Universität Karlsruhe (TH) Institut für Telematik

Entwurf und Implementierung einer Simulation des Real-time Transport Protocol unter OMNeT++

Studienarbeit am Institut für Telematik
Prof. Dr. W. Juling
Fakultät für Informatik
Universität Karlsruhe (TH)

von

cand. inform.
Matthias Oppitz

Betreuer:
Prof. Dr. W. Juling
Dipl.-Inform. Verena Kahmann
Dipl.-Inform. Bernhard Thurm

Tag der Anmeldung: 1. März 2002
Tag der Abgabe: 31. Mai 2002

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Karlsruhe, den 31. Mai 2002

Inhaltsverzeichnis

1. Einleitung

Mit der heutzutage verfügbaren Rechenleistung von Computern und der Bandbreite von Netzwerken ist eine Übertragung und Verarbeitung von Echtzeitdaten möglich. Als ein Standard für die Übertragung solcher Datenströme hat sich das *Real-time Transport Protocol* (RTP)[Grou96a] etabliert.

Es hat folgende Eigenschaften:

- RTP bietet Anwendungen eine Ende-zu-Ende-Transportverbindung, die geeignet ist, *Echtzeitdaten* zu übertragen.
- Es kann sowohl im *Unicast*-Fall, als auch in *Multicast*-Konferenzen verwendet werden.
- RTP setzt normalerweise auf dem *User Datagram Protocol* (*UDP*) auf und verwendet dessen Multiplex- und Prüfsummendienste. Es kann aber auch mit anderen Netzwerk- und Transportprotokollen verwendet werden.
- RTP selbst reserviert keine Ressourcen, die für die Übertragung der Daten benötigt werden und stellt auch nicht die rechtzeitige Auslieferung der Pakete sicher.
- Das *RTP Control Protocol* (RTCP) unterstützt RTP bei der Datenübertragung. Mit seiner Hilfe kann die Auslieferung der RTP-Pakete überwacht werden. Des Weiteren ermöglicht es, andere Teilnehmer zu erkennen und zu beschreiben.
- Des Weiteren wird der Einsatz von Zwischensystemen auf RTP-Ebene unterstützt.

Um die Eignung von RTP für ein konkretes Anwendungsszenario zu prüfen, und dabei besonders seine Skalierbarkeit für große Multicast-Konferenzen zu untersuchen, bietet sich an, das Protokoll in einer Simulationsumgebung ablaufen zu lassen. Dadurch kann sein Verhalten unter veränderbaren Bedingungen untersucht werden und es lässt sich entscheiden, ob es die Anforderungen für diese Anwendung erfüllt.

1.1 Zielsetzung der Arbeit

Im Rahmen dieser Studienarbeit wird ein Modell von RTP für die Simulationsumgebung OMNeT++ entwickelt. Dies beinhaltet folgende Tätigkeiten:

- Reduktion des Protokolls auf die für eine Simulation relevanten Elemente und Funktionen.
- Entwurf einer geeigneten Modulstruktur in OMNeT++.
- Implementierung der Module.
- Erstellung eines Moduls für das Audio-Video-Profil [Grou96b].
- Erstellung von Modulen für die Übertragung von MPEG-Videos nach RFC 2250 [Grou98].
- Aufbau und Test von Beispielnetzwerken.
- Dokumentation des Modells.

Das Modell von RTP setzt dabei zur Datenübertragung auf die in [Zhu02] entwickelte Socketlayer-Bibliothek und auf die in [Rebe00] entwickelte IP-Suite auf. Im Rahmen der Simulation werden keine echten MPEG-Video-Daten übertragen, sondern nur simulierte. Für die Erstellung dieser Simulations-Daten und die Analyse der übertragenen Daten wird auf die in [Barq00] entwickelten Programme `MpegStat` und `MpegPlay` zurückgegriffen sowie die darin beschriebenen Dateiformate verwendet.

1.2 Gliederung der Arbeit

Zunächst werden in Kapitel 2.1 das Real-time Transport Protocol und das RTP Control Protocol ausführlich dargestellt. Daran schießt sich eine Vorstellung der für die MPEG-Übertragung notwendigen Standards an: Das Audio/Video-Profil für RTP in Kapitel 2.2, das Format von MPEG-Video soweit es für die Simulation von Interesse ist in Kapitel 2.3.1, sowie das Verpackungsschema für die Übertragung von MPEG-Video über RTP in Kapitel 2.3.2.

Danach folgt in Kapitel 2.4 ein Überblick über die Simulationsumgebung OMNeT++ und ihre Funktionsweise. Das Kapitel 3 über den Entwurf des RTP-Modells beinhaltet eine Vorstellung der Anforderungen und Zielsetzungen sowie eine ausgiebige Beschreibung der einzelnen Module und deren Zusammenarbeit. Die Beschreibung der Implementierung in Kapitel 4 enthält Informationen über deren Eigenschaften, wie sie besonders für die Erweiterung dieses Modells für andere Profile oder Nutzdatentypen von Interesse sind. In Kapitel 5 wird das Modell anhand einiger Beispielnetzwerke evaluiert. Dabei wird auch aufgezeigt, welche Möglichkeiten das entwickelte Modell bietet.

Die abschließende Zusammenfassung in Kapitel 6 stellt nochmals dar, was im Rahmen dieser Studienarbeit entwickelt und implementiert wurde. Es werden außerdem Anregungen gegeben, welche Erweiterungen und Ergänzungen des Modells zweckmäßig erscheinen und in welchen Anwendungsszenarien es eingesetzt werden kann.

2. Grundlagen

2.1 Das Real-time Transport Protocol

Die in dieser Arbeit betrachtete Version 2 des *Real-time Transport Protocol* (RTP) wurde im Januar 1996 im RFC 1889 [Grou96a] veröffentlicht. Es bietet Applikationen eine Ende-zu-Ende-Übertragung von *Echtzeitdaten*. Darunter sind Daten mit Zeitbezug zu verstehen, wie z. B. ein Videofilm, dessen Einzelbilder in einem bestimmten Zeitabstand aufgenommen wurden und im gleichen Zeitabstand wieder abgespielt werden müssen, oder z. B. Messwerte, die an einem bestimmten Zeitpunkt abgelesen wurden.

Grundsätzlich ist RTP geeignet, jede Art von Echtzeitdaten zu übertragen, der RFC spezifiziert nur das Protokoll selbst, nicht aber, wie bestimmte Nutzdaten damit übertragen werden. Daher benötigt man für eine Datenübertragung mit RTP weitere Dokumente: Ein *Profil* und für jede Art von zu übertragenden Nutzdaten ein *Verpackungsschema*. Das Profil legt dabei globale Parameter für die Übertragung fest, wie etwa eine Zuordnung von Nutzdaten-Typ-Nummer zu einem konkreten Datenformat. Im Verpackungsschema wird dann festgelegt, wie solche Daten mit RTP übertragen werden.

2.1.1 Anforderungen

Für seine Aufgaben benötigt RTP die Dienste anderer Protokolle. Obwohl es eigentlich ein Transportprotokoll ist, setzt es normalerweise nicht direkt auf ein Netzwerkprotokoll wie das *Internet Protocol* (IP), sondern auf ein Datagramm-orientiertes Transportprotokoll auf. In den meisten Fällen wird das *User Datagram Protocol* (UDP) verwendet. RTP benutzt dessen Multiplex-Dienst, um über die gleiche Netzwerkverbindung sowohl RTP als auch RTCP zu übertragen und seinen Prüfsummen-Dienst, um fehlerhafte Pakete erkennen zu können. Selbstverständlich kann RTP aber auch auf andere Protokollfamilien aufsetzen, wie etwa *Asynchronous Transfer Mode* (ATM).

Um seine Dienste in zuverlässiger Weise anbieten zu können, stellt RTP einige Anforderungen an die verwendete Netzwerkschicht: Um eine Datenübertragung ohne allzu viele Paketverluste durchführen zu können, sollte ausreichende Bandbreite für RTP reserviert werden. Damit die empfangenen Daten sofort angezeigt oder abgespielt werden können, ohne dass es dabei zu ungewollten Pausen kommt, ist es notwendig, dass Schwankungen der Übertragungsdauer (*Jitter*) unter einem den Daten angemessenen Wert bleiben. Die Einhaltung der *Quality-of-Service-Parameter* ist daher vor der eigentlichen Übertragung mit Hilfe eines entsprechenden Protokolls, wie etwa mit dem *Resource Reservation Protocol* (*RSVP*), der Netzwerkschicht mitzuteilen.

Da RTP nicht nur für Unicast-, sondern auch für Multicast-Übertragungen geeignet ist, ist für diesen Fall eine multicast-fähige Netzwerkschicht natürlich zwingend notwendig. Für eine Multicast-Übertragung benötigt man noch ein Protokoll zur Steuerung der Gruppenmitgliedschaft, wie etwa das *Internet Group Management Protocol* (IGMP).

Zur Steuerung der Datenübertragung mit RTP, wie etwa das zeitweise Anhalten der Übertragung und ein späteres Fortsetzen, erfolgt durch ein weiteres Protokoll, das *Real-time Streaming Protocol* (*RTSP*).

2.1.2 Die RTP-Sitzung

Sobald die vor einer Übertragung notwendigen Aufgaben mittels geeigneter Protokolle erledigt sind, kann RTP zum Einsatz kommen. Die Kommunikation erfolgt dabei immer in einer sogenannten *Sitzung*, eine Assoziation aller an der Übertragung beteiligten Systeme anhand zweier Transportadressen: eine Netzwerkadresse und Portnummern für RTP und RTCP. Bei einer Multicast-Übertragung sind die Transportadressen für alle Teilnehmer gleich, bei einer Unicast-Übertragung ist die Netzwerkadresse die des Partners, während die Portnummern für beide gleich sind.

Innerhalb einer Sitzung werden nur Daten eines Nutzdatentyps übertragen. Falls mehrere Datentypen zusammen übertragen werden sollen, wie etwa Videobilder und der zugehörige Ton, muss dies in zwei parallelen Sitzungen geschehen.

2.1.3 Identifikation der Teilnehmer

Jeder Teilnehmer einer RTP-Sitzung wird durch zwei zwingend notwendige Bezeichner identifiziert: Den *Common Name* (CNAME) und den *Synchronization Source Identifier* (SSRC).

Der CNAME ist eine global eindeutige, algorithmisch bestimmte Zeichenkette, die einen Teilnehmer sitzungsübergreifend identifiziert. In Mehrbenutzerumgebungen wird oft `user@host.domain`, in Einbenutzerumgebungen `host.domain` verwendet. Mit seiner Hilfe können Teilnehmer sitzungsübergreifend identifiziert werden. Der CNAME ist ein sogenanntes *Source Description Item* (SDES item) und wird mit RTCP-Paketen übertragen.

Der SSRC ist eine 32 Bit breite Ganzzahl, welche die Teilnehmer mit einem Zufallszahlengenerator wählen. Er dient zur eindeutigen Identifikation eines Teilnehmers innerhalb einer Sitzung und wird sowohl in jedem RTP-Datenpaket, als auch in RTCP-Paketen übertragen. Innerhalb einer Sitzung dürfen keine zwei Teilnehmer den gleichen SSRC haben, deshalb sollte jede RTP-Implementierung SSRC-Konflikte erkennen und beheben können.

2.1.4 RTP-Zwischensysteme

Das Real-time Transport Protocol unterstützt zwei Arten von Zwischensystemen auf RTP-Ebene: *Translator* und *Mixer*.

Ein Translator leitet RTP- und RTCP-Pakete transparent weiter, wobei er auch Veränderungen an den Nutzdaten vornehmen kann. Er kann eingesetzt werden, um Pakete an einen Teilnehmer weiterzuleiten, der sich in einem nicht Multicast-fähigen Netzwerk befindet. Der Translator empfängt die Pakete für die Multicast-Gruppe und sendet sie als Unicast an den Empfänger; von diesem empfangene Pakete werden an die Multicast-Gruppe verteilt. Eine weitere Anwendungsmöglichkeit besteht darin, ihn einem Teilnehmer, der nur über eine schmalbandige Anbindung verfügt, vorzuschalten und die Datenrate durch geeignete Maßnahmen, wie zum Beispiel durch Verkleinerung der Bildauflösung, zu verringern.

Ein Mixer zeichnet sich im Gegensatz zu einem Translator dadurch aus, dass er mehrere RTP-Pakete zu einem einzigen RTP-Paket zusammen fassen kann, wodurch Bandbreite gespart werden kann. So können zum Beispiel bei einer Telefonkonferenz alle zur gleichen Zeit gesendeten RTP-Pakete zu einem Paket zusammen gemischt werden, da die Informationen sowieso beim Empfänger zusammen gemischt werden, wenn sie ausgegeben werden. Ein Mixer ist für andere Teilnehmer erkennbar, er verfügt über einen eigenen SSRC. Die SSRCs aller zusammen gemischten Sender werden im RTP-Paket in einer Liste als *Contributing Sources* (CSRC) aufgeführt.

