1_maxNumCoeffs4[1][1] = 2;
        tz_1_maxNumCoeffs4[2][0] = 1;
        tz_1_maxNumCoeffs4[2][1] = 3;
        tz_1_maxNumCoeffs4[3][0] = 0;
        tz_1_maxNumCoeffs4[3][1] = 3;

        // tz_2_maxNumCoeffs4[x][0] = totalZeros.value
        // tz_2_maxNumCoeffs4[x][1] = totalZeros.length
        tz_2_maxNumCoeffs4[0][0] = 1;
        tz_2_maxNumCoeffs4[0][1] = 1;
        tz_2_maxNumCoeffs4[1][0] = 1;
        tz_2_maxNumCoeffs4[1][1] = 2;
        tz_2_maxNumCoeffs4[2][0] = 0;
        tz_2_maxNumCoeffs4[2][1] = 2;

        // tz_3_maxNumCoeffs4[x][0] = totalZeros.value
        // tz_3_maxNumCoeffs4[x][1] = totalZeros.length
        tz_3_maxNumCoeffs4[0][0] = 1;
        tz_3_maxNumCoeffs4[0][1] = 1;
        tz_3_maxNumCoeffs4[1][0] = 0;
        tz_3_maxNumCoeffs4[1][1] = 1;

        // TODO: chroma(4:2:2)
}


public void createRunBeforeField()
{
        // rb_1[x][0] = runBefore.value
        // rb_1[x][1] = runBefore.length
        rb_1 = new int[2][2];
        rb_2 = new int[3][2];
        rb_3 = new int[4][2];
        rb_4 = new int[5][2];
        rb_5 = new int[6][2];
        rb_6 = new int[7][2];
        rb_gt6 = new int[15][2];

        rb_1[0][0] = 1;
        rb_1[0][1] = 1;
        rb_1[1][0] = 0;
        rb_1[1][1] = 1;

        rb_2[0][0] = 1;
        rb_2[0][1] = 1;
        rb_2[1][0] = 1;
```

```
rb_2[1][1] = 2;
rb_2[2][0] = 0;
rb_2[2][1] = 2;

rb_3[0][0] = 3;
rb_3[0][1] = 2;
rb_3[1][0] = 2;
rb_3[1][1] = 2;
rb_3[2][0] = 1;
rb_3[2][1] = 2;
rb_3[3][0] = 0;
rb_3[3][1] = 2;

rb_4[0][0] = 3;
rb_4[0][1] = 2;
rb_4[1][0] = 2;
rb_4[1][1] = 2;
rb_4[2][0] = 1;
rb_4[2][1] = 2;
rb_4[3][0] = 1;
rb_4[3][1] = 3;
rb_4[4][0] = 0;
rb_4[4][1] = 3;

rb_5[0][0] = 3;
rb_5[0][1] = 2;
rb_5[1][0] = 2;
rb_5[1][1] = 2;
rb_5[2][0] = 3;
rb_5[2][1] = 3;
rb_5[3][0] = 2;
rb_5[3][1] = 3;
rb_5[4][0] = 1;
rb_5[4][1] = 3;
rb_5[5][0] = 0;
rb_5[5][1] = 3;

rb_6[0][0] = 3;
rb_6[0][1] = 2;
rb_6[1][0] = 0;
rb_6[1][1] = 3;
rb_6[2][0] = 1;
rb_6[2][1] = 3;
rb_6[3][0] = 3;
rb_6[3][1] = 3;
rb_6[4][0] = 2;
rb_6[4][1] = 3;
rb_6[5][0] = 5;
rb_6[5][1] = 3;
rb_6[6][0] = 4;
rb_6[6][1] = 3;

rb_gt6[0][0] = 7;
rb_gt6[0][1] = 3;
rb_gt6[1][0] = 6;
rb_gt6[1][1] = 3;
rb_gt6[2][0] = 5;
rb_gt6[2][1] = 3;
rb_gt6[3][0] = 4;
rb_gt6[3][1] = 3;
rb_gt6[4][0] = 3;
rb_gt6[4][1] = 3;
rb_gt6[5][0] = 2;
rb_gt6[5][1] = 3;
rb_gt6[6][0] = 1;
rb_gt6[6][1] = 3;
rb_gt6[7][0] = 1;
rb_gt6[7][1] = 4;
rb_gt6[8][0] = 1;
rb_gt6[8][1] = 5;
rb_gt6[9][0] = 1;
rb_gt6[9][1] = 6;
rb_gt6[10][0] = 1;
rb_gt6[10][1] = 7;
rb_gt6[11][0] = 1;
rb_gt6[11][1] = 8;
rb_gt6[12][0] = 1;
rb_gt6[12][1] = 9;
rb_gt6[13][0] = 1;
rb_gt6[13][1] = 10;
```

```
                rb_gt6[14][0] = 1;
                rb_gt6[14][1] = 11;
        }


        public int getValue()
        {
                return value;
        }


        public void setValue(int value)
        {
                this.value = value;
        }


        public int getNextByteOffset()
        {
                return nextByteOffset;
        }


        public void setNextByteOffset(int nextByteOffset)
        {
                this.nextByteOffset = nextByteOffset;
        }


        public int getNextBitOffset()
        {
                return nextBitOffset;
        }


        public void setNextBitOffset(int nextBitOffset)
        {
                this.nextBitOffset = nextBitOffset;
        }
}
```

D.1.11 Die Klasse Base64

```
public class Base64
{
        public byte[] decode(String encoded)
        {
                char b64e;
                byte b64d;
                byte[] decoded = new byte[encoded.length() * 3 / 4];

                for(int i = 0; i < encoded.length(); i++)
                {
                        b64e = encoded.charAt(i);
                if(b64e != '=')
                {
                                b64d = map(b64e);
                        if(i % 4 == 0)
                        {
                                decoded[i * 3 / 4] = (byte)((b64d & 0x3f) << 2);
                        }
                        if(i % 4 == 1)
                        {
                                decoded[i * 3 / 4] |= (byte)((b64d & 0x30) >> 4);
                                decoded[i * 3 / 4 + 1] |= (byte)((b64d & 0x0f) << 4);
                        }
                        if(i % 4 == 2)
                        {
                                decoded[i * 3 / 4] |= (byte)((b64d & 0x3c) >> 2);
                                decoded[i * 3 / 4 + 1] |= (byte)((b64d & 0x03) << 6);
                        }
                        if(i % 4 == 3)
                        {
                                decoded[i * 3 / 4] |= (byte)(b64d & 0xff);
                        }
```

```
                }
                }
                return decoded;
        }


        public byte map(char b64e)
        {
                switch(b64e)
                {
                        case 'A': return 0;
                        case 'B': return 1;
                        case 'C': return 2;
                        case 'D': return 3;
                        case 'E': return 4;
                        case 'F': return 5;
                        case 'G': return 6;
                        case 'H': return 7;
                        case 'I': return 8;
                        case 'J': return 9;
                        case 'K': return 10;
                        case 'L': return 11;
                        case 'M': return 12;
                        case 'N': return 13;
                        case 'O': return 14;
                        case 'P': return 15;
                        case 'Q': return 16;
                        case 'R': return 17;
                        case 'S': return 18;
                        case 'T': return 19;
                        case 'U': return 20;
                        case 'V': return 21;
                        case 'W': return 22;
                        case 'X': return 23;
                        case 'Y': return 24;
                        case 'Z': return 25;
                        case 'a': return 26;
                        case 'b': return 27;
                        case 'c': return 28;
                        case 'd': return 29;
                        case 'e': return 30;
                        case 'f': return 31;
                        case 'g': return 32;
                        case 'h': return 33;
                        case 'i': return 34;
                        case 'j': return 35;
                        case 'k': return 36;
                        case 'l': return 37;
                        case 'm': return 38;
                        case 'n': return 39;
                        case 'o': return 40;
                        case 'p': return 41;
                        case 'q': return 42;
                        case 'r': return 43;
                        case 's': return 44;
                        case 't': return 45;
                        case 'u': return 46;
                        case 'v': return 47;
                        case 'w': return 48;
                        case 'x': return 49;
                        case 'y': return 50;
                        case 'z': return 51;
                        case '0': return 52;
                        case '1': return 53;
                        case '2': return 54;
                        case '3': return 55;
                        case '4': return 56;
                        case '5': return 57;
                        case '6': return 58;
                        case '7': return 59;
                        case '8': return 60;
                        case '9': return 61;
                        case '+': return 62;
                        case '/': return 63;
                        default:  return -1;
                }
        }
}
```

D.1.12 Die Klasse ExtendedMath