Bei beiden Arten von Zwischensystemen ist es möglich, dass es zu Rückkopplungen durch Schleifenbildung kommt. Eine Implementierung sollte dies erkennen und beheben können.

2.1.5 RTP-Pakete

Das Real-time Transport Protocol (RTP) dient innerhalb einer RTP-Sitzung zur Übertragung der Nutzdaten. Es setzt meist auf UDP auf, wobei ein RTP-Paket in ein UDP-Paket gekapselt wird.

Ein RTP-Paket besteht aus einem Kopf und den Nutzdaten. Der Paketkopf enthält einige Felder fester Größe, eine variabel große Liste von Contributing Sources und eine optionale Paketkopf-Erweiterung. Die Länge der Nutzdaten wird durch die maximale Größe eines Pakets bestimmt; wie die Nutzdaten in einem RTP-Paket gespeichert werden, wird im Verpackungsschema fest gelegt.

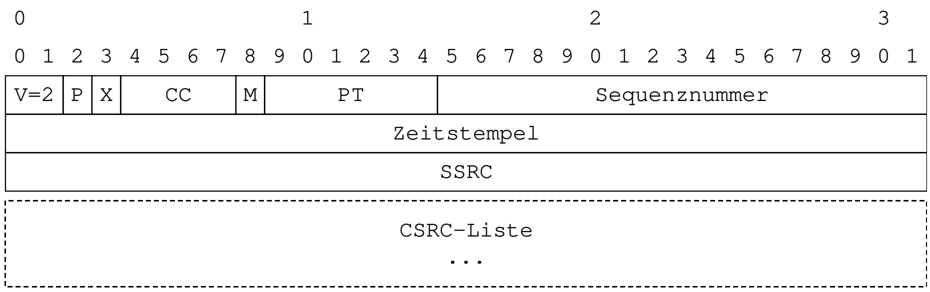

Abbildung 2.1: Der Kopf eines RTP-Paketes.

Version (V): 2 Bit. Dieses Feld identifiziert die verwendete Version des Protokolls. Die hier beschriebene Version ist 2.

Füllbytes (padding, P): 1 Bit. Das Padding-Bit ist 1, falls am Ende des RTP-Paketes Bytes angehängt wurden, damit die Paketgröße ein Vielfaches einer bestimmten Größe ist, was für einige Verschlüsselungsalgorithmen notwendig ist. Das letzte Byte im Paket gibt an, wie viele Padding-Bytes angehängt wurden.

Erweiterung (extension, X): 1 Bit. Ein RTP-Paket kann eine Erweiterung des Paketkopfs besitzen, die auf den Standard-Paketkopf folgt. Falls das Paket eine solche Erweiterung besitzt, ist dieses Bit auf 1 gesetzt.

CSRC Zähler (CSRC count, CC): 4 Bit. Dieses Feld enthält die Anzahl SSRC Identifiers, die in der CSRC Liste eingetragen sind.

Markierung (M): 1 Bit. Ein gesetztes Markierungsbit weist darauf hin, dass mit diesem RTP-Paket eine signifikante Änderung des Datenstroms einhergeht, dass etwa bei der Übertragung eines Videofilms das Ende eines Einzelbildes erreicht ist. Die genaue Bedeutung der Markierung wird im Profil und im Verpackungsschema festgelegt. Ein Profil kann außerdem festlegen, dass weitere Markierungsbits auf Kosten des folgenden Feldes im RTP-Paketkopf enthalten sind; auch gar kein Markierungsbit ist möglich.

Nutzdaten-Typ (payload type, PT): 7 Bit. Mit diesem Feld wird das Format der in diesem RTP-Paket übertragenen Nutzdaten identifiziert. Die Zuordnung von einem Wert zu einem konkreten Datenformat erfolgt durch das Profil.

Sequenznummer: 16 Bit. Alle RTP-Pakete eines Senders werden fortlaufend nummeriert, damit Empfänger Duplikate, Paketverluste und Vertauschungen erkennen können. Der Startwert wird zufällig gewählt, um Klartextangriffe auf verschlüsselte RTP-Pakete zu erschweren. Falls Laufe einer Übertragung der zulässige Zahlenwert für eine 16 Bit breite Ganzahl überschritten wird, erfolgt ein Umlauf zu 0.

Zeitstempel: 32 Bit. Der in jedem RTP-Paket enthaltene Zeitstempel ist das zentrale Element, welches die Fähigkeit des Protokolls, Echtzeitdaten zu übertragen, ermöglicht. Dadurch kann ein Empfänger den Zeitpunkt relativ zu anderen erhaltenen Paketen bestimmen, wann die in diesem Paket enthaltenen Nutzdaten angezeigt oder abgespielt werden müssen. Der Zeitstempel bezieht sich auf den Moment, an dem das erste Byte der Nutzdaten in diesem Paket aufgezeichnet wurde. Der zugehörige Zeitgeber wird monoton und linear mit der Zeit erhöht. Die Auflösung muss hoch genug sein, um die Synchronisation mehrerer, in verschiedenen RTP-Sitzungen übertragenen Datenströme zu ermöglichen. Sie wird entweder im Profil oder im Verpackungsschema festgelegt. Mehrere Pakete können den gleichen Zeitstempel aufweisen, wenn sie zu einer logischen, zu einem Zeitpunkt aufgezeichneten Einheit, wie etwa ein Einzelbild in einem Videofilm gehören. Der Startwert wird wie bei der Sequenznummer zufällig gewählt.

SSRC: 32 Bit. Wie bereits erwähnt ist der SSRC Identifier das primäre Erkennungsmerkmal eines Teilnehmers einer RTP-Sitzung. Er wird zufällig gewählt; Konflikte, wenn zwei Teilnehmer den gleichen SSRC wählen, sollten bereits bei der Wahl so unwahrscheinlich wie möglich gemacht werden. Dies geschieht dadurch, dass ein neuer Teilnehmer an einer RTP-Sitzung erst eine gewisse Zeit die von den anderen Teilnehmern gesendeten Pakete empfängt und somit einen Großteil der bereits verwendeten SSRCs kennt und diese von seiner Wahl ausschließt.

CSRC Liste: 0 bis 15 Einträge zu je 32 Bit. Falls ein Mixer RTP-Pakete mehrerer Quellen zu einem RTP-Paket zusammen fasst, trägt er hier die SSRCs der ursprünglichen Quellen ein und setzt als SSRC dieses Paketes seinen eigenen ein. Falls das Paket aus mehr als 15 Quellpaketen zusammen gesetzt wurde, werden trotzdem nur maximal 15 Quellen aufgeführt. Falls mehrere Mixer kaskadiert auftreten, erscheinen in dieser Liste nur SSRCs der ursprünglichen Quellen, aber keine von den vorherigen Mixern.

2.1.6 Das RTP Control Protocol

Das RTP Control Protocol (RTCP) läuft parallel zur eigentlichen Datenübertragung mit RTP ab. Ebenso wie bei diesem Protokoll werden die Pakete mit Hilfe eines anderen Transportprotokolls übertragen. Bei einer Verwendung von UDP wird dabei die nächsthöhere Portnummer als für RTP verwendet. Für RTCP wird ein bestimmter Anteil der für die Datenübertragung mit RTP/RTCP reservierten Bandbreite verwendet, üblich ist ein Anteil von fünf Prozent.

Alle Teilnehmer einer Sitzung senden in gewissen Abständen RTCP-Pakete, die Information über den Versand und den Empfang von RTP-Paketen sowie Information über den Teilnehmer enthalten. Mit Hilfe dieser Informationen kann jeder Teilnehmer einen Überblick über alle Teilnehmer erhalten. Der Zeitabstand zwischen zwei RTCP-Paketen, das sogenannte RTCP-Intervall, wird dynamisch so bestimmt, dass die für RTCP verfügbare Bandbreite ganz ausgenutzt wird, aber auch keine Netzüberlastung verursacht wird.

Dazu ist es zunächst notwendig, dass jeder Teilnehmer einen Überblick über die Gesamtzahl der Teilnehmer besitzt. Neue Teilnehmer werden dabei durch gesendete RTP- oder RTCP-Pakete erkannt und in eine Liste bekannter Teilnehmer hinzugefügt. Aus dieser Liste wird ein Teilnehmer wieder gelöscht, wenn von ihm eine Meldung über das Verlassen der Sitzung erhalten wurde oder über einen längeren Zeitraum weder RTP- noch RTCP-Pakete empfangen wurden, was auf technische Probleme hinweist. Des Weiteren wird die durchschnittliche Größe eines RTCP-Paketes bestimmt, wobei in die Berechnung sowohl ein- als auch ausgehende Pakete eingehen. Danach wird die Bandbreite durch die Anzahl Teilnehmer und die durchschnittliche Größe eines RTCP-Pakets geteilt, wodurch die Sende-Frequenz für RTCP-Pakete bestimmt ist. Die RTCP-Intervall bis zum Versand des nächsten RTCP-Paketes ist der Kehrwert davon. Der erhaltene Wert wird dann noch mit einer Zufallszahl im Bereich 0.5 bis 1.5 multipliziert, um eine ungewollte Synchronisation mehrerer Teilnehmer zu vermeiden.

Insgesamt sind fünf verschiedene RTPC-Pakettypen zu unterscheiden:

- RTCP Sender Report-Pakete (SR) enthalten Informationen über RTP-Pakete, die ein Teilnehmer selbst gesendet hat und über RTP-Pakete, die er von anderen Teilnehmern empfangen hat.

- RTCP Receiver Report-Pakete (RR) enthalten Informationen über RTP-Pakete, die ein Teilnehmer von anderen Teilnehmern empfangen hat.

- RTCP Source Description-Pakete (SDES) transportieren textuelle Beschreibungen über Teilnehmer an der RTP-Sitzung. Sie beinhalten auch den CNAME.

- RTCP BYE-Pakete teilen mit, dass ein Teilnehmer die RTP-Sitzung verlassen hat.

- RTCP APP-Pakete dienen zur Übermittlung anwendungsspezifischer Informationen.

Es wird allerdings nie nur ein einziges RTCP-Paket verschickt, sondern immer mehrere zu einem sogenannten *Compound-Paket* zusammen gefasst, welches dann übertragen wird. Dieses ist eine einfache Aneinanderreihung einzelner RTCP-Pakete, ohne dass ein zusätzlicher Paketkopf hinzugefügt wird. Der Aufbau eines Compound Packet folgt folgendem Schema: Als erstes befindet sich darin ein RTCP Sender Report-Paket oder ein RTCP Receiver Report-Paket. Danach können weitere RTCP Receiver Report-Pakete folgen. Als nächstes enthält es mindestens ein RTCP SDES-Paket, darauf können anwendungsspezifische RTCP APP-Pakete folgen. Falls das Compound-Paket ein RTCP BYE-Paket enthält, ist dieses an letzter Stelle.

2.1.6.1 RTCP Sender Report-Paket

Ein Sender Report-Paket wird von jedem Sender von Nutzdaten als erstes Paket innerhalb eines zusammen gesetzten RTCP-Pakets übermittelt. Es enthält Informationen über von diesem Teilnehmer gesendete und von anderen Sendern empfangene RTP-Pakete. Innerhalb des Sender Report-Pakets lassen sich drei Teile unterscheiden: Als erstes enthält es einen festen Paketkopf, der dieses Paket als Sender Report-Paket identifiziert. Darauf folgt der namensgebende Sender Report. In ihm sind Daten über gesendete RTP-Pakete sowie weitere für Empfänger dieser Pakete wichtige Informationen. Als letztes folgen 0 bis 31 Reception Reports, in denen dieser Teilnehmer andere Sender über den Empfang ihrer RTP-Pakete informiert. Falls von mehr als 31 anderen Teilnehmern RTP-Pakete empfangen wurden, befinden sich diese Reception Reports in einem weiteren Paket, einem Receiver Report-Paket, welches im zusammen gesetzten RTCP-Paket dem Sender Report-Paket folgt.

Der Paketkopf eines RTCP Sender Report-Paketes beinhaltet folgende Felder:

Version: 2 Bit. Gibt die verwendetete Version des Protokolls an und ist die gleiche wie in RTP-Paketen, also 2.

Füllbytes (padding, P): 1 Bit. Dieses Feld entspricht ebenfalls dem im RTP-Paketkopf definierten Feld. Falls es eins ist, wurde das Paket um Füllbytes erweitert, damit die Länge in Bytes ein Vielfaches einer bestimmten Blocklänge ist, was für bestimmte Verschlüsselungsalgorithmen notwendig ist. Es werden immer komplette Compound-Pakete verschlüsselt, so dass Füllbytes nur an das letzte RTCP-Paket angehängt werden müssen.