```
public class ExtendedMath
{
	public static int ceil(double value)
	{
		if((value - (int)value) > 0)
			return (int)value + 1;
		return (int)value;
	}

	public static double log2(double value)
	{
		// logx(y) = ln(y) / ln(x)
		return (Math.log(value)/Math.log(2));
	}

}
```

D.2 Der plattformabhängige Teil

D.2.1 Die Android spezifischen Klassen

D.2.1.1 Die Klasse StartActivity

```
public class StartActivity extends Activity
{
	private EditText url;
	private EditText mp4;
	private ImageButton play;
	private ImageButton details;
	private ImageButton exit;


	public void onCreate(Bundle icicle)
	{
		super.onCreate(icicle);
		setContentView(R.layout.start);

		url = (EditText)findViewById(R.id.url);
		mp4 = (EditText)findViewById(R.id.mp4);
		play = (ImageButton)findViewById(R.id.play);
		details = (ImageButton)findViewById(R.id.details);
		exit = (ImageButton)findViewById(R.id.exit);

		setOnClickListener();
	}


	public void setOnClickListener()
	{
		play.setOnClickListener(new View.OnClickListener()
		{
			@Override
			public void onClick(View view)
			{
				String socket = url.getText().toString();
				String file = mp4.getText().toString();

				Bundle bundle = new Bundle();
				bundle.putString("socket", socket);
				bundle.putString("file", file);

				Intent intent = new Intent(StartActivity.this,
							VideoActivity.class);
				intent.putExtras(bundle);
				startActivity(intent);
```

```
                }
        }
        );

        details.setOnClickListener(new View.OnClickListener()
        {
                @Override
                public void onClick(View view)
                {
                        String socket = url.getText().toString();
                        String file = mp4.getText().toString();

                        Bundle bundle = new Bundle();
                        bundle.putString("socket", socket);
                        bundle.putString("file", file);

                        Intent intent = new Intent(StartActivity.this,
                                                   StatsActivity.class);
                        intent.putExtras(bundle);
                        startActivity(intent);
                }
        }
        );


        exit.setOnClickListener(new View.OnClickListener()
        {
                @Override
                public void onClick(View arg0)
                {
                        finish();
                }
        }
        );
    }
}
```

D.2.1.2 Die Klasse VideoActivity

```
public class VideoActivity extends Activity
{
        private StreamingHandler streamingHandler;
        private VideoView videoView;
        private MediaController mediaController;

        private ResultView resultView;
        private Thread updater;
        private Handler handler;
        private final int updateMessage = 1;
        private final int updateIntervall = 1000;

        private RTPHandler videoHandler;


        public void onCreate(Bundle icicle)
        {
                super.onCreate(icicle);
                requestWindowFeature(Window.FEATURE_NO_TITLE);
                getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
                                     WindowManager.LayoutParams.FLAG_FULLSCREEN);
                setContentView(R.layout.video);

                Bundle bundle = this.getIntent().getExtras();
                String[] socket = bundle.getString("socket").split(":");
                String host = socket[0];
                int port;
                if(socket.length > 1)
                        port = Integer.parseInt(socket[1]);
                else
                        port = 554;
                String file = bundle.getString("file");

                String[] codecs = {"H264", "MP4V-ES"};
                streamingHandler = new StreamingHandler(host, port, codecs);
                streamingHandler.start();
```

```
        mediaController = new MediaController(this);
        videoView = (VideoView) findViewById(R.id.screen);
        videoView.setVideoURI(Uri.parse("rtsp://localhost:8554/" + file));
        videoView.setMediaController(mediaController);

        startHandler();
}


@Override
public void onDestroy()
{
        super.onDestroy();
        if(streamingHandler != null)
                streamingHandler.end();
        updater.interrupt();
}


public void startHandler()
{
        resultView = (ResultView)findViewById(R.id.result);

        updater = new Thread(new Timer());
        updater.start();
        handler = new Handler()
        {
                public void handleMessage(Message msg)
                {
                        switch (msg.what)
                        {
                                case updateMessage:    setColourLight();
                                                       break;
                        }
                        super.handleMessage(msg);
                }
        };
}


public class Timer implements Runnable
{
        @Override
        public void run()
        {
                while(!Thread.currentThread().isInterrupted() && videoHandler == null)
                {
                        if(streamingHandler.getRTPHandlerList() != null)
                        {
                                // sleep to avoid concurrency access
                                try
                                {
                                        Thread.sleep(1000);
                                }
                                catch (InterruptedException e)
                                {
                                        Log.e("Quality Thread",
                                              "InterruptedException\n" + e.getMessage());
                                }

                                // get VideoHandler from HandlerList
                                for(RTPHandler rtpHandler :
                                                        streamingHandler.getRTPHandlerList())
                                {
                                        if(rtpHandler.getCodec() == "H264")
                                                videoHandler = rtpHandler;
                                        if(rtpHandler.getCodec() == "MP4V-ES")
                                                videoHandler = rtpHandler;
                                }
                        }
                }

                while(!Thread.currentThread().isInterrupted())
                {
                        try
                        {
                                Thread.sleep(updateIntervall);
                                Message msg = new Message();
                                msg.what = updateMessage;
                                handler.sendMessage(msg);
```

```
                        }
                        catch (InterruptedException e)
                        {
                                Log.e("Quality Thread",
                                        "InterruptedException\n" + e.getMessage());
                        }
                }
        }
    }

    public void setColourLight()
    {
            // jitter
            int red = (int)videoHandler.getServerHandler().getJitter()[0] -
                      ((int)videoHandler.getServerHandler().getJitter()[1] +
                      (int)videoHandler.getServerHandler().getJitter()[2]) / 16;
            // paket loss
            if(videoHandler.getServerHandler().getNumPackets() != 0)
                    red += (videoHandler.getServerHandler().getSingleLosses() +
                            videoHandler.getServerHandler().getBurstLosses() *
                            videoHandler.getServerHandler().getAvgBurstLossDuration() +
                            videoHandler.getServerHandler().getOutOfOrder()) * 1000 /
                            videoHandler.getServerHandler().getNumPackets();
            int green = 255 - red;
            resultView.setRed(red);
            resultView.setGreen(green);
            resultView.invalidate();
    }
}
```

D.2.1.3 Die Klasse StatsActivity