V=2	P	RC	PT=SR=200	Länge
SSRC				
NTP Zeitstempel, höchstwertiges Wort				
NTP Zeitstempel, niedrigstwertiges Wort				
RTP Zeitstempel				
Anzahl gesendeter Pakete				
Anzahl gesendeter Bytes				
SSRC_1 (SSRC der ersten Quelle)				
Anteil verloren			Gesamtzahl verloren gegangener Pakete	
erweiterte höchste empfangene Sequenznummer				
Schwankungen der Übertragungsdauer				
letzter Sender Report				
vergangene Zeit seit letztem Sender Report				
weitere Reception Reports ...				

Abbildung 2.2: Das Format eines RTCP Sender Report-Paketes.

Anzahl Reception Reports (reception report count, RC): 5 Bit. Dieses Feld gibt an, wie viele Reception Reports in diesem Paket enthalten sind. Ein Wert Null ist dann möglich, wenn der Teilnehmer im letzten RTCP-Intervall keine RTP-Pakete empfangen hat. Falls von mehr als den hier maximal möglichen 31 Quellen RTP-Pakete im letzten RTCP-Intervall empfangen wurden, werden zusätzliche Receiver Report-Pakete versendet.

Pakettyp (PT): 8 Bit. Zur Unterscheidung der verschiedenen RTCP-Pakettypen enthält jedes RTCP-Paket dieses Feld. In einem Sender Report-Paket ist hier der Wert 200 eingetragen.

Länge: 16 Bit. In diesem Feld ist die Länge dieses RTCP-Paketes in 32 Bit-Wörtern minus eins gespeichert. Die Angabe beinhaltet auch eventuell vorhandene Füllbytes.

SSRC: 32 Bit. Der SSRC des Senders dieses Paketes.

Der Sender Report folgt auf den Paketkopf und enthält folgende Felder:

NTP-Zeitstempel: 64 Bit. Der NTP-Zeitstempel gibt an, zu welchem Zeitpunkt der Sender Report geschickt wurde. Falls der Sender keine Möglichkeit hat, die Zeit zu bestimmen, kann er die Zeitpunkt des Beginns der Teilnahme an der RTP-Sitzung als Nullpunkt annehmen und von dort aus hochzählen. Falls auch diese Möglichkeit nicht besteht, sendet er konstant 0. Das Zeitformat entspricht dem *Network Time Protocol* (*NTP*).

RTP-Zeitstempel: 32 Bit. Der RTP-Zeitstempel beschreibt den genau gleichen Zeitpunkt wie der NTP-Zeitstempel, allerdings in der Einheit wie die Zeitstempel in den RTP-Paketen. Mit den beiden Zeitstempeln zusammen hat der Empfänger die Möglichkeit, eine Abbildung von RTP-Zeitstempel in die Realzeit vorzunehmen; er kann somit für jedes RTP-Paket den Zeitpunkt der Aufnahme berechnen.

Anzahl gesendeter Pakete: 32 Bit. Hiermit übermittelt der Sender, wie viele RTP-Pakete er seit Beitritt zu der Sitzung geschickt hat.

Anzahl gesendeter Bytes: 32 Bit. Dieses Feld enthält die Anzahl Nutzdaten-Bytes, die der Sender seit Beitritt zur Sitzung übertragen hat.

Ein RTCP-Sender Report-Paket kann 0 bis 31 Reception Reports enthalten. Diese ermöglichen dem Nutzdaten-Sender, die Qualität der Datenübertragung einzuschätzen und unter Umständen Maßnahmen für eine Verbesserung einzuleiten. Dies könnte etwa in Form einer Verringerung der Datenrate durch eine geringere Abtastrate bei einer Audioübertragung oder durch eine kleine Bildauflösung bei einer Videoübertragung geschehen. Ein Reception Report ist folgendermaßen aufgebaut:

SSRC_n: 32 Bit. Der SSRC der Quelle, für deren gesendete RTP-Pakete dieser Reception Report ist.

Anteil verloren gegangener Pakete: 8 Bit. Dieses Feld enthält den Anteil verloren gegangener RTP-Pakete in diesem RTCP-Intervall. Der Wert berechnet sich aus der Anzahl verloren gegangener Pakete geteilt durch die Anzahl erwarteter Pakete innerhalb des RTCP-Intervalls. Dabei ist die Anzahl erwarteter Pakete definiert durch die Differenz zwischen höchster Sequenznummer eines RTP-Paketes in diesem RTCP-Intervall und höchster Sequenznummer eines RTP-Paketes im vorigen RTCP-Intervall. Die Anzahl verloren gegangener Pakete ist die Anzahl erwarteter Paket minus der Anzahl empfangener Pakete.

Gesamtzahl verloren gegangener Pakete: 24 Bit. Die Gesamtzahl verloren gegangener Pakete berechnet sich aus der Anzahl erwarteter Pakete minus der Anzahl tatsächlich erhaltener Pakete, einschließlich verspätet erhaltenen Paketen und Duplikaten.

Erweiterte höchste erhaltene Sequenznummer: 32 Bit. Die unteren 16 Bit enthalten die höchste erhaltene Sequenznummer eine RTP-Paketes, die oberen 16 Bit die Anzahl der Umläufe, also wie oft bereits die Sequenznummern die höchste mit 16 Bit darstellbare Zahl überschritten haben und wieder bei 0 begonnen haben.

Schwankungen der Übertragungsdauer: 32 Bit. Eine Schätzung der Varianz der Übertragungsdauer von RTP-Paketen in der gleichen Einheit wie die Zeitstempel dieser Pakete.

NTP-Zeitstempel des letzten erhaltenen Sender Reports: 32 Bit. Die mittleren 32 Bit des NTP-Zeitstempels aus dem letzten von dieser Quelle erhaltenen Sender Report.

Vergangene Zeit seit dem letzten erhaltenen Sender Report: 32 Bit. Falls bereits ein Sender Report erhalten wurde, enthält dieses Feld die seit dessen Erhalt vergangene Zeit in Einheiten zu 1/65536 Sekunden.

2.1.6.2 RTCP Receiver Report-Paket

RTCP Receiver Report-Pakete dienen sendenden Teilnehmern als Rückmeldung über die Ankunft ihrer RTP-Pakete bei anderen Teilnehmern. Sie werden von Teilnehmern gesendet, die selbst keine Nutzdaten übertragen, aber auch von Sendern, die mehr als 31 andere Nutzdaten-Sender empfangen. Ihr Aufbau entspricht bis auf den Pakettyp 201 statt 200 und dem fehlenden Sender Report einem Sender Report-Paket, weshalb an dieser Stelle auf eine Beschreibung der einzelnen Felder verzichtet werden kann.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```

V=2	P	RC	PT=RR=201	Länge
SSRC				
SSRC_1 (SSRC der ersten Quelle)				
Anteil verloren			Gesamtzahl verloren gegangener Pakete	
erweiterte höchste empfangene Sequenznummer				
Schwankungen der Übertragungsdauer				
letzter Sender Report				
vergangene Zeit seit letztem Sender Report				
weitere Reception Reports ...				

Abbildung 2.3: Das Format eines RTCP Receiver Report-Paketes.

2.1.6.3 RTCP Source Description-Paket

RTCP Source Description-Pakete liefern textuelle Informationen über Teilnehmer an der RTP-Sitzung. Ein Paket kann dabei Informationen über mehrere Teilnehmer enthalten, wobei die Informationen über einen Teilnehmer in einem SDES Chunk zusammen gefasst sind. Ein SDES Chunk enthält den SSRC des beschriebenen Teilnehmers sowie ein oder mehrere SDES Items, in denen die beschreibenden Zeichenketten enthalten sind. Dabei ist in jedem SDES Chunk das SDES Item enthalten, das den CNAME des Teilnehmers enthält. Andere SDES Items können seltener übertragen werden.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```

V=2	P	SC	PT=SDES=202	Länge
SSRC/CSRC_1				
SDES Items ...				
SSRC/CSRC_2				
SDES Items ...				
weitere SDES Chunks ...				

Abbildung 2.4: Das Format eines RTCP Source Description-Paketes.

Version, Füllbytes, Länge: Diese Felder haben dieselbe Bedeutung wie in den bereits beschriebenen RTCP-Pakettypen.

Anzahl Quellen (source count, SC): 5 Bit. Gibt an, wie viele SDES Chunks in diesem Paket enthalten sind. Während ein Teilnehmer an einer Sitzung wird nur Informationen über sich sendet und dementsprechend hier den Wert Eins einträgt, können Zwischensysteme mehrere Source Description-Pakete zusammen fassen, wodurch sich Werte größer Eins ergeben.

SSRC: 32 Bit. Der SSRC des Teilnehmers, der hier beschrieben wird.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+-------------------------------+
|      Typ      |     Länge     |                               |
+---------------+---------------+                               |
|                 beschreibende Zeichenkette                    |
|                                                               |
+---------------------------------------------------------------+
```

Abbildung 2.5: Das Format eines RTCP Source Description Item.

Typ: 8 Bit. Das Typfeld gibt an, um was für ein SDES Item es sich hier handelt. Es gibt dabei vordefinierte Werte für CNAME = 1, NAME = 2, EMAIL = 3, TELEFON = 4, ORT = 5, PROGRAMMNAME = 6, NOTIZ = 7. Für eigene Erweiterungen ist PRIVAT = 8 vorgesehen. In diesem Fall folgen auf das Längenfeld noch zwei weitere Felder: Die Präfixlänge und der Präfix. Das Letztere ist dabei eine Zeichenkette, die als Name für dieses SDES item dient, das Erstere ist die Länge dieses Namens.

Länge: 8 Bit. Die Länge der beschreibenden Zeichenkette.

Zeichenkette: 0 bis 255 Zeichen. Die beschreibende Zeichenkette.

2.1.6.4 RTCP BYE-Paket

Ein RTCP BYE-Paket teilt mit, dass einer oder mehrere Teilnehmer ab sofort nicht mehr an dieser RTP-Sitzung teilnehmen. Das Verlassen kann auf Anweisung des Benutzers geschehen, aber auch durch die Protokollinstanz selbst. Der zweite Fall ist zur Beseitigung von Problemen, wie die Verwendung desselben SSRC durch zwei Teilnehmer oder Schleifenbildung durch Zwischensysteme, gedacht.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+-+---------+-------------+---------------------------------+
|V=2|P|   SC    | PT=BYE=203  |             Länge               |
+---+-+---------+-------------+---------------------------------+
|                           SSRC/CSRC                           |
+---------------------------------------------------------------+
:                                                               :
:                   SSRC oder weitere CSRCs                     :
:                              ...                              :
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
```

Abbildung 2.6: Das Format eines RTCP Bye-Paketes.

Version, Füllbytes, Länge: Wie in den bereits beschriebenen RTCP-Pakettypen.

Pakettyp: 8 Bit. Hier ist für ein RTCP BYE-Paket der Wert 203 eingetragen.

Anzahl Quellen: 5 Bit. Die Anzahl Quellen, welche mit diesem BYE-Paket die Sitzung verlassen.

SSRC/CSRC-Liste: variable Anzahl 32 Bit-Wörter. Diese Liste enthält die SSRCs/CSRCs der Teilnehmer, deren Verlassen der Sitzung durch dieses Paket angezeigt werden soll. Wenn ein Endsystem eine RTCP BYE-Paket verschickt, trägt es nur seinen eigenen SSRC in die Liste ein. Falls ein Mixer seine Teilnahme an der Sitzung beendet, trägt er sowohl seinen eigenen SSRC, als auch alle CSRCs ein, die er zusammen mischt, in die Liste ein.

Nach der SSRC-Liste kann noch ein Grund für das Verlassen der Sitzung im RTCP BYE-Paket übermittelt werden. Sein Format ähnelt dem eines SDES Item, zuerst ein 8 Bit breites Feld für die Länge der Zeichenkette, die Zeichenkette folgt danach.

2.2 Das RTP Audio/Video-Profil

Das Real-time Transport-Protocol (RTP) zur Übertragung von Echtzeitdaten benötigt für seine Anwendung ein Profil, welches Parameter, Zuordnungen und optionale Erweiterungen definiert. Im Folgenden soll hier nun das wohl wichtigste und häufig eingesetzte *RTP-Profil für Audio- und Videokonferenzen mit minimaler Steuerung* [Grou96b] vorgestellt werden.

Wie bereits am Namen ersichtlich ist es für Anwendungen geeignet, welche Audio- oder Videodaten mit Hilfe von RTP übertragen. Das Profil spezifiziert keine zusätzlichen Dienste wie die Aushandlung von Übertragungsparametern oder dynamische Nutzdatentypen. Weiterhin bietet es keine Mechanismen zur Teilnehmerverwaltung in der RTP-Sitzung. Anwendungen, die unter diesem Profil arbeiten, müssen die von RTCP angebotenen Dienste zum Austausch von Teilnehmerinformationen genügen; zur Multicastgruppen-Verwaltung müssen andere Protokolle verwendet werden.