```
public class StatsActivity extends Activity
{
    private StreamingHandler streamingHandler;
    private String host;
    private int port;
    private VideoView videoView;
    private MediaController mediaController;

    private TextView rowOneLeft;
    private TextView rowOneRight;
    private TextView rowTwo;
    private TextView rowThree;
    private Thread updater;
    private Handler handler;
    private final int updateMessage = 1;
    private final int initMessage = 2;
    private final int updateIntervall = 1000;

    private SDPParser sdpParser;
    private DatagramParser datagramParser;
    private RTSPHandler rtspHandler;
    private RTPHandler videoHandler;
    private boolean isH264 = false;
    private boolean isMP4V = false;

    private final String DIRECTORY = "sdcard/";
    private final String FILE_NAME = "video";
    private final String FILE_TYPE = ".xml";


    public void onCreate(Bundle icicle)
    {
        super.onCreate(icicle);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
                             WindowManager.LayoutParams.FLAG_FULLSCREEN);
        setContentView(R.layout.stats);

        Bundle activity = this.getIntent().getExtras();
        String[] socket = activity.getString("socket").split(":");
        String host = socket[0];
```

```
        int port;
        if(socket.length > 1)
                port = Integer.parseInt(socket[1]);
        else
                port = 554;
        String file = activity.getString("file");

        String[] codecs = {"H264", "MP4V-ES"};
        streamingHandler = new StreamingHandler(host, port, codecs);
        streamingHandler.start();

        mediaController = new MediaController(this);
        videoView = (VideoView) findViewById(R.id.screen);
        videoView.setVideoURI(Uri.parse("rtsp://localhost:8554/" + file));
        videoView.setMediaController(mediaController);

        startHandler();
}


@Override
public void onDestroy()
{
        save();

        super.onDestroy();
        if(streamingHandler != null)
                streamingHandler.end();
        updater.interrupt();
}


public void startHandler()
{
        rowOneLeft = (TextView)findViewById(R.id.rowOneLeft);
        rowOneRight = (TextView)findViewById(R.id.rowOneRight);
        rowTwo = (TextView)findViewById(R.id.rowTwo);
        rowThree = (TextView)findViewById(R.id.rowThree);

        updater = new Thread(new Timer());
        updater.start();

        handler = new Handler()
        {
                public void handleMessage(Message message)
                {
                        switch (message.what)
                        {
                                case updateMessage:
                                        setResults();
                                        break;
                                case initMessage:
                                        setParameterSets();
                                        break;
                        }
                        super.handleMessage(message);
                }
        };
}


public class Timer implements Runnable
{
        @Override
        public void run()
        {
                while(videoHandler == null)
                {
                        if(streamingHandler.getRTPHandlerList() != null)
                        {
                                // sleep to avoid concurrency access
                                try
                                {
                                        Thread.sleep(1000);
                                }
                                catch (InterruptedException e)
                                {
                                        Log.e("Stats Thread",
                                              "InterruptedException\n" + e.getMessage());
                                }
```

```
                                // get videoHandler from RTPHandlerList
                                for(RTPHandler rtpHandler :
                                                streamingHandler.getRTPHandlerList())
                                {
                                        if (rtpHandler.getCodec() == "H264")
                                        {
                                                isH264 = true;
                                                videoHandler = rtpHandler;
                                                rtspHandler =
                                                        streamingHandler.getRTSPHandler();
                                                datagramParser = videoHandler.getParser();
                                                sdpParser =
                                         rtspHandler.getServerHandler().getParser("H264");

                                                Message message = new Message();
                                                message.what = initMessage;
                                                handler.sendMessage(message);
                                        }
                                        if (rtpHandler.getCodec() == "MP4V-ES")
                                        {
                                                isMP4V = true;
                                                videoHandler = rtpHandler;
                                                rtspHandler =
                                                        streamingHandler.getRTSPHandler();
                                        }
                                }
                        }
                }
                while(!Thread.currentThread().isInterrupted())
                {
                        try
                        {
                                Thread.sleep(updateIntervall);
                                Message message = new Message();
                                message.what = updateMessage;
                                handler.sendMessage(message);
                        }
                        catch (InterruptedException e)
                        {
                                Log.e("Stats Thread",
                                      "InterruptedException\n" + e.getMessage());
                        }
                }
        }
}


public void setParameterSets()
{
        String sequenceParameterSet = "", pictureParameterSet = "";
        int spsIndex = 0, ppsIndex = 0;
        for(int i = 0; i < sdpParser.getParameter().size(); i++)
        {
                if(sdpParser.getParameter().get(i).equals("<sequence_parameter_set>"))
                {
                        spsIndex = i;
                        break;
                }
        }
        for(int i = spsIndex; i < sdpParser.getParameter().size(); i++)
        {
                if(sdpParser.getParameter().get(i).equals("<picture_parameter_set>"))
                {
                        ppsIndex = i;
                        break;
                }
                sequenceParameterSet += sdpParser.getParameter().get(i) + "\n";
        }
        for(int i = ppsIndex; i < sdpParser.getParameter().size(); i++)
        {
                pictureParameterSet += sdpParser.getParameter().get(i) + "\n";
        }
        rowTwo.setText(sequenceParameterSet);
        rowThree.setText(pictureParameterSet);
    }


public void setResults()
{
```

```
// Network Parameter - Codec independent
long lBitrate = 0;
for(RTPHandler rtpHandler : streamingHandler.getRTPHandlerList())
       lBitrate += rtpHandler.getServerHandler().getBitrate();
long lVideoBitrate = videoHandler.getServerHandler().getBitrate();
long lJitter = videoHandler.getServerHandler().getJitter()[0];
long lJitterMax = videoHandler.getServerHandler().getJitter()[1];
long lJitterMin = videoHandler.getServerHandler().getJitter()[2];
long lSingleLosses = videoHandler.getServerHandler().getSingleLosses();
long lBurstLosses = videoHandler.getServerHandler().getBurstLosses();
long lAvgBurstLossDuration =
                    videoHandler.getServerHandler().getAvgBurstLossDuration();
long lOutOfOrder = videoHandler.getServerHandler().getOutOfOrder();
long lPackets = videoHandler.getServerHandler().getNumPackets();
long lBitsPerPacket = 0;
if(videoHandler.getServerHandler().getNumPackets() != 0)
       lBitsPerPacket = videoHandler.getServerHandler().getNumBits() /
                              videoHandler.getServerHandler().getNumPackets();
String sBitrate = "Bitrate:\t\t\t" + lBitrate + " kb/s\n";
String sVideoBitrate = "Bitrate:\t\t" + lVideoBitrate + " kb/s\n";
String sJitter = "Jitter:\t\t\t\t" + lJitter + " ms\n" +
                 "Max. Jitter:\t" + lJitterMax + "ms\n" +
                 "Min. Jitter:\t" + lJitterMin + "ms\n";
String sSingleLosses = "Single Losses:\t" + lSingleLosses + "\n";
String sBurstLosses = "Burst Losses:\t" + lBurstLosses + "(" +
                                          lAvgBurstLossDuration + ")" + "\n";
String sOutOfOrder = "Out of Order:\t" + lOutOfOrder + "\n";
String sPackets =  "Packets:\t\t\t" + lPackets + "\n";
String sBitsPerPacket = "Bits/Packet:\t\t" + lBitsPerPacket + "\n";
String serverIP = "Server IP:\t" +
rtspHandler.getServerHandler().getSocket().getInetAddress().getHostAddress() +
                                                                        "\n";

if(isH264)
{
       DecimalFormat zweiNachKomma = new DecimalFormat("#0.00");
       DecimalFormat eineNachKomma = new DecimalFormat("#0.0");

       // Transport Parameter - Non-Interleaved Packetization Mode
       long lPacketsSTAPA = datagramParser.getOutputParameter().get(3)[24];
       long lNALsSTAPA = datagramParser.getOutputParameter().get(0)[24];
       long lPacketsFUA = datagramParser.getOutputParameter().get(3)[28];
       long lNALsFUA = datagramParser.getOutputParameter().get(0)[28];
       long lPacketsSUNAL = datagramParser.getOutputParameter().get(3)[1] +
                            datagramParser.getOutputParameter().get(3)[5] +
                            datagramParser.getOutputParameter().get(3)[6] +
                            datagramParser.getOutputParameter().get(3)[9];

       String packetizationMode = "Packet. Mode:\t" +
                sdpParser.getParameter().get(sdpParser.getParameter().size() -
                1) + "\n";
       String sPacketsSTAPA = "Aggregation:\t" + lPacketsSTAPA + "(0)\n";
       if(lPacketsSTAPA != 0)
              sPacketsSTAPA = "Aggregation:\t" + lPacketsSTAPA + "(" +
                               eineNachKomma.format((double)lNALsSTAPA /
                               (double)lPacketsSTAPA) + ")" + "\n";
       String sPacketsFUA = "Fragmentation:\t" + lPacketsFUA + "(0)\n";
       if(lNALsFUA != 0)
              sPacketsFUA = "Fragmentation:\t" + lPacketsFUA + "(" +
                             eineNachKomma.format((double)lPacketsFUA /
                             (double)lNALsFUA) + ")" + "\n";
       String sPacketsSUNAL = "Single Units:\t\t" + lPacketsSUNAL + "\n";

       // H264 Parameter
       String resolution = "Resolution:\t" + sdpParser.getParameter().get(4) +
                           "*" + sdpParser.getParameter().get(5) + "\n";
       String framerate = "Framerate:\t" +
      zweiNachKomma.format((Double.parseDouble(sdpParser.getParameter().get(7))
            / Double.parseDouble(sdpParser.getParameter().get(6)) / 2)) + "\n";
       String profile = "Profile:\t\t" + sdpParser.getParameter().get(0) +
                                                                        "\n";
       String level = "Level:\t\t\t" + sdpParser.getParameter().get(1) + "\n";
       String maxGOP = "Max GOP:\t" + sdpParser.getParameter().get(2) + "\n";
       String refPics = "Ref Pics:\t\t" + sdpParser.getParameter().get(3) +
                                                                        "\n";
       String entropie = "Entropie:\t\t" + sdpParser.getParameter().get(9) +
                                                                        "\n";
       String chroma = "Chroma:\t\t" + sdpParser.getParameter().get(8) + "\n";
       String sliceGroups = "Slice Grps.:\t" + sdpParser.getParameter().get(10)
                                                                  + "\n";
```

```
                long lSlicesPerFrame = 0;
                if (datagramParser.getOutputParameter().get(4)[0] > 0)
                        lSlicesPerFrame = datagramParser.getOutputParameter().get(4)[1]
                                        / datagramParser.getOutputParameter().get(4)[0];
                String sSlicesPerFrame = "Slc./Frm.:\t" + lSlicesPerFrame + "\n";

                long lRef = (datagramParser.getOutputParameter().get(1)[1] +
                             datagramParser.getOutputParameter().get(1)[2] +
                             datagramParser.getOutputParameter().get(1)[3]);
                long lNonRef = datagramParser.getOutputParameter().get(1)[0];
                long lShortTermRef = datagramParser.getOutputParameter().get(7)[0];
                long lLongTermRef = datagramParser.getOutputParameter().get(7)[1];

                String sRef = "Ref:\t\t\t" + lRef + "\n";
                String sNonRef = "non Ref:\t\t" + lNonRef + "\n";
                String sShortTermRef ="ShortTerm:\t" + lShortTermRef + "\n";
                String sLongTermRef ="LongTerm:\t" + lLongTermRef + "\n";

                long lCodedSlice = datagramParser.getOutputParameter().get(0)[1];
                long lIDR = datagramParser.getOutputParameter().get(0)[5];
                long lAUDR = datagramParser.getOutputParameter().get(0)[9];
                long lSEI = datagramParser.getOutputParameter().get(0)[6];

                String sCodedSlice = "CS:\t\t\t\t" + lCodedSlice + "\n";
                String sIDR = "IDR:\t\t\t" + lIDR + "\n";
                String sAUD = "AUD:\t\t\t"  + lAUDR + "\n";
                String sSEI = "SEI:\t\t\t"  + lSEI + "\n";

                long lISlice = (datagramParser.getOutputParameter().get(2)[7] +
                                datagramParser.getOutputParameter().get(2)[2]);
                long lPSlice = (datagramParser.getOutputParameter().get(2)[5] +
                                datagramParser.getOutputParameter().get(2)[0]);

                String sISlice ="I:\t\t\t\t" + lISlice + "\n";
                String sPSlice ="P:\t\t\t\t" + lPSlice + "\n";

                rowOneLeft.setText("Network:\n" + sBitrate + sJitter + sSingleLosses +
                                                  sBurstLosses + sOutOfOrder + serverIP +
                                   "\nTransport:\n" + sPackets + sBitsPerPacket +
                                                  packetizationMode + sPacketsSUNAL +
                                                  sPacketsSTAPA + sPacketsFUA);
                rowOneRight.setText("H264:\n" + sVideoBitrate + resolution + framerate +
                                                  profile + level + maxGOP + refPics +
                                                  entropie + chroma + sliceGroups +
                                                  sSlicesPerFrame + "\n" + sRef + sNonRef
                                                  + sShortTermRef + sLongTermRef + "\n" +
                                                  sCodedSlice + sIDR + sSEI + sAUD + "\n"
                                                  + sISlice + sPSlice);
        }
        if(isMP4V)
        {
                rowOneLeft.setText("Network:\n" + sBitrate + sJitter + sSingleLosses +
                                                  sBurstLosses + sOutOfOrder + serverIP +
                                   "\nTransport:\n" + sPackets + sBitsPerPacket);
                rowOneRight.setText("MPEG4:\n" + sVideoBitrate);
        }
}


public void save()
{
        try
        {
                int i = 0;
                File save = new File(DIRECTORY + FILE_NAME + i + FILE_TYPE);
                File root = Environment.getExternalStorageDirectory();
                if (root.canWrite())
                {
                        while(save.exists())
                                save = new File(DIRECTORY + FILE_NAME + i++ + FILE_TYPE);
                        FileWriter writer = new FileWriter(save);
                        BufferedWriter bufWriter = new BufferedWriter(writer);

                        // Metadata
                        String time = "<time>" + new Date().toGMTString() + "</time>";
                        String url = "<url>rtsp://" + host + ":" + port +
                                                                "/" + file + "</url>";
                        // Network Parameter - Codec independent
                        long lBitrate = 0;
                        for(RTPHandler rtpHandler :streamingHandler.getRTPHandlerList())
```

```
                lBitrate += rtpHandler.getServerHandler().getBitrate();
            long lVideoBitrate=videoHandler.getServerHandler().getBitrate();
            long lJitter = videoHandler.getServerHandler().getJitter()[0];
            long lJitterMax =videoHandler.getServerHandler().getJitter()[1];
            long lJitterMin =videoHandler.getServerHandler().getJitter()[2];
            long lSingleLosses =
                        videoHandler.getServerHandler().getSingleLosses();
            long lBurstLosses =
                         videoHandler.getServerHandler().getBurstLosses();
            long lAvgBurstLossDuration =
                  videoHandler.getServerHandler().getAvgBurstLossDuration();
            long lOutOfOrder =
                         videoHandler.getServerHandler().getOutOfOrder();
            long lPackets = videoHandler.getServerHandler().getNumPackets();
            long lBitsPerPacket = 0;
            if(videoHandler.getServerHandler().getNumPackets() != 0)
                    lBitsPerPacket =
                         videoHandler.getServerHandler().getNumBits() /
                         videoHandler.getServerHandler().getNumPackets();