2.2.1 Veränderungen and RTP- und RTCP-Paketen

RTP-Pakete werden in diesem Profil verwendet wie in der Protokoll-Spezifikation beschrieben: Der Paketkopf besitzt genau ein Markierungsbit und sieben Bit für den Nutzdatentyp. Es wird nicht von der Möglichkeit Gebrauch gemacht, nach dem Standard-Paketkopf weitere Felder fester Länge zu definieren, auch legt es keine Paketkopf-Erweiterungen fest. Anwendungen, welche unter diesem Profil arbeiten, können jedoch Letzteres verwenden, so dass das entsprechende Erweiterungsbit in jedem Falle zu beachten ist. Bei RTCP-Paketen werden keine Veränderungen gegenüber dem Standard vorgenommen, es werden auch keine weiteren RTCP-Pakettypen definiert.

2.2.2 Nutzdatentypen

Die Zuordnung von Werten im Feld Nutzdatentyp in RTP-Paketen zu konkreten Datenformaten ist die entscheidende Festlegung in diesem Profil. Es werden jedoch nicht nur die entsprechenden Nummern festgelegt, sondern auch weitere Hinweise sowohl allgemeiner Natur, als auch für die einzelnen Datenformate gegeben. Zusätzlich gibt es noch Informationen, wie man weitere Datenformate registrieren kann. Für jedes Datenformat benötigt man noch ein Verpackungsschema, das beschreibt, wie die Daten in RTP-Pakete eingepackt werden.

2.2.3 Übertragung von Audiodaten

Das Audio/Video-Profil legt einige grundlegende Vorgehensweise für Audiodaten verschiedener Formate fest:

- Das Markierungsbit in einem RTP-Paket kann bei Kodierungen, welche keine Daten bei Stille übertragen, verwendet werden um anzuzeigen, dass dieses Paket das erste nach einer Stilleperiode ist. Für Kodierungen, welche auch bei Stille Daten senden, wird vorgeschlagen, das Markierungebit konstant auf Null zu setzen.
- Die Frequenz, mit der der RTP-Zeitgeber erhöht wird, soll unabhängig von der Anzahl der Audiokanäle und der Kodierung sein. Sie soll stattdessen der Abtastrate der Audiodaten entsprechen.
- Falls mehrere Audiokanäle verwendet werden, gibt das Profil eine Empfehlung ab, in welcher Reihenfolge die Audiodaten der einzelnen Kanäle angeordnet werden sollen. Dabei ist zu beachten, dass alle Daten, die zum gleichen Zeitpunkt aufgenommen wurden, in einem Paket enthalten sein müssen.
- Die verwendete Abtastrate sollte aus der folgenden Liste gewählt werden: 8000, 11025, 16000, 22050, 24000, 32000, 44100 und 48000 Hertz. Ein RTP-Paket sollte 20 ms Audiodaten enthalten, dieser Wert kann aber im Verpackungsschema geändert werden.

Nutzdaten-Typ	Kodierung	RTP-Frequenz in Hz
0	PCMU	8000
1	1016	8000
2	G.721	8000
3	GSM	8000
5	DVI4	8000
6	DVI4	16000
7	LPC	8000
8	PCMA	8000
9	G.722	8000
10	L16	44100
11	L16	44100
14	MPEG-1- oder MPEG-2-Audio	90000
15	G728	8000

2.2.4 Videodaten

Für Videodaten können keine allgemeinen Hinweise wie bei Audiodaten gegeben werden, da sich die einzelnen Datenformate zu sehr unterscheiden. Genaue Hinweise, wie die Daten in RTP-Pakete verpackt werden und das Markierungsbit verwendet wird, müssen daher dem Verpackungsschema entnommen werden.

Nutzdaten-Typ	Kodierung	RTP-Frequenz in Hz
25	CELL-B	90000
26	JPEG	90000
28	nv	90000
31	H.261	90000
32	MPEG-1- oder MPEG-2-Video	90000
33	MPEG-2-Transport-Strom	90000

2.3 MPEG-Video und dessen Übertragung mit RTP

2.3.1 MPEG-Video

Im Rahmen dieser Arbeit soll nur ein grober Überblick über MPEG gegeben werden, wie er für die Entwicklung der Module zur Übertragung von simuliertem MPEG-Video notwendig war. Eine ausführliche Beschreibung von MPEG findet sich in [Barq00].

MPEG dient zur verlustbehafteten Komprimierung von Computer-Videos. Der Name ist eine Abkürzung und steht für *Motion Pictures Experts Group*. Der ursprüngliche MPEG-1-Standard wurde 1993 von der *International Standards Organization* (*ISO*) verabschiedet. Inzwischen gibt es die Weiterentwicklungen MPEG-2, mit dem höhere Qualität und höhere Auflösungen möglich sind, sowie MPEG-4, das besonders auf Video-Übertragungen in relativ schmalbandigen Netzwerken ausgerichtet ist.

Die Grundidee bei MPEG-1-Video ist, dass mit einer konstanten Rate Einzelbilder dargestellt werden. Dabei werden die Einzelbilder nicht alle komplett gespeichert, sondern bei einigen nur die Differenzen zu anderen Bildern. Dies führt zu einer Unterscheidung von drei Bildtypen: Als erstes seinen die *Intra Frames* (*I-Frames*) erwähnt, die ein komplettes JPEG-komprimiertes Bild enthalten. Als zweites gibt es die *Predictive Frames* (*P-Frames*), bei sich die enthaltenen Daten auf einen vorherigen I- oder P-Frame beziehen und das Bild aus beiden zusammen gesetzt wird. Bei den *Bilinear Predictive Frames* (*B-Frames*) ist ein Bezug auf vorhergehende und auf nachfolgende I- und P-Frames möglich. Innerhalb eines Frames gibt es noch weitere Strukturen: Erstens die *Makroblöcke* und die darin enthalten so genannten *Slices*.

2.3.2 MPEG-Übertragung mit RTP gemäß RFC 2250

Der RFC 2250 [Grou98] legt fest, wie MPEG-1 und MPEG-2, sowohl Audio als auch Video mit RTP unter dem Audio/Video-Profil übertragen werden.

Bei der Übertragung von MPEG-Video ist ein einzelner Frame oftmals größer als die maximale Größe der Nutzdaten in einem RTP-Paket. In diesem Fall wird der Frame an den Slice-Grenzen auf mehrere Pakete aufgeteilt. Da ein MPEG-Decoder immer an Slice-Grenzen ansetzen muss, bedeutet dies eine Erleichterung, falls ein RTP-Paket eines Frames verloren gegangen ist. Das Markierungsbit in einem RTP-Paket bedeutet, dass dieses Paket das Ende des Frames beinhaltet, also das letzte Paket für diesen Frame ist.