            String sBitrate = "<bit-rate>" + lBitrate + " kb/s" +
                                                              "</bit-rate>";
            String sVideoBitrate = "<video-bit-rate>" + lVideoBitrate +
                                                 " kb/s" + "</video-bit-rate>";
            String sJitter = "<jitter>" + lJitter + " ms" + "</jitter>" +
                         "<max_jitter>"+lJitterMax+" ms"+"</max_jitter>"+
                         "<min_jitter>"+lJitterMin+" ms"+"</min_jitter>";
            String sSingleLosses = "<single-losses>" + lSingleLosses +
                                                           "</single-losses>";
            String sBurstLosses = "<burst-losses>" + lBurstLosses + "(" +
                         lAvgBurstLossDuration + ")" + "</burst-losses>";
            String sOutOfOrder = "<out-of-order>" + lOutOfOrder +
                                                             "</out-of-order>";
            String sPackets =  "<packets>" + lPackets + "</packets>";
            String sBitsPerPacket = "<bits-per-packet>" + lBitsPerPacket +
                                                          "</bits-per-packet>";
            String serverIP = "<server-ip>" +
rtspHandler.getServerHandler().getSocket().getInetAddress().getHostAddress()
                                                             + "</server-ip>";

            if(isH264)
                    {
                    DecimalFormat zweiNachKomma = new DecimalFormat("#0.00");
                    DecimalFormat eineNachKomma = new DecimalFormat("#0.0");

                    // Transport Parameter - NonInterleaved PacketizationMode
                    long lPacketsSTAPA =
                              datagramParser.getOutputParameter().get(3)[24];
                    long lNALsSTAPA =
                              datagramParser.getOutputParameter().get(0)[24];
                    long lPacketsFUA =
                              datagramParser.getOutputParameter().get(3)[28];
                    long lNALsFUA =
                              datagramParser.getOutputParameter().get(0)[28];
                    long lPacketsSUNAL =
                              datagramParser.getOutputParameter().get(3)[1] +
                              datagramParser.getOutputParameter().get(3)[5] +
                              datagramParser.getOutputParameter().get(3)[6] +
                              datagramParser.getOutputParameter().get(3)[9];

                    String packetizationMode = "<packetization-mode>" +
                  sdpParser.getParameter().get(sdpParser.getParameter().size() -
                                                    1) + "</packetization-mode>";
                    String sPacketsSTAPA = "<aggregation-packets>" +
                             lPacketsSTAPA + "(0)" + "</aggregation-packets>";
                    if(lPacketsSTAPA != 0)
                            sPacketsSTAPA = "<aggregation-packets>" +
                                     lPacketsSTAPA + "(" +
                                     eineNachKomma.format((double)lNALsSTAPA /
                                     (double)lPacketsSTAPA) + ")" +
                                     "</aggregation-packets>";
                    String sPacketsFUA = "<fragmentation-units>" +
                               lPacketsFUA + "(0)" + "</fragmentation-units>";
                    if(lNALsFUA != 0)
                            sPacketsFUA = "<fragmentation-units>" +
                                    lPacketsFUA + "(" +
                                    eineNachKomma.format((double)lPacketsFUA /
                                    (double)lNALsFUA) + ")" +
                                    "</fragmentation-units>";
```

```
                        String sPacketsSUNAL = "<single-packets>" + lPacketsSUNAL
                                                        + "</single-packets>";

                        // H264 Parameter
                        String resolution = "<resolution>" +
                               sdpParser.getParameter().get(4) +
                               "*" + sdpParser.getParameter().get(5) +
                               "</resolution>";
                        String framerate = "<framerate>" +
zweiNachKomma.format((Double.parseDouble(sdpParser.getParameter().get(7)) /
Double.parseDouble(sdpParser.getParameter().get(6)) / 2)) + "</framerate>";
                        String profile = "<profile>" +
                                  sdpParser.getParameter().get(0) + "</profile>";
                        String level = "<level>" +
                                    sdpParser.getParameter().get(1) + "</level>";
                        String maxGOP = "<max-gop>" +
                                  sdpParser.getParameter().get(2) + "</max-gop>";
                        String refPics = "<max-ref-pics>" +
                             sdpParser.getParameter().get(3) + "</max-ref-pics>";
                        String entropie = "<entropy-coder>" +
                            sdpParser.getParameter().get(9) + "</entropy-coder>";
                        String chroma = "<chroma-format>" +
                            sdpParser.getParameter().get(8) + "</chroma-format>";
                        String sliceGroups = "<num-slice-groups>" +
                        sdpParser.getParameter().get(10) + "</num-slice-groups>";
                        long lSlicesPerFrame = 0;
                        if (datagramParser.getOutputParameter().get(4)[0] > 0)
                               lSlicesPerFrame =
                                  datagramParser.getOutputParameter().get(4)[1] /
                                  datagramParser.getOutputParameter().get(4)[0];
                        String sSlicesPerFrame = "<slices-per-frame>" +
                                        lSlicesPerFrame + "</slices-per-frame>";

                        long lRef =(datagramParser.getOutputParameter().get(1)[1]
                               + datagramParser.getOutputParameter().get(1)[2] +
                               datagramParser.getOutputParameter().get(1)[3]);
    long lNonRef = datagramParser.getOutputParameter().get(1)[0];
    long lShortTermRef = datagramParser.getOutputParameter().get(7)[0];
    long lLongTermRef = datagramParser.getOutputParameter().get(7)[1];

                        String sRef = "<ref-nals>" + lRef + "</ref-nals>";
                        String sNonRef = "<non-ref-nals>" + lNonRef +
                                                          "</non-ref-nals>";
                        String sShortTermRef = "<short-term-pics>" +
                                         lShortTermRef + "</short-term-pics>";
                        String sLongTermRef = "<long-term-pics>" + lLongTermRef +
                                                       "</long-term-pics>";

                        long lCodedSlice =
                                    datagramParser.getOutputParameter().get(0)[1];
                        long lIDR =datagramParser.getOutputParameter().get(0)[5];
                        long lAUDR=datagramParser.getOutputParameter().get(0)[9];
                        long lSEI =datagramParser.getOutputParameter().get(0)[6];

                        String sCodedSlice = "<coded-slices>" + lCodedSlice +
                                                         "</coded-slices>";
                        String sIDR = "<instantaneous-decoding-refresh>" + lIDR +
                                          "</instantaneous-decoding-refresh>";
                        String sAUD = "<access-unit-delimiter>"  + lAUDR +
                                                "</access-unit-delimiter>";
                        String sSEI = "<supplement-enhancement-message>"  + lSEI
                                          + "</supplement-enhancement-message>";

                        long lISlice =
                               (datagramParser.getOutputParameter().get(2)[7] +
                               datagramParser.getOutputParameter().get(2)[2]);
                        long lPSlice =
                               (datagramParser.getOutputParameter().get(2)[5] +
                               datagramParser.getOutputParameter().get(2)[0]);

                        String sISlice ="<i-slices>" + lISlice + "</i-slices>";
                        String sPSlice ="<p-slices>" + lPSlice + "</p-slices>";

                        bufWriter.write("<?xml version=\"1.0\" encoding=\"UTF-8\"
                               standalone=\"yes\"?><video><video>" +
                               "<execution>" + time + url + "</execution>" +
                               "<network>" + sBitrate + sJitter + sSingleLosses +
                               sBurstLosses + sOutOfOrder + serverIP +
                               "</network><transport>" + sPackets +
                               sBitsPerPacket + packetizationMode + sPacketsSUNAL
```

```
                                        + sPacketsSTAPA + sPacketsFUA  + "</transport>");
                                bufWriter.write("<h264><parameter-sets>" + sVideoBitrate
                                        + resolution + framerate + profile + level +
                                        maxGOP + refPics + entropie + chroma + sliceGroups
                                        + sSlicesPerFrame + "</parameter-sets><nal-units>"
                                        + sRef + sNonRef + sCodedSlice + sIDR + sSEI +
                                        sAUD + "</nal-units><slices>" + sShortTermRef +
                                        sLongTermRef + sISlice + sPSlice +
                                        "</slices></h264></video>");

                        }
                        if(isMP4V)
                        {
                                bufWriter.write("<?xml version=\"1.0\" encoding=\"UTF-8\"
                                        standalone=\"yes\"?><video>" + "<execution>" +
                                        time + url + "</execution>" + "<network>" +
                                        sBitrate + sJitter + sSingleLosses + sBurstLosses
                                        + sOutOfOrder + serverIP + "</network><transport>"
                                        + sPackets + sBitsPerPacket + "</transport>");
                                bufWriter.write("<mpeg4>" + sVideoBitrate +
                                        "</mpeg4></video>");
                        }
                        bufWriter.close();
                }
        }
        catch(IOException e)
        {
                Log.e("StatsActivity", "IOException: " + e.getMessage());
        }
    }
}
```

D.2.1.4 Die Klasse ResultView

```
public class ResultView extends View
{
    Paint color;
    int alpha = 255;
    int red = 255;
    int green = 0;
    int blue = 0;


    public ResultView(Context context)
    {
        super(context);
        color = new Paint();
        color.setARGB(alpha, red, green, blue);
    }


    public ResultView(Context context, AttributeSet attrs)
    {
        super(context, attrs);
        color = new Paint();
        color.setARGB(alpha, red, green, blue);
    }


    @Override
    protected void onDraw(Canvas canvas)
    {
        color.setARGB(alpha, red, green, blue);
        canvas.drawCircle(10, 10, 5, color);
    }


    public Paint getColor()
    {
        return color;
    }


    public void setColor(Paint color)
    {
```

```
                this.color = color;
        }


        public int getAlpha()
        {
                return alpha;
        }


        public void setAlpha(int alpha)
        {
                this.alpha = alpha;
        }

        public int getRed()
        {
                return red;
        }


        public void setRed(int red)
        {
                this.red = red;
        }


        public int getGreen()
        {
                return green;
        }


        public void setGreen(int green)
        {
                this.green = green;
        }


        public int getBlue()
        {
                return blue;
        }


        public void setBlue(int blue)
        {
                this.blue = blue;
        }
}
```

D.2.2 Portierung spezifische Teile

D.2.2.1 Die Klasse VideoParser

```
public class VideoParser
{
        private StreamingHandler streamingHandler;
        private SDPParser sdpParser;
        private DatagramParser datagramParser;
        private RTSPHandler rtspHandler;
        private RTPHandler videoHandler;
        private boolean isH264 = false;
        private boolean isMP4V = false;
        private Timer updater;

        private JLabel left;
        private JLabel middleLeft;
        private JLabel middleRight;
        private JLabel right;
        private JPanel panel;
        private JFrame frame;
```

```
public VideoParser()
{
	updater = new Timer();
}


public static void main(String args[])
{
	VideoParser videoParser = new VideoParser();
	String[] codecs = {"H264", "MP4V-ES"};
	String host;
	int port;
	if(args.length > 0)
		host = args[0];
	else
		host = "192.168.1.2";
	if(args.length > 1)
		port = Integer.parseInt(args[1]);
	else
		port = 554;

	videoParser.streamingHandler = new StreamingHandler(host, port, codecs);
	videoParser.streamingHandler.start();

	videoParser.updater.start();

	videoParser.left = new JLabel();
	videoParser.left.setLocation(0, 0);
	videoParser.left.setSize(250, 768);
	videoParser.left.setFont(new Font("Comic Sans", Font.PLAIN, 12));
	videoParser.left.setVerticalAlignment(JLabel.TOP);
	videoParser.left.setForeground(Color.WHITE);

	videoParser.middleLeft = new JLabel("");
	videoParser.middleLeft.setLocation(250, 0);
	videoParser.middleLeft.setSize(250, 768);
	videoParser.middleLeft.setFont(new Font("Comic Sans", Font.PLAIN, 12));
	videoParser.middleLeft.setVerticalAlignment(JLabel.TOP);
	videoParser.middleLeft.setForeground(Color.WHITE);

	videoParser.middleRight = new JLabel("");
	videoParser.middleRight.setLocation(500, 0);
	videoParser.middleRight.setSize(250, 768);
	videoParser.middleRight.setFont(new Font("Comic Sans", Font.PLAIN, 12));
	videoParser.middleRight.setVerticalAlignment(JLabel.TOP);
	videoParser.middleRight.setForeground(Color.WHITE);

	videoParser.right = new JLabel("");
	videoParser.right.setLocation(750, 0);
	videoParser.right.setSize(250, 768);
	videoParser.right.setFont(new Font("Comic Sans", Font.PLAIN, 12));
	videoParser.right.setVerticalAlignment(JLabel.TOP);
	videoParser.right.setForeground(Color.WHITE);

	videoParser.panel = new JPanel();
	videoParser.panel.setLayout(null);
	videoParser.panel.setBackground(Color.DARK_GRAY);
	videoParser.panel.add(videoParser.left);
	videoParser.panel.add(videoParser.middleLeft);
	videoParser.panel.add(videoParser.middleRight);
	videoParser.panel.add(videoParser.right);

	videoParser.frame = new JFrame();
	videoParser.frame.setTitle("Video Parser");
	videoParser.frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	videoParser.frame.setSize(1024, 768);
	videoParser.frame.add(videoParser.panel);
	videoParser.frame.setVisible(true);
}


public class Timer extends Thread
{
	// @Override
	public void run()
	{
		while(videoHandler == null)
		{
			if(streamingHandler.getRTPHandlerList() != null)
			{
```

```
                                // sleep to avoid concurrency access
                                try
                                {
                                        Thread.sleep(1000);
                                }
                                catch (InterruptedException e)
                                {
                                System.out.println("Stats Thread: InterruptedException\n"
                                                                        + e.getMessage());
                                }

                                // get videoHandler from RTPHandlerList
                                for(RTPHandler rtpHandler :
                                                        streamingHandler.getRTPHandlerList())
                                {
                                        if (rtpHandler.getCodec() == "H264")
                                        {
                                                isH264 = true;
                                                videoHandler = rtpHandler;
                                                rtspHandler =
                                                        streamingHandler.getRTSPHandler();
                                                datagramParser = videoHandler.getParser();
                                                sdpParser =
                                                 rtspHandler.getServerHandler().getParser("H264");