Der RFC definiert für MPEG-Video einen zusätzlichen festen Paketkopf für RTP-Pakete, der sich im Nutzdaten-Bereich direkt an den RTP-Kopf anschließt.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
|   MBZ   |T|         TR        |AN|N|S|B|E|  P  |FBV| BFC |FFV| FFC |
```

Abbildung 2.7: Der zusätzliche Paketkopf für MPEG-1-Video.

MBZ: 5 Bit. Ist in dieser Spezifikation unbenutzt und muss auf 0 gesetzt werden.

MPEG-2 (two, T): 1 Bit. Ist auf Eins gesetzt, falls eine zusätzliche, optionale MPEG-2-Video-spezifische Paketkopf-Erweiterung folgt.

Zeitliche Referenz (temporal reference, TR): 10 Bit. Eine zeitliche Referenz für dieses Bild innerhalb einer *Group of Pictures*.

Aktives N-Bit (AN): 1 Bit Ist auf Eins, falls das folgende N-Bit benutzt wird (nur für MPEG-2 Video).

N-Bit (N): 1 Bit. Falls das N-Bit benutzt wird, bedeutet eine Eins, dass der zuvor übertragene Bild-Kopf nicht zur Rekonstruktion diese Kopfes verwendet werden kann.

Sequenz-Kopf vorhanden (S): 1 Bit. Auf Eins gesetzt, falls in diesem Paket ein MPEG Videosequenz-Kopf enthalten ist.

Beginn eines Slice (BS): 1 Bit. Dieses Bit ist gesetzt, falls direkt nach eventuell vorhandenen anderen Kopfdaten oder am Beginn der Nutzdaten ein Slice-Kopf enthalten ist.

Ende eines Slice (ES): 1 Bit. Dieses Bit ist gesetzt, falls das letzte Byte der Nutzdaten das Ende eines Slice ist.

Bildtyp (picture type, PT): 3 Bit. Der Frame-Typ: I=1, P=2, B=3

full_pel_backward_vector (FBV): 1 Bit.

backward_f_code (BFC): 1 Bit.

full_pel_forward_vector (FFV): 1 Bit.

forward_f_code (FFC): 1 Bit. Diese Bits werden aus dem aktuellen Bild-Kopf entnommen.

Für MPEG-2 Video gibt es noch die bereits oben erwähnte Kopf-Erweiterung; für MPEG-Audio gibt es ebenfalls einen Kopf, der am Beginn des Nutzdaten-Bereichs in RTP-Paketen enthalten ist. Auf beide soll an dieser Stelle mangels Relevanz für diese Arbeit nicht eingegangen werden.

2.4 Die Simulationsumgebung OMNeT++

OMNeT++, das *Objective Modular Network Testbed in C++* [Varg00], ist eine modulare, objekt-orientierte, diskrete Ereignis-Simulationsumgebung. Sie ist geeignet, eine Vielzahl von Systemen zu modellieren, als Beispiele seien Kommunikationsprotokolle, Computernetzwerke und Multiprozessorsysteme genannt. Ein mit OMNeT++ modelliertes System besteht dabei aus hierarchisch ineinander verschachtelten Modulen, die mittels Nachrichten miteinander kommunizieren. Das besondere Merkmal einer diskreten Ereignissimulation ist, dass Zustandsübergänge an bestimmten Zeitpunkten geschehen und ohne dass dabei Zeit vergeht. Im Zeitraum zwischen zwei aufeinander folgenden Zustandsübergängen geschieht nicht Relevantes für die Simulation. Eine Simulation läuft ab, indem Module Nachrichten versenden, die von der Simulationsumgebung in eine Warteschlange, das sogenannte Future Event Set (FES), eingereiht werden und dann zu einem bestimmten Zeitpunkt an das Zielmodul weitergeleitet werden. Wenn keine Nachrichten mehr in der Warteschlange sind, ist die Simulation beendet. Es kann durch den Benutzer auch eine Endbedingung angegeben werden, so dass die Simulation endet, wenn entweder die Bedingung erfüllt oder die Warteschlange leer ist.

2.4.1 Aufbau eines Modells

Ein Modell besteht aus mehreren Modulen, wobei zwei verschiedene Modultypen zu unterscheiden sind: Als erstes seien die *einfachen Module* (*simple modules*) genannt. Sie enthalten die eigentliche Anwendungslogik und werden als C++-Klassen implementiert. Module des zweiten Typs, die sogenannten *zusammen gesetzten Module* (*compound modules*), enthalten hingegen meist keinen eigenen Quellcode. Sie dienen dazu, eine logische Struktur zu bilden, indem sie andere Module, sowohl einfache als auch zusammen gesetzte, enthalten können. Die Schachtelungstiefe ist dabei unbegrenzt. Zur Kommunikation der Module dienen *Gatter* (*gates*), wobei jeweils zwei durch eine *Verbindung* (*link*) einander zugeordnet werden. Über diese Verbindungen erfolgt dann der Nachrichtenaustausch zwischen den Modulen.

2.4.2 Einfache Module

Wie bereits oben erwähnt wird ein einfaches Modul in Form einer C++-Klasse implementiert. Dazu wird eine neue Klasse von `cSimple` abgeleitet. Um die Anwendungslogik zu implementieren, bestehen zwei verschiebene Möglichkeiten. Als erstes sei hier die `activity()`-Methode genannt. Sie wird von der Simulationsumgebung nach der Initialisierung des Moduls aufgerufen und genau einmal durchlaufen und muss selbst die ankommenden Nachrichten entgegennehmen. Die andere Variante ist die `handleMessage()`-Methode. Sie wird jedesmal aufgerufen, wenn das Modul eine Nachricht erhält, wobei die Nachricht als Parameter übergeben wird. Welche Variante für ein Modul verwendet wird, hängt von der konkreten Aufgabe ab. Die `activity()`-Methode ist dann von Vorteil, wenn viele Zustandsinformationen behalten werden müssen, viele Zähler mit Hilfe von `wait()`-Anweisungen realisiert werden oder paralleles Verhalten mehrerer Module simuliert werden soll. Die `handleMessage()`-Methode ist angemessen, wenn das Modul wenig Zustandsinformationen speichern muss oder wenn aus einem Zustand viele verschiedene Folgezustände erreicht werden können, wie dies bei endlichen Automaten (*finite state machines*, *FSM*) der Fall ist. In einem Modell können beide Typen verwendet werden, eine Modulklasse implementiert aber immer genau eine der beiden Varianten.

2.4.3 Zusammen gesetzte Module

Ein zusammen gesetztes Modul dient als Behälter für weitere Module. Sie ermöglichen einerseits, die logische Struktur eines Modells darzustellen, andererseits sind sie sehr hilfreich, wenn man vorhandene Module flexibel einsetzen möchte. Enthaltene Module lassen sich parametrisieren, so dass die genaue Zusammensetzung erst zur Laufzeit festgelegt wird.

2.4.4 Gatter und Verbindungen

Gatter dienen Modulen als unidirektionale Sende- oder Empfangseinheiten für Nachrichten. Es können sowohl Einzelgatter als auch Gattervektoren verwendet werden, wobei die Größe eines Gattervektors parametrisiert werden kann. Verbindungen haben ein Ausgangsgatter als Startpunkt und ein Eingangsgatter als Endpunkt. Eine Verbindung überträgt Nachrichten normalerweise ohne Verzögerungen auf einmal. Es können allerdings auch eine Übertragungszeit, eine Bitfehlerrate und eine Datenrate angegeben werden, was vor allem bei der Modellierung von Computernetzwerken hilfreich ist.

2.4.5 Nachrichten

Nachrichten dienen zur Kommunikation zwischen Modulen. Eine Nachricht ist eine Instanz der Nachrichtenklasse `cMessage` oder einer davon abgeleiteten Klasse. Eigene Nachrichten können eine Instanz einer dieser Klassen sein, wobei eigene Daten in Form von benannten Parametern hinzugefügt werden können. Diese Variante bedeutet weniger Implementierungsaufwand und lässt sich schnell erweitern. Die Verwendung wirkt sich allerdings negativ auf die Geschwindigkeit der Simulation aus. Die zweite Möglichkeit ist das sogenannte *message subclassing*, bei der eigene Nachrichtenklassen direkt oder indirekt von `cMessage` abgeleitet werden und diese zur Nachrichtenübermittlung verwendet werden. Dies bietet neben einer höheren Simulationsgeschwindigkeit auch noch den Vorteil von Typsicherheit.

Nachrichten werden von der Simulationsumgebung in der Zukunftsereignis-Warte-Schlange abgelegt. Darin sind sie zunächst nach ihrer Ankunftszeit am Zielmodul sortiert, falls mehrere Nachrichten zur gleichen Zeit an einem Modul ankommen, wird die Nachricht mit der höheren Priorität zuerst bearbeitet.

2.4.6 Die Klassenbibliothek

Neben den bereits oben erwähnten Klassen für die Modellierung bietet OMNeT++ noch eine ganze Reihe weiterer Klassen zur Unterstützung von Entwicklern an. Neben eher allgemeinen Klassen für Warteschlagen, Mengen und Feldern bietet es auch noch Klassen zur Erstellung von Statistiken über die ablaufende Simulation.

Als Besonderheit bietet es für eigene und davon abgeleitete Klassen eine Speicherverwaltung an, die es dem Benutzer ermöglicht, einen Überblick über vorhandene Objekte zu erhalten.

2.4.7 Weitere Informationen

Dies soll jetzt für eine Einführung in die Simulationsumgebung OMNeT++ genügen, weitere Informtionen findet man im Benutzerhandbuch [Varg00].

3. Entwurf

3.1 Anforderungen

Der Entwurf der Modulstruktur zur Simulation des Real-time Transport Protocol unter OMNeT++ stellt den Hauptteil dieser Arbeit dar. Dabei ist besonders zu berücksichtigen, dass dieses Modell als Bestandteil zukünftiger Arbeiten verwendet werden soll. Dies erfordert, dass das Modell flexibel, leicht erweiterbar und in verschiedenen Anwendungsszenarien einsetzbar ist.

Die Flexibilität besteht darin, dass das Modell für Unicast und Multicast geeignet ist, verschiedene Datenformate übertragen kann und nicht auf ein Profil festgelegt ist. Außerdem soll sich das Modell so weit notwendig zur Laufzeit selbst konfigurieren, so dass auch größere Netzwerke ohne allzu großen Aufwand simuliert werden können.

Das Modell soll einfach für weitere Profile und Datenformate erweiterbar sein, ohne dass vorhandener Quellcode verändert werden muss. Ebenso sollen RTP-Zwischen-Systeme, die im Rahmen dieser Arbeit nicht behandelt werden, in das Modell integrierbar sein.

3.2 Das Anwendungsmodul RTPApplication

Obwohl RTP normalerweise direkt in eine Anwendung integriert wird, wurde hier zugunsten der Wiederverwendung des Modells ein anderer Weg gewählt: Die Anwendung ist als separates Modul realisiert, das über eine Schnittstelle mit dem Modul für die Protokollverarbeitung kommuniziert. Dadurch ist es möglich, das RTP-Modul mit verschiedenen Anwendungsmodulen zu verwenden.

Die Steuerung des Anwendungsmoduls erfolgt durch OMNeT-Parameter, wodurch das Simulationsverhalten beim Aufbau des Netzwerks, in der `omnetpp.ini`-Datei oder zur Laufzeit durch den Benutzer festgelegt werden kann.

commonName: Hiermit wird der CNAME für diesen Teilnehmer festgelegt.

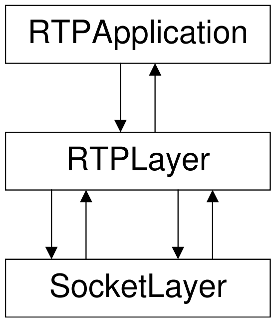

Abbildung 3.1: Die Schichten des RTP-Modells.

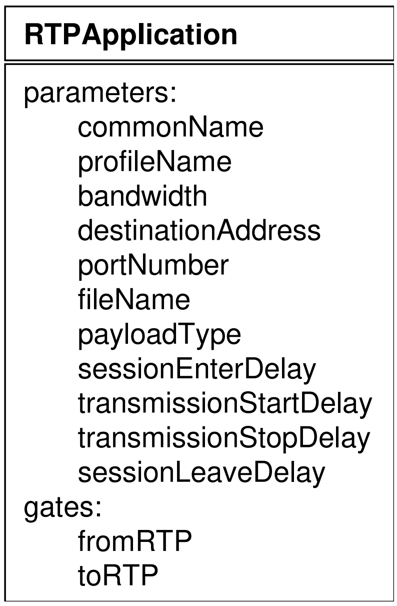

Abbildung 3.2: Das Modul RTPApplication.

profileName: Das Profil, unter welchem die Übertragung stattfindet, wird mit diesem Parameter festgelegt. Der Name muss dem Namen eines Profil-Moduls entsprechen.

bandwidth: Die Bandbreite (in Byte pro Sekunde), welche für die Datenübertragung mit RTP zur Verfügung steht.

destinationAddress: Hiermit wird die Netzwerk-Zieladresse der RTP-Übertragung festgelegt. Es kann sich dabei um die Unicast-Adresse des Zielsystems oder die Adresse einer Multicast-Gruppe handeln.

portNumber: Falls nicht des Standardport des Profils verwendet werden soll, kann hiermit die Portnummer festgelegt werden. Normalerweise wird für RTP eine gerade Portnummer, für RTCP die nächsthöhere verwendet.

fileName: Das Modul `RTPApplication` kann sowohl als Sender als auch als Empfänger fungieren. Im Parameter `fileName` kann der Name einer `.gdf`-Datei angegeben werden, welche übertragen werden soll. Falls der Parameter eine leere Zeichenkette enthält, arbeitet das Modul nur als Empfänger.

payloadType: Mit diesem Wert wird RTP mitgeteilt, von welchem Format die zu übertragenen Daten sind.

sessionEnterDelay: Dieser Wert gibt an, wie lange der Teilnehmer warten soll, bevor er der RTP-Sitzung beitritt. Da in der IP-Suite zur Zeit nur statisches Multicast-Routen möglich ist und der Socketlayer eine Fehlermeldung liefert, wenn Pakete an einen noch nicht geöffneten UDP-Port ankommen, ist hier nur ein Wert von 0 Sekunden möglich.

transmissionStartDelay: Nachdem das Anwendungsmodul die Rückmeldung über den erfolgreichen Beitritt zur RTP-Sitzung erhalten hat, wartet es den hier festgelegten Zeitraum, bevor es die Übertragung der Nutzdaten anweist. Reine Empfänger ignorieren diesen Wert.

transmissionStopDelay: Eine Übertragung von Nutzdaten endet entweder auf Anweisung durch das Anwendungsmodul oder wegen Erreichen des Dateiendes. Mit diesem Wert wird festgelegt, nach welcher Zeit die Anwendung die Übertragung beenden soll, falls noch nicht das Dateiende erreicht wurde. Ein Wert von 0 bedeutet, dass die Anwendung keine Anweisung zum Beenden der Übertragung geben soll, sondern auf das Erreichen des Dateiendes warten soll. Reine Empfänger ignorieren den hier angegebenen Wert.

sessionLeaveDelay: Dieser Wert dient zur Steuerung des Verlassens der RTP-Sitzung. Bei einem Sender bezieht sich die hier angegebene Zeit auf das Ende der Nutzdatenübertragung, bei reinen Empfängern auf die Rückmeldung über den Beitritt zur Sitzung.

3.3 Das RTP-Schicht-Modul RTPLayer

Das Modul `RTPLayer` ist ein zusammen gesetztes Modul und dient in erster Linie dazu, das Modell zu strukturieren, indem es die Module für die Protokoll-Funktionen zusammen fasst.

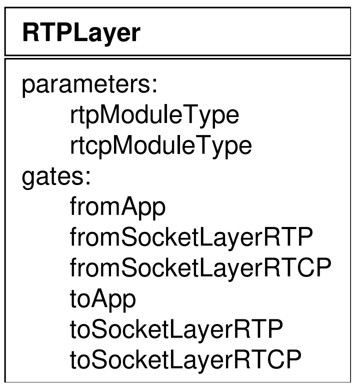

Abbildung 3.3: Das Modul RTPLayer.

rtpModuleType: Mit diesem Parameter wird festgelegt, welche konkrete Implementierung des RTP-Moduls verwendet werden soll.

rtcpModuleType: Ebenso wie der Modultyp für RTP wird auch der Modultyp für RTCP parametrisiert.

3.4 Das RTP-Modul

Das RTP-Modul bildet das zentrale Element im Modul `RTPLayer`. Es steuert das Verhalten der übrigen Module im `RTPLayer` und dient zur Kommunikation mit der Anwendung. Ebenso bildet es die Verbindung zum Socketlayer, um RTP-Pakete an das Netzwerk zu versenden oder von diesem zu empfangen. Nicht seine Aufgabe ist jedoch das Erstellen von ausgehenden oder das Verarbeiten von ankommenden RTP-Paketen. Dies wird von den Nutzdaten-Modulen erledigt.

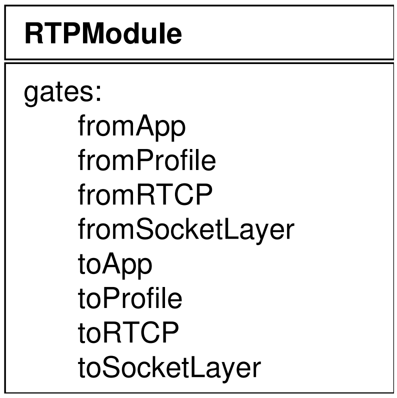

Abbildung 3.4: Das RTP-Modul.

Es lassen sich folgende Aufgabenbereiche für das RTP-Modul unterscheiden:

- Beitritt zu einer RTP-Sitzung
- Steuerung der Datenübertragung
- Versand und Empfang von RTP-Paketen
- Verlassen der RTP-Sitzung

Der Beitritt zu einer RTP-Sitzung ist dabei immer der erste Schritt. Ausgelöst wird er durch eine Nachricht, welche das RTP-Modul von der Anwendung erhält. An dieser Stelle kommt die Flexibilität zutage, dass das Modell nicht auf ein RTP-Profil festgelegt ist, sondern zur Laufzeit entschieden wird, unter welchem Profil das System arbeitet. Dazu wird vor dem Beitritt ein Profil-Modul des von der Anwendung gewünschten Typs erzeugt und mit dem RTP-Modul verbunden. Danach können das neu erzeugte Profil-Modul, die darunterliegende Transportschicht und das RTCP-Modul initialisiert werden und sobald dies erfolgreich ausgeführt ist, der Anwendung der Beitritt zu der RTP-Sitzung gemeldet werden. Die Steuerung der Datenübertragung sowie der Versand und Empfang von RTP-Paketen bedeutet für das RTP-Modul in erster Linie das Weiterleiten von Paketen zwischen Anwendung, Profil-Modul, RTCP-Modul und Transportschicht. Das Verlassen der RTP-Sitzung ist das Gegenstück zum Beitritt. Das Profil-Modul wird abgebaut, Transportschicht und RTCP-Modul beenden ihre Arbeit und die Anwendung über das Verlassen der Sitzung informiert.

3.5 Das RTCP-Modul

Das RTCP-Modul ist für die Erzeugung und Verarbeitung von RTCP-Paketen zuständig. Um diese Tätigkeiten erledigen zu können, verwaltet es alle notwendigen Informationen, wie etwa die Liste der Teilnehmer an der RTP-Sitzung. Die Informationen über ein- und ausgehende RTP-Pakete erhält es vom RTP-Modul. Es hat eine eigene Verbindung zum Socketlayer, um RTCP-Pakete selbst senden und empfangen zu können.

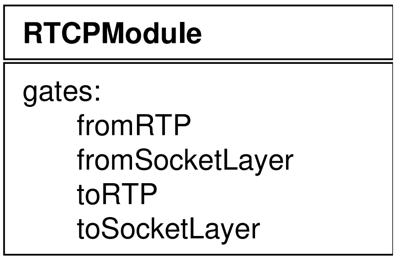

Abbildung 3.5: Das RTCP-Modul.

Die Trennung von RTP- und RTCP-Modul soll dabei in erster Linie die logische Trennung der beiden Protokolle und ihren parallelen Ablauf darstellen. Dadurch ist es außerdem möglich, RTP- und RTCP-Modul unabhängig voneinander weiter zu entwickeln, im Rahmen der Simulation etwa im Bezug auf statistische Auswertungen des Protokollverhaltens. Das RTCP-Modul besitzt keine Parameter, sondern wird mit Nachrichten vom RTP-Modul initialisiert.

3.6 Das Profil-Modul

Das Profil-Modul repräsentiert ein RTP-Profil. Dies beinhaltet das Setzen bestimmter Parameterwerte wie etwa der zu verwendenden Standard-Portnummer, falls durch die Anwendung keine angegeben wurde. Bei bestimmten Profilen gehört dazu auch die Veränderung von RTP-Paketen, etwa durch Hinzufügen zusätzlicher Paketkopf-Felder nach dem Standard-Kopf. Da die zweite Aufgabe ein Durchlaufen aller ein- und ausgehenden RTP-Pakete durch das Profil-Modul nötig macht, ist es zwischen den Nutzdaten-Modulen, welche RTP-Pakete erzeugen oder verarbeiten, und dem RTP-Modul angeordnet. Da diese Module erst bei Bedarf erzeugt werden, übernimmt das Profil-Modul auch deren Verwaltung, was Erzeugung, Steuerung und Abbau beinhaltet.

Ein Profil-Modul erzeugt ein Sender-Modul, wenn das Endsystem mit einer Datenübertragung beginnt und baut es wieder ab, wenn die Übertragung beendet ist. Ein Endsystem kann maximal einen Datenstrom senden. Ein Empfänger-Modul wird erzeugt, wenn erstmals ein Paket von der Quelle empfangen wird. Die maximale Anzahl Empfänger-Module wird durch die Größe der Gatter-Vektoren bestimmt. An dieser Stelle kommt auch der einzige Parameter des Profil-Moduls, autoOutputFileNames, zum Einsatz. Er bestimmt, wie die Namen der Dateien, in denen die Empfänger-Module die empfangenen Nutzdaten abgespeichern, vergeben werden. Falls dieser Parameter auf den Wert `true` gesetzt ist, legt das Profil-Modul die Namen fest, wobei eine Zeichenkette `id` + Modul-Id des Empfänger-Moduls + `.sim` verwendet wird. Da die Modul-Ids innerhalb einer Simulation einzigartig sind, schreibt jedes

Empfänger-Modul in eine eigene Datei. Falls der Parameter auf den Wert `false` gesetzt wird, muss der Parameter `outputFileName` für jedes Empfänger-Modul separat festgelegt werden, entweder durch die `omnetpp.ini`-Datei oder durch den Benutzer zur Laufzeit. Eine Festlegung in der Netzwerk-Definition ist nicht vorgesehen, da die Empfänger-Module dynamisch erzeugt werden.

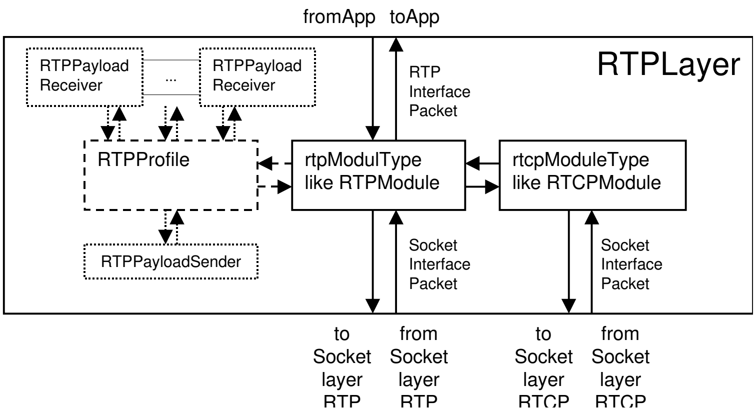

Abbildung 3.6: Die Module im RTP-Layer.

Das Modul selbst ist beim Start der Simulation noch nicht vorhanden, sondern wird erst mit dem Beitritt zu einer RTP-Sitzung durch das RTP-Modul erzeugt und mit diesem verbunden.

3.7 Sender- und Empfänger-Module für Nutzdaten

Für jedes verwendete Datenformat benötigt die Simulation geeignete Module zum Erzeugen und Verarbeiten der RTP-Pakete. Diese Sender- und Empfänger-Module werden dynamisch durch das Profil-Modul erzeugt. Folgende Aufgaben hat dabei ein Sender-Modul zu erfüllen: Öffnen der Quelle, Lesen der Daten aus der Quelle, Erzeugen von RTP-Paketen, Weiterleiten dieser Pakete an das Profil-Modul und Schließen der Quelle. Dabei ist zu bedenken, dass im Rahmen einer Simulation nur Dateien als Quellen in Frage kommen, da der Ablauf der Simulation nicht in Echtzeit erfolgt. Ein Empfänger-Modul kümmert sich um die Verarbeitung der empfangenen RTP-Pakete von einer Quelle. Im Rahmen der Simulation wird dies üblicherweise darin bestehen, die empfangenen Nutzdaten in einer Datei abzulegen. Es übernimmt im Rahmen der Simulation auch die wichtige Aufgabe, das Verhalten von RTP zu protokollieren. Dies beinhaltet das Erkennen von Paketverlusten und Vertauschungen, aber auch das Festhalten des zeitlichen Verhaltens des Protokolls besonders im Bezug auf Schwankungen der Übertragungsdauer. Die Art der Protokollierung kann frei gewählt werden. Bei einem geeigneten Dateiformat können diese Angaben direkt in die Ausgabedatei geschrieben werden, es kann aber auch separate Datei mit den statistischen Angaben erzeugt werden. Ebenso besteht die Möglichkeit, die von von OMNeT++ angebotene Speicherung von statistischen Daten mit Hilfe der Klasse `OutputVector` zu verwenden.

4. Implementierung

In diesem Kapitel werden die Klassen vorgestellt, mit denen RTP simuliert wird. Diese lassen sich dabei in drei Gruppen einteilen: Als erstes seien hier die Klassen für die im vorigen Kapitel entworfenen Module genannt. Als zweites gibt es Klassen, welche Nachrichten darstellen, die zwischen den Modulen ausgetauscht werden. Darüber hinaus werden auch die Hilfsklassen für die strukturierte Speicherung von Daten vorgestellt.

4.1 Die Modul-Klassen

4.1.1 Die Klasse RTPApplication

Die Klasse `RTPApplication` ist eine Implementierung des Moduls `RTPApplication`. Sie dient in erster Linie dazu, die Funktionalität der RTP-Module zu testen. Eine Instanz dieser Klasse kann sowohl als Sender als auch als Empfänger dienen, abhängig davon, ob der OMNeT-Parameter `fileName` einen Dateinamen enthält oder nur eine leere Zeichenkette ist.

Das Verhalten wird komplett in der `activity()`-Methode implementiert. Es beinhaltet das Auslesen der Parameter, den Beitritt zur Sitzung und Warten auf die Rückmeldung. Falls die Instanz als Sender tätig ist, wird nach der im Parameter `transmissionStartDelay` angegebenen Zeitdauer die Übertragung der Datei angewiesen. Die Übertragung endet, wenn das Ende der Datei erreicht ist oder der im Parameter `transmissionStopDelay` angegebene Zeitraum seit Beginn der Übertragung verstrichen ist, worauf die Anwendung die Übertragung beendet. Danach wartet der Sender die in `sessionLeaveDelay` angegebene Zeit, bevor er die Sitzung verlässt. Bei einem reinen Empfänger beginnt dieser Zeitraum mit der Rückmeldung über den Beitritt zur RTP-Sitzung.

4.1.2 Die Klasse RTPEndsystemModule

Die Klasse `RTPEndsystemModule` dient als Implementierung des RTP-Moduls in einem RTP-Endsystem. Sie übernimmt einen Großteil der Steuerungsfunktionen in der RTP-Schicht, besonders beim Beitritt und Verlassen einer Sitzung, aber auch bei der Datenübertragung.

Der Empfang eingehender Nachrichten erfolgt in der `handleMessage()`-Methode. In ihr findet keine Verarbeitung statt, diese wird je nach Herkunft der Nachricht an die Methoden `handleMessageFromApp()`, `handleMessageFromProfile()`, `handleMessageFromSocketLayer()` oder `handleMessageFromRTCP()` delegiert. Innerhalb dieser Methoden findet dann eine Unterscheidung nach Typ der Nachricht statt, die eigentliche Verarbeitung erfolgt dann in Typ-spezifischen Methoden.

Der Beitritt zu einer Sitzung läuft folgendermaßen ab: Auf eine entsprechende Nachricht von der Anwendung wird die Methode `enterSession()` aufgerufen. Darin werden die übergebenen Parameter ausgelesen, der Port bestimmt und die Methode `resolveMTU()` aufgerufen, mit der die maximale Paketgröße bestimmt wird. In dieser Implementierung wird immer UDP und IP über Ethernet angenommen und der entsprechende Wert zurückgegeben. Danach wird das Profil-Modul in der Methode `createProfile()` erzeugt und in der Methode `initializeProfile()` initialisiert. Die Rückmeldung vom Profil-Modul führt dann zu einem Aufruf von `profileInitialized()`, das dann `createServerSocket()` aufruft, worin dann ein Socket durch eine Nachricht an den Socketlayer beantragt wird. Die Rückmeldung führt dann zu einem Aufruf von `createClientSocket()`. Wenn auch der zweite Socket erzeugt ist wird der Socket verbunden. Wenn dies erfolgt ist, kann das RTCP-Modul in `initializeRTCP()` initialisiert werden. Das RTCP-Modul liefert daraufhin ein Rückmeldung, woraufhin das RTP-Modul eine Rückmeldung an die Anwendung schickt.