                                                setParameterSets();
                                        }
                                        if (rtpHandler.getCodec() == "MP4V-ES")
                                        {
                                                isMP4V = true;
                                                videoHandler = rtpHandler;
                                                rtspHandler =
                                                        streamingHandler.getRTSPHandler();
                                        }
                                }
                        }
                }
                while(!Thread.currentThread().isInterrupted())
                {
                        setResults();
                }
        }
}


public void setParameterSets()
{
        String sequenceParameterSet = "<html><body>",
                pictureParameterSet = "<html><body>";
        int spsIndex = 0, ppsIndex = 0;
        for(int i = 0; i < sdpParser.getParameter().size(); i++)
        {
                if(sdpParser.getParameter().get(i).equals("<sequence_parameter_set>"))
                {
                        spsIndex = i;
                        break;
                }
        }
        for(int i = spsIndex; i < sdpParser.getParameter().size(); i++)
        {
                if(sdpParser.getParameter().get(i).equals("<picture_parameter_set>"))
                {
                        ppsIndex = i;
                        break;
                }
                sequenceParameterSet += deXML(sdpParser.getParameter().get(i)) + "<br>";
        }
        sequenceParameterSet += "</body></html>";
        for(int i = ppsIndex; i < sdpParser.getParameter().size(); i++)
        {
                pictureParameterSet += deXML(sdpParser.getParameter().get(i)) + "<br>";
        }
        pictureParameterSet += "</body></html>";
        middleRight.setText(sequenceParameterSet);
        middleRight.revalidate();
        right.setText(pictureParameterSet);
        right.revalidate();
}
```

```
public String deXML(String parameter)
{
        if(parameter.contains("<") && parameter.contains(">"))
        {
                String name = parameter.substring(1, parameter.indexOf(">"));
                String value = parameter.substring(parameter.indexOf(">") + 1,
                                                   parameter.length());
                return  name + " " + value;
        }
        return parameter;
}


public void setResults()
{
        // Network Parameter - Codec independent
        long lBitrate = 0;
        for(RTPHandler rtpHandler : streamingHandler.getRTPHandlerList())
                lBitrate += rtpHandler.getServerHandler().getBitrate();
        long lVideoBitrate = videoHandler.getServerHandler().getBitrate();
        long lJitter = videoHandler.getServerHandler().getJitter()[0];
        long lJitterMax = videoHandler.getServerHandler().getJitter()[1];
        long lJitterMin = videoHandler.getServerHandler().getJitter()[2];
        long lSingleLosses = videoHandler.getServerHandler().getSingleLosses();
        long lBurstLosses = videoHandler.getServerHandler().getBurstLosses();
        long lAvgBurstLossDuration =
                          videoHandler.getServerHandler().getAvgBurstLossDuration();
        long lOutOfOrder = videoHandler.getServerHandler().getOutOfOrder();
        long lPackets = videoHandler.getServerHandler().getNumPackets();
        long lBitsPerPacket = 0;
        if(videoHandler.getServerHandler().getNumPackets() != 0)
                lBitsPerPacket = videoHandler.getServerHandler().getNumBits() /
                                      videoHandler.getServerHandler().getNumPackets();
        String sBitrate = "Bitrate:\t\t\t" + lBitrate + " kb/s\n";
        String sVideoBitrate = "Bitrate:\t\t" + lVideoBitrate + " kb/s\n";
        String sJitter = "Jitter:\t\t\t\t" + lJitter + " ms\n";
        String sMaxJitter = "Max. Jitter:\t\t" + lJitterMax + "ms\n";
        String sMinJitter = "Min. Jitter:\t\t" + lJitterMin + "ms\n";
        String sSingleLosses = "Single Losses:\t" + lSingleLosses + "\n";
        String sBurstLosses = "Burst Losses:\t" + lBurstLosses + "(" +
                                                    lAvgBurstLossDuration + ")" + "\n";
        String sOutOfOrder = "Out of Order:\t" + lOutOfOrder + "\n";
        String sPackets =  "Packets:\t\t\t" + lPackets + "\n";
        String sBitsPerPacket = "Bits/Packet:\t" + lBitsPerPacket + "\n";
        String serverIP = "Server IP:\t" +
           rtspHandler.getServerHandler().getSocket().getInetAddress().getHostAddress()
           + "\n";

        if(isH264)
        {
                // Transport Parameter - Non-Interleaved Packetization Mode
                long lPacketsSTAPA = datagramParser.getOutputParameter().get(3)[24];
                long lNALsSTAPA = datagramParser.getOutputParameter().get(0)[24];
                long lPacketsFUA = datagramParser.getOutputParameter().get(3)[28];
                long lNALsFUA = datagramParser.getOutputParameter().get(0)[28];
                long lPacketsSUNAL = datagramParser.getOutputParameter().get(3)[1] +
                                     datagramParser.getOutputParameter().get(3)[5] +
                                     datagramParser.getOutputParameter().get(3)[6] +
                                     datagramParser.getOutputParameter().get(3)[9];

                String packetizationMode = "Packet. Mode:\t" +
                           sdpParser.getParameter().get(sdpParser.getParameter().size()
                           - 1) + "\n";
                String sPacketsSTAPA = "Aggregation:\t" + lPacketsSTAPA + "(0)\n";
                if(lPacketsSTAPA != 0)
                       sPacketsSTAPA = "Aggregation:\t" + lPacketsSTAPA + "(" +
                                       lNALsSTAPA/lPacketsSTAPA + ")" + "\n";
                String sPacketsFUA = "Fragmentation:\t" + lPacketsFUA + "(0)\n";
                if(lNALsFUA != 0)
                       sPacketsFUA = "Fragmentation:\t" + lPacketsFUA + "(" +
                                      lPacketsFUA/lNALsFUA + ")" + "\n";
                String sPacketsSUNAL = "Single Units:\t\t" + lPacketsSUNAL + "\n";

                // H264 Parameter
                String resolution = "Resolution:\t" + sdpParser.getParameter().get(4) +
                                    "*" + sdpParser.getParameter().get(5) + "\n";
                DecimalFormat precision = new DecimalFormat("#0.00");
                String framerate = "Framerate:\t" +
                   precision.format((Double.parseDouble(sdpParser.getParameter().get(7))
                   / Double.parseDouble(sdpParser.getParameter().get(6)) / 2)) + "\n";
```

```
String profile = "Profile:\t\t" + sdpParser.getParameter().get(0) +
                 "\n";
String level = "Level:\t\t\t" + sdpParser.getParameter().get(1) + "\n";
String maxGOP = "Max GOP:\t" + sdpParser.getParameter().get(2) + "\n";
String refPics = "Ref Pics:\t\t" + sdpParser.getParameter().get(3) +
                 "\n";
String entropie = "Entropie:\t\t" + sdpParser.getParameter().get(9)
                  + "\n";
String chroma = "Chroma:\t\t" + sdpParser.getParameter().get(8) + "\n";
String sliceGroups = "Slice Grps.:\t" + sdpParser.getParameter().get(10)
                     + "\n";
long lSlicesPerFrame = 0;
if (datagramParser.getOutputParameter().get(4)[0] > 0)
       lSlicesPerFrame = datagramParser.getOutputParameter().get(4)[1]
                         / datagramParser.getOutputParameter().get(4)[0];
String sSlicesPerFrame = "Slc./Frm.:\t" + lSlicesPerFrame + "\n";

long lRef = (datagramParser.getOutputParameter().get(1)[1] +
            datagramParser.getOutputParameter().get(1)[2] +
            datagramParser.getOutputParameter().get(1)[3]);
long lNonRef = datagramParser.getOutputParameter().get(1)[0];
long lShortTermRef = datagramParser.getOutputParameter().get(7)[0];
long lLongTermRef = datagramParser.getOutputParameter().get(7)[1];

String sRef = "Ref:\t\t\t" + lRef + "\n";
String sNonRef = "non Ref:\t\t" + lNonRef + "\n";
String sShortTermRef ="ShortTerm:\t" + lShortTermRef + "\n";
String sLongTermRef ="LongTerm:\t" + lLongTermRef + "\n";

long lCodedSlice = datagramParser.getOutputParameter().get(0)[1];
long lIDR = datagramParser.getOutputParameter().get(0)[5];
long lAUDR = datagramParser.getOutputParameter().get(0)[9];
long lSEI = datagramParser.getOutputParameter().get(0)[6];