Die Steuerung der Datenübertragung durch die Anwendung und Rückmeldungen durch die Nutzdaten-Module werden in den entsprechenden Methoden im `RTPEndsystemModule` behandelt. Sie nehmen meist nur eine Umsetzung des Nachrichten-Typs vor. Eine Ausnahme bildet die Methode `transmissionStarted()`. In ihr wird neben der Rückmeldung an die Anwendung auch noch eine Nachricht an das RTCP-Modul gesendet, um dieses über den Beginn einer Datenübertragung zu informieren.

Bei der Datenübertragung muss das `RTPEndsystemModule` die Pakete vom Socketlayer an das Profil-Modul, welches diese wiederum an die Nutzdaten-Module überreicht, beziehungsweise anders herum weiterleiten. Dies geschieht in den Methoden `dataIn()` und `dataOut()`. Das RTP-Modul muss allerdings das RTCP-Modul über den Datenverkehr informieren, daher werden Kopien aller Pakete an das RTCP-Modul geschickt.

Das Verlassen einer Sitzung, ausgelöst durch eine Nachricht von der Anwendung, beinhaltet das Löschen des Profil-Moduls, Schließen der Sockets und Informieren des RTCP-Moduls. Dieses sendet dann eine Rückmeldung, woraufhin das RTP-Modul eine Rückmeldung an die Anwendung liefert.

4.1.3 Die Klasse RTCPEndsystemModule

Die Klasse `RTCPEndsystemModule` bildet das RCTP-Modul in einem RTP-Endsystem. Sie stellt alle für eine Simulation notwendigen Funktionen von RTCP zur Verfügung. Dies beinhaltet das Erstellen der Sockets, das Erzeugen, das Versenden, das Empfangen und das Verarbeiten von RTCP-Paketen. Die Sitzungssteuerung erfolgt durch das RTP-Modul, welches das RTCP-Modul anweist, einer Sitzung beizutreten oder sie zu verlassen. Beides wird nach erfolgreicher Ausführung durch die Klasse

`RTCPEndsystemModule` bestätigt. Sonstiger Nachrichtenaustausch zwischen diesen beiden Modulen erfolgt immer unbestätigt vom RTP-Modul ausgehend. Dazu gehört einerseits die Information, dass dieses RTP-Endsystem eine Datenübertragung gestartet hat, außerdem wird von jedem gesendeten und empfangenem RTP-Paket eine Kopie an das RTCP-Modul gesendet.

Wie auch bei der Klasse `RTPEndsystemModule` wird zur Nachrichtenverarbeitung die `handleMessage()`-Methode verwendet, in der eingehende Nachrichten zunächst nach ihrer Herkunft unterschieden werden, wobei es beim RTCP-Modul drei Möglichkeiten gibt: Vom RTP-Modul, vom Socketlayer oder vom RTCP-Modul selbst. Als Nächstes erfolgt eine Unterscheidung nach Art der Nachricht. Die Initialisierungsnachricht vom RTP-Modul führt zu einem Aufruf der Methode `initializeRTCP()`, welche die übergebenen Parameter ausliest und das Erzeugen der Sockets anstößt. Dies erfolgt in gleicher Weise wie in der Klasse `RTPEndsystemModule`. Danach setzt das RTCP-Modul eine Nachricht an sich selbst in die Warteschlange, der Zeitpunkt der Ankunft wird dabei auf jetzt plus ein halbes RTCP-Intervall gesetzt. Bis dahin werden RTCP-Pakete von anderen Teilnehmern empfangen, um Aufschluss über bereits verwendete SSRCs zu erhalten. Sobald diese Eigennachricht erhalten worden ist, wählt sich das Endsystem einen SSRC, der verschieden ist von den bereits bekannten. Als Nächstes wird in der Methode `scheduleInterval()` das neue RTCP-Intervall bestimmt und eine Nachricht an sich selbst gesendet. Deren Ankunftszeit ist dann der Moment, an dem dieses Endsystem ein RTCP-Paket sendet. Damit ist die Initialisierung abgeschlossen und es erfolgt eine Rückmeldung an das RTP-Modul.

Das weitere Verhalten der Klasse `RTCPEndsystemModule` wird durch die Eigennachrichten gesteuert: Immer wenn diese ankommen, wird ein RTCP-Paket erzeugt und verschickt und außerdem das RTCP-Intervall neu berechnet und eine neue Eigennachricht in die Warteschlange gesetzt.

Die für die Erzeugung der RTCP-Pakete notwendigen Daten werden aus den vom RTP-Modul kopierten und an das RTCP-Modul weitergeleiteten RTP-Paketen extrahiert. Informationen über weitere an der Sitzung beiteiligte System erhält das Modul durch eingehende RTCP-Pakete. Für die Verarbeitung gibt es die Methoden `processIncomingRTPPacket()`, `processOutgoingRTPPacket()` und `processIncomingRTCPPacket()`. Die zu versendenden RTCP-Pakete werden in der Methode `createPacket()` erzeugt.

Informationen über sich den Teilnehmer selbst werden in einer Instanz der Klasse `RTPSenderInfo` gespeichert, Informationen über andere Teilnehmer in Instanzen der Klasse `RTPReceiverInfo`. Alle zusammen werden im `cArray participantInfos` abgelegt, der Zugriff erfolgt über den Namen, der einer Zeichenkette in Hexadezimalnotation des SSRC entspricht.

4.1.4 Die Klasse RTPProfile

Die Klasse `RTPProfile` dient als Oberklasse für konkrete Profil-Klassen. Sie stellt insbesondere die Funktionen bereit, die im Rahmen des RTP-Modells vom Profil-Modul erledigt werden müssen, im Besonderen die Erzeugung und Verwaltung von Nutzdaten-Modulen. Der Empfang eingehender Nachrichten erfolgt in der Methode `handleMessage()`, welche die Verarbeitung an weitere Methoden delegiert. Folgende Aufgaben werden erledigt: Auf eine Nachricht `initializeProfile` wird nach

Auslesen übergebener Parameter mit der Nachricht `profileInitialized` geantwortet, welche die Profil-spezifischen Parameterfestlegungen beinhaltet. Die Anweisung `startTransmission` bewirkt das Erzeugen eines entsprechenden Sender-Moduls, welches verbunden und initialisiert wird. Weitere Steuerungsnachrichten wie `stopTransmission`, `transmissionStopped`, `transmissionFinished` werden von Sender-Modul zu RTP-Modul oder in die andere Richtung weitergeleitet. Die Nachrichten `transmissionFinished` und `transmissionStopped` weisen das Profil-Modul außerdem an, das Sender-Modul abzubauen. Ausgehende RTP-Pakete werden vom Sender-Modul an das RTP-Modul weitergeleitet. Bei ankommenden RTP-Paketen wird zunächst geprüft, ob für diesen Sender bereits ein Empfänger-Modul vorhanden ist. Falls nicht, wird eines erzeugt und mit dem Profil-Modul verbunden. Das RTP-Paket wird dann an das entsprechende Empfänger-Modul weitergeleitet.

Von `RTPProfile` abgeleitete Klassen, die ein konkretes Profil implementieren, müssen die Methode `initialize()` überschreiben. In dieser werden Profil-spezifische Parameter festgelegt.

4.1.5 Die Klasse RTPAVProfile

Die Klasse `RTPAVProfile` ist von `RTPProfile` abgeleitet und stellt das Profil-Modul für das Audio/Video-Profil mit minimaler Steuerung dar. Von seiner Oberklasse wird nur die Methode `initialize()` überschrieben, in der einige Parameterwerte gesetzt werden. Entsprechend dem Profil finden keine Veränderungen an ein- oder ausgehenden RTP-Paketen statt.

4.1.6 Die Klasse RTPPayloadSender

Die Klasse `RTPPayloadSender` beinhaltet Methoden und Variablen, wie sie für alle Sender- Module im Rahmen dieser RTP-Simulation benötigt werden. Die Methode `initialize()` setzt dabei alle Variablen auf sinnvolle Startwerte, insbesondere werden initiale Werte für Sequenznummer und Zeitstempel gewählt. Die Methoden `startTransmission()`, `stopTransmission()` und `transmissionFinished()` dienen zur Steuerung der Datenübertragung bzw. Rückmeldung darüber und zur Kommunikation mit dem Profil-Modul. Eine abgeleitete Klasse muss noch eine Methode zur Datenübertragung implementieren, wobei sowohl `handleMessage()` als auch `activity()` verwendet werden können.

4.1.7 Die Klasse RTPAVProfilePayload32Sender

Diese von RTPPayloadSender abgeleitete Klasse ermöglicht es, simuliertes MPEG-1-Video mit Hilfe der oben beschriebenen Module mit RTP zu übertragen. Als Quelle dient eine mit den in [Barq00] entwickelten Programm `MpegStat` aus einem MPEG-1-Video erstellte `.gdf`-Datei. Die Datenübertragung erfolgt dabei soweit mit simulierten MPEG-Video möglich entsprechend der Vorgaben aus RFC 2250 [Grou98]. Dazu wird in das RTP-Paket ein weiteres Paket vom Typ `RTPMpegPacket` eingekapselt, welches die im RFC für MPEG-1-Video definierten Felder beinhaltet. Da die Daten jedoch nicht direkt aus der MPEG-Datei sondern aus der `.gdf`-Datei gelesen werden, können die meisten Felder nicht mit Werten belegt werden, nur der Bild-Typ wird eingetragen. Die Nutzdaten werden mit `addLength()` zu dem `RTPMpegPacket` hinzugefügt. Die eventuell notwendige Aufteilung eines Frames auf mehrere RTP-Pakete ist nicht entsprechend der Slice-Grenzen möglich, da diese Informationen nicht in der `.gdf`-Datei enthalten sind.

In der Methode `initialize()`, welche die entsprechende Methode der Oberklasse aufruft, werden zunächst für dieses Datenformat spezifische Parameter gesetzt. Die Datenübertragung erfolgt dann in der Methode `activity()`, welche zunächst die Methode `startTransmission()` aufruft, welche um die Fähigkeit erweitert wurde, den Kopf der `.gdf`-Datei zu lesen und zu verarbeiten. Der weitere Ablauf besteht nur darin, dass sich das Modul Eigennachrichten schickt, die zu dem Zeitpunkt eintreffen, an dem das nächste MPEG-Bild verschickt werden soll. Die Eigennachricht führt zum Aufruf der Methode `sendFrame()`. Diese liest zunächst eine Zeile aus der `.gdf`-Datei. In einer Schleife, in der die noch noch versandten Bytes des Bildes heruntergezählt werden, wird zunächst ein `RTPMpegPacket` erzeugt, die Kopffelder mit Werten gefüllt und die Daten mit `addLength()` dem Paket hinzugefügt, wobei die Größe das Minimum aus übrigen Bytes des Bildes und maximale Größe der Nutzdaten in einem Paket ist. Die noch übrigen Bytes des Bildes werden nun entsprechend verringert, falls das Bild vollständig gesendet ist, wird die Schleife verlassen. Daraufhin wird eine neue Eigennachricht in die Warteschlange gesetzt und die Methode mit Rückgabewert `true` verlassen. Der Rückgabewert `false` bedeutet, dass das Ende der `.gdf`-Datei erreicht ist und `activity()` nun die Methode `transmissionFinished()` aufrufen kann.

4.1.8 Die Klasse RTPPayloadReceiver

Diese Klasse bildet das Gegenstück zur Klasse RTPPayloadSender. Mit von ihr abgeleiteten Klassen werden RTP-Pakete empfangen und verarbeitet. Da im Rahmen dieser nicht in Echtzeit ablaufenden Simulation eine direkte Ausgabe der Daten nicht sinnvoll ist, werden die Daten und Informationen über ihren Erhalt normalerweise in eine Datei geschrieben. Der Parameter legt den Dateinamen fest, wobei dieser sowohl durch das Profil-Modul, als auch durch den Benutzer bei der laufenden Simulation angegeben werden kann.

4.1.9 Die Klasse RTPAVProfilePayload32Receiver