String sCodedSlice = "CS:\t\t\t\t" + lCodedSlice + "\n";
String sIDR = "IDR:\t\t\t" + lIDR + "\n";
String sAUD = "AUD:\t\t\t"  + lAUDR + "\n";
String sSEI = "SEI:\t\t\t"  + lSEI + "\n";

long lISlice = (datagramParser.getOutputParameter().get(2)[7] +
                datagramParser.getOutputParameter().get(2)[2]);
long lPSlice = (datagramParser.getOutputParameter().get(2)[5] +
                datagramParser.getOutputParameter().get(2)[0]);

String sISlice ="I:\t\t\t\t" + lISlice + "\n";
String sPSlice ="P:\t\t\t\t" + lPSlice + "\n";

long lINxN = datagramParser.getOutputParameter().get(5)[5];
long lI16x16 = 0;
for(int i = 6; i < 30; i++)
       lI16x16 += datagramParser.getOutputParameter().get(5)[i];
long lP8x8 = datagramParser.getOutputParameter().get(5)[3] +
             datagramParser.getOutputParameter().get(5)[4];
long lP8x16 = datagramParser.getOutputParameter().get(5)[2];
long lP16x8 = datagramParser.getOutputParameter().get(5)[1];
long lP16x16 = datagramParser.getOutputParameter().get(5)[0];
long lP_Skip = datagramParser.getOutputParameter().get(5)[55];

long lSubP8x8 = datagramParser.getOutputParameter().get(6)[0];
long lSubP8x4 = datagramParser.getOutputParameter().get(6)[1];
long lSubP4x8 = datagramParser.getOutputParameter().get(6)[2];
long lSubP4x4 = datagramParser.getOutputParameter().get(6)[3];

String sINxN = "I NxN:\t\t\t\t" + lINxN + "\n";
String sI16x16 = "I 16x16:\t\t\t\t" + lI16x16 + "\n";
String sP8x8 = "P 8x8:\t\t\t\t" + lP8x8 + "\n";
String sP8x16 = "P 8x16:\t\t\t\t" + lP8x16 + "\n";
String sP16x8 = "P 16x8:\t\t\t\t" + lP16x8 + "\n";
String sP16x16 = "P 16x16:\t\t\t\t" + lP16x16 + "\n";
String sP_Skip = "P_Skip:\t\t\t\t" + lP_Skip + "\n";

String sSubP8x8 ="P 8x8:\t\t\t\t" + lSubP8x8 + "\n";
String sSubP8x4 ="P 8x4:\t\t\t\t" + lSubP8x4 + "\n";
String sSubP4x8 ="P 4x8:\t\t\t\t" + lSubP4x8 + "\n";
String sSubP4x4 ="P 4x4:\t\t\t\t" + lSubP4x4 + "\n";

left.setText("<html><body>Network:<br>" + sBitrate + "<br>" + sJitter +
                                  "<br>" + sMaxJitter + "<br>" +
                                  sMinJitter + "<br>" + sSingleLosses +
                                  "<br>" + sBurstLosses + "<br>" +
```

```
                                                    sOutOfOrder + "<br>" + serverIP +
                                                    "<br><br>" +
                                    "Transport:<br>" + sPackets + "<br>" + sBitsPerPacket +
                                                    "<br>" + packetizationMode + "<br>" +
                                                    sPacketsSUNAL + "<br>" + sPacketsSTAPA
                                                    + "<br>" +sPacketsFUA + "<br></body></
                                                                                    html>");
                        left.revalidate();
                        middleLeft.setText("<html><body>H264:<br>" + sVideoBitrate + "<br>" +
                                                        resolution + "<br>" + framerate +
                                                        "<br>" + profile + "<br>" + level +
                                                        "<br>" + maxGOP + "<br>" + refPics
                                                        + "<br>" + entropie + "<br>" +
                                                        chroma + "<br>" + sliceGroup  s +
                                                        "<br>" + sSlicesPerFrame + "<br>" +
                                                        "<br>" + sRef + "<br>" + sNonRef +
                                                        "<br>" + sShortTermRef + "<br>"
                                                        +sLongTermRef + "<br>" + "<br>" +
                                                        sCodedSlice + "<br>" + sIDR +
                                                        "<br>" + sSEI + "<br>" + sAUD +
                                                        "<br>" + "<br>" + sISlice + "<br>"
                                                        + sPSlice + "<br>" + "<br>" + sINxN
                                                        + "<br>" + sI16x16 + "<br>" + sP8x8
                                                        + "<br>" + sP8x16 + "<br>" + sP16x8
                                                        + "<br>" + sP16x16 + "<br>" +
                                                        sP_Skip + "<br>" + "<br>" +
                                                        sSubP8x8 + "<br>" + sSubP8x4 +
                                                        "<br>" + sSubP4x8 + "<br>" +
                                                        sSubP4x4 + "<br>");
                        middleLeft.revalidate();
                }
                if(isMP4V)
                {
                        left.setText("<html><body>Network:<br>" + sBitrate + "<br>" + sJitter +
                                                        "<br>" + sMaxJitter + "<br>" +
                                                       sMinJitter + "<br>"  +
                                                        sSingleLosses + "<br>" +
                                                        sBurstLosses + "<br>" + sOutOfOrder
                                                        + "<br>" + serverIP);
                        middleLeft.setText("<html><body>MPEG4:<br>" + sVideoBitrate);
                }
        }
}
```

Anhang E - Quellcodes des TestPlayers

E.1 Die Klasse PlayerActivity

```
public class PlayerActivity extends Activity
{
	private VideoView videoView;
	private MediaController mediaController;


	public void onCreate(Bundle icicle)
	{
		super.onCreate(icicle);
		requestWindowFeature(Window.FEATURE_NO_TITLE);
		getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
					WindowManager.LayoutParams.FLAG_FULLSCREEN);
		setContentView(R.layout.video);

		Bundle bundle = this.getIntent().getExtras();
		String socket = bundle.getString("socket");
		String file = bundle.getString("file");

		mediaController = new MediaController(this);

		videoView = (VideoView) findViewById(R.id.screen);
		videoView.setVideoURI(Uri.parse("rtsp://" + socket + "/" + file));
		videoView.setMediaController(mediaController);
		videoView.requestFocus();
	}
}
```

E.2 Die Klasse TestPlayer

```
public class TestPlayer extends Activity
{
	private ImageButton play;
	private ImageButton exit;
	private EditText ip;
	private EditText mp4;


	public void onCreate(Bundle icicle)
	{
		super.onCreate(icicle);
		setContentView(R.layout.start);

		play = (ImageButton)findViewById(R.id.play);
		exit = (ImageButton)findViewById(R.id.exit);
		ip = (EditText)findViewById(R.id.ip);
		mp4 = (EditText)findViewById(R.id.mp4);

		setOnClickListener();
	}


	public void setOnClickListener()
	{
		play.setOnClickListener(new View.OnClickListener()
		{
			@Override
			public void onClick(View view)
			{
				String socket = ip.getText().toString();
				String file = mp4.getText().toString();
```

```
                        Bundle bundle = new Bundle();
                        bundle.putString("socket", socket);
                        bundle.putString("file", file);

                        Intent intent = new Intent(TestPlayer.this,
                                                   PlayerActivity.class);
                        intent.putExtras(bundle);
                        startActivity(intent);
                }
        }
        );


        exit.setOnClickListener(new View.OnClickListener()
        {
                @Override
                public void onClick(View arg0)
                {
                        finish();
                }
        }
        );
    }
}
```