Diese von `RTPPayloadReceiver` abgeleitete Klasse bildet das Gegenstück zur Klasse `RTPAVProfilePayload32Sender` und dient dem Empfangen und Verarbeiten der RTP-Pakete und der darin enthalten MPEG-Daten. Die Funktionalität ist dabei auf drei Methoden aufgeteilt: `initalize()` zum Auslesen von OMNeT-Parametern und zum Öffnen der Ausgabedatei, `activity()` zum Empfangen und Verarbeiten der RTP-Pakete und Schreiben in die Ausgabedatei und `finalize()` zum Schließen der Ausgabedatei. In der Reihenfolge vertauschte Pakete werden in einem gewissen Maß wieder umsortiert, genau dann wenn das MPEG-Bild, zu dem dieses Paket gehört, noch nicht abgeschlossen ist. Dabei wird ein Bild als abgeschlossen angesehen, wenn das letzte, mit einer Markierung versehene Paket dieses Bildes oder ein Paket eines folgenden Bildes empfangen wurde. Die Daten werden dabei in dem Format der in [Barq00] definierten `.sim`-Datei geschrieben, so dass sie mit dem veränderten MPEG-Player abgespielt werden können.

4.2 Klassen für Nachrichten

Sämtliche Kommunikation zwischen Modulen in diesem RTP-Modell erfolgt durch spezielle Nachrichtenklassen, welche direkt oder indirekt von `cPacket` abgeleitet sind. Dieses *message subclassing* ermöglicht im Gegensatz zu der bei OMNeT++ oftmals verwendeten Methode, eine Nachricht vom Typ `cMessage` zu verwenden und die eigenen Daten in Form von Parametern hinzuzufügen, einen schnelleren Simulationsablauf, Typsicherheit und ein leichteres Nachvollziehen der Abläufe innerhalb des Modells.

4.2.1 Die Klasse RTPInterfacePacket

Mit Nachrichten dieser Klasse kommunizieren Anwendungsmodul und RTP-Modul miteinander. Sie ermöglicht es der Anwendung, das RTP-Modul zu steuern. Dies beinhaltet den Beitritt zur Sitzung (`enterSession()`), Beginn einer Datenübertragung (`startTransmission()`), Beenden der Datenübertragung (`stopTransmission()`) und Verlassen der Sitzung (`leaveSession()`). Das RTP-Modul kann die Anwendung dann über die erfolgreiche Ausführung informieren (`sessionEntered()`, `transmissionFinished()`, `transmissionStopped()`, `sessionLeft()`). Zunächst muss eine Instanz eines `RTPInterfacePacket` erzeugt werden. Dann kann die gewünschte Methode aufgerufen werden, mit der die notwendigen Parameter übergeben werden. Danach wird die Nachricht versendet.

4.2.2 Die Klasse RTPInnerPacket

Die Klasse `RTPInnerPacket` dient zur Kommunikation zwischen Modulen innerhalb des Moduls `RTPLayer`. Die einzelnen Aufgaben sind bereits bei den Klassen `RTPEndsystemModule`, `RTCPEndsystemModule` und `RTPProfile` beschrieben, die Verwendung erfolgt wie bei Nachrichten der Klasse `RTPInterfacePacket`. Für die Übertragung von RTP-Paketen innerhalb des Moduls `RTPLayer` werden diese in ein `RTPInnerPacket` gekapselt.

4.2.3 Die Klasse RTPPacket

Die Klasse `RTPPacket` stellt ein RTP-Paket dar. Sie besitzt die Felder des RTP-Paketkopfs als Variablen und Methoden, um Werte zu schreiben und zu lesen. Die eigentlichen Daten können entweder als Pseudodaten, bei denen es nur auf die Paketgröße ankommt, mit der Methode `addLength()` simuliert werden, oder durch Einkapseln eines weiteren Paketes in das RTP-Paket übertragen werden.

4.2.4 Die Klasse RTCPPacket

Um die verschiedenen Typen von RTCP-Paketen darzustellen, wird eine kleine Klassenhierarchie verwendet, bei der `RTCPPacket` die Oberklasse bildet.

Die Klasse stellt die in jedem RTCP-Paket vorhandenen Kopffelder in Form von Variablen sowie zugehörige Methoden zum Lesen und Schreiben von diesen zur Verfügung.

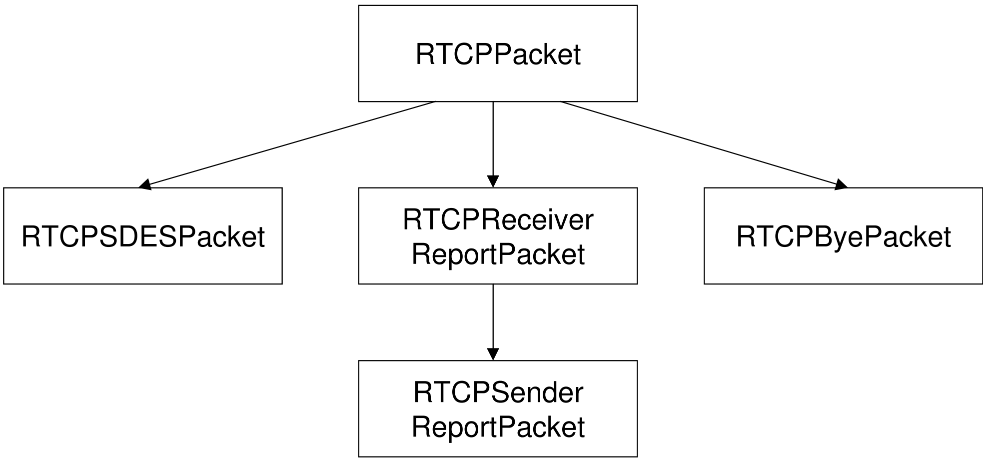

Abbildung 4.1: Die Klassenstruktur für RTCP-Pakete.

4.2.5 Die Klasse RTCPReceiverReportPacket

Die Klasse `RTCPReceiverReportPacket` ist direkt von `RTCPPacket` abgeleitet und dient dazu, Reception Reports zu übertragen. Zusätzlich zu den in der Oberklasse definierten Variablen besitzt sie eine Variable für den SSRC und ein `cArray`, in dem die `ReceptionReport`-Objekte gespeichert werden. Des Weiteren bietet sie Methoden, um auf diese Variablen zugreifen zu können, im Falle des SSRC Lese- und Schreibmethode, für das `cArray` eine Methode, um einen `ReceiverReport` hinzuzufügen und eine Methode, um eine Kopie des gesamten `cArray` zu erhalten.

4.2.6 Die Klasse RTCPSenderReportPacket

Diese Klasse ist eine Unterklasse von `RTCPReceiverReportPacket` und bietet somit die gleiche Funktionalität wie diese in Bezug auf die enthaltenen `ReceiverReport`-Objektes. Zusätzlich enthält sie aber noch ein `SenderReport`-Objekt sowie Methoden zum Lesen und Schreiben von diesem.

4.2.7 Die Klasse RTCPSDESPacket

Mit Nachrichten dieser Klassen können textuelle Informationen über Teilnehmer übertragen werden. Informationen über einen Teilnehmer werden in einem `SDESChunk` gespeichert, das den SSRC des beschriebenen Teilnehmers und mehrere `SDESItem`-Objekte enthalten kann. Ein `SDESItem` enthält die Zeichenkette, ihre Länge und ein Typfeld, das festlegt, was die Zeichenkette beschreibt. Im Rahmen der Simulation wird allerdings nur der zwingend notwendige CNAME unterstützt. Die Klasse bietet Methoden, um eine Kopie des `cArray`, in dem die `SDESChunk`-Objekte gespeichert werden, zu erhalten und um einen neuen `SDESChunk` hinzuzufügen.

4.2.8 Die Klasse RTCPByePacket

Mit einem `RTCPByePacket` verabschiedet sich ein Teilnehmer von einer RTP-Sitzung. Diese Implementierung bietet keine Möglichkeit, einen Grund für das Verlassen zu übertragen. Es kann mit einem `RTCPByePacket` nur das Verlassen eines einzigen Teilnehmers angezeigt werden. Zusätzlich zu seiner Oberklasse `RTCPPacket` beinhaltet diese Klasse nur eine Variable für den SSRC des verlassenden Teilnehmers und entsprechende Schreib- und Lesemethoden dafür.

4.2.9 Die Klasse RTCPCompoundPacket

Entsprechend der Spezifikation werden RTCP-Pakete nie allein, sondern immer in einem zusammen gesetztem Paket übertragen. Diese Container-Funktion übernimmt die Klasse `RTCPCompoundPacket`, die direkt von `cPacket` abgeleitet ist. Sie beinhaltet ein `cArray`, in dem die einzelnen RTCP-Pakete gespeichert werden und bietet Methoden, um einzelne Pakete hinzuzufügen und eine Kopie des gesamten `cArray` zu erhalten.

4.2.10 Die Klasse RTPMpegPacket

Diese Klasse dient dazu, simuliertes MPEG-Video nach RFC 2250 [Grou98] mit RTP zu übertragen. Die Klasse enthält dazu die entsprechenden Felder, sowie Methoden, um auf diese zuzugreifen. Die Länge der eigentlichen MPEG-Daten wird mit `addLength()` hinzugefügt und das `RTPMpegPacket` dann in ein `RTPPacket` eingekapselt.

4.3 Hilfsklassen

Neben den Modul- und Nachrichtenklassen gibt es noch weitere Klassen zur strukturierten Speicherung von Daten. Diese werden in bestimmten RTCP-Pakettypen sowie innerhalb des RTCP-Moduls benötigt.

4.3.1 Die Klasse ReceptionReport

Ein `ReceptionReport` enthält Informationen über den Empfang von Daten, wie er in `RTCPReceiverReportPacket` und `RTCPSenderReportPacket` verwendet wird. Sein Aufbau entspricht der Struktur *Reception Report* aus dem RFC [Grou96a]. Die Klasse besitzt dazu entsprechende Methoden, um auf die gespeicherten Daten zugreifen zu können.

4.3.2 Die Klasse SenderReport

Die Klasse `SenderReport` entspricht der im RFC [Grou96a] eingeführten Struktur *Sender Report* und dient zur Speicherung von Informationen über den Versand von Daten. Ein `RTCPSenderReportPacket` enthält immer einen `SenderReport`.

4.3.3 Die Klasse SDESChunk

Die Klasse `SDESChunk` dient zur Zusammenfassung mehrerer `SDESItem`-Objekte für einen Teilnehmer. Neben einem `cArray` zur Speicherung der `SDESItems` enthält die Klasse auch noch eine Variable für den SSRC des Teilnehmers, der mit den `SDESItems` beschrieben wird. Dazu kommen noch Methoden zum Zugriff auf diese Variablen. Verwendung findet die Klasse in `RTCPSDESPacket` und in `RTPParticipantInfo`.

4.3.4 Die Klasse SDESItem

Textuelle Informationen über einen Teilnehmer werden in einer bestimmten Struktur gespeichert, bestehend aus der Zeichenkette selbst, ihrer Länge und einem Code der angibt, was die Zeichenkette angibt. Diese Klasse besitzt die entsprechenden Variablen und Methoden zum Zugriff darauf.

4.3.5 Die Klasse RTPParticipantInfo

Entsprechend des RFC 1889 [Grou96a] speichert ein Endsystem Informationen über alle bekannten Teilnehmer an der RTP-Sitzung in einer nach dem SSRC sortierten Liste. Im Rahmen dieser Simulation erfolgt die Speicherung dieser Daten durch die Klasse `RTCPEndsystemModule` in einem `cArray`. Die einzelnen Einträge darin sind Instanzen von Unterklassen von `RTPParticipantInfo`.

Viele der Methoden dieser Klasse erhalten erst in einer der Unterklassen `RTPSenderInfo` oder `RTPReceiverInfo` ihre volle Funktionalität. Die Klasse `RTPParticipantInfo` implementiert in ihren Methoden jedoch immer die Behandlung von Zählern, welche die Aktivität von Teilnehmern beobachten. Die zeitliche Steuerung erfolgt anhand eines Aufrufs der Methode `nextInterval()` mit dem mitgeteilt wird, dass nun das nächste RTCP-Intervall begonnen hat und die Zähler erhöht werden. Mit `toBeDeleted()` kann dann abgefragt werden, ob dieser Teilnehmer aus der Liste der Teilnehmer gelöscht werden kann, entweder weil er zu lange inaktiv war, oder weil er sich verabschiedet hat. Die Klasse `RTPParticipantInfo` bietet dazu die Möglichkeit, den RTP- und den RTCP-Port, die IP-Adresse und `SDESItem`s in einem `SDESChunk` zu speichern und auf die gespeicherten Informationen zuzugreifen.

4.3.6 Die Klasse RTPSenderInfo

Diese von `RTPParticipantInfo` abgeleitete Klasse ermöglicht einem Teilnehmer an einer RTP-Sitzung Informationen über sich selbst zu speichern. Neben den von der Oberklasse geerbten Möglichkeiten für die Verwaltung von `SDESItem`s in einem `SDESChunk`, extrahiert die Klasse in der Methode `processRTPPacket()` Informationen über gesendete Daten, die dann in Form eines `SenderReport` mit der Methode `senderReport()` abgefragt werden können.

4.3.7 Die Klasse RTPReceiverInfo

Die Speicherung von von Informationen über andere Teilnehmer erfolgt mit Hilfe der Klasse `RTPReceiverInfo`, die ebenfalls von `RTPParticipantInfo` abgeleitet ist. Sie bietet zusätzliche Funktionalität in den Methoden `processRTPPacket()` und `processSenderReport()`, um darin Informationen für einen mit `receiverReport()` abzufragenden `ReceiverReport` zu erhalten.

5. Evaluation

Im Rahmen der Evaluation wird das zuvor entwickelte RTP-Modell untersucht, indem Netzwerke aufgebaut werden, in denen die Funktionen des Modells verwendet werden. Die Untersuchung beinhaltet folgende Elemente:

- Steuerung von RTP durch das Anwendungsmodul
- Beitritt und Verlassen der RTP-Sitzung
- Austausch von RTCP-Paketen
- Versand und Empfang von Nutzdaten
- Unicast- und Multicast-Übertragungen
- Mehrere gleichzeitige Sender
- Verhalten bei unzuverlässigen Netzwerken

Dabei ist ersichtlich, dass insbesondere die ersten drei Punkte natürlich in jedem Szenario behandelt werden und dementsprechend nur einmal näher beobachtet werden müssen. Außerdem können mehrere Elemente auch zusammen untersucht werden, solange keine gegenseitige Beeinflussung eintreten kann.

Folgende Möglichkeiten sind gegeben, das Verhalten der Simulation zu untersuchen: Als erstes sei die Beobachtung des Ablaufs durch den Benutzer genannt. Hier kann besonders gut der Austausch von Paketen zwischen den Teilnehmern, aber auch zwischen den einzelnen Modulen eines Teilnehmers überprüft werden. Dabei besteht außerdem die Möglichkeit, neu erzeugte Objekte durch Inspektion der Module zu betrachten. Im Besonderen seien hier die dynamisch erzeugten Profil-Module und die Nutzdaten-Module erwähnt. Außerdem lässt sich anhand der in den RTCP-Modulen erzeugten `ReceiverInfo`-Objekten beobachten, wieviele andere Teilnehmer der RTP-Sitzung diesem Endsystem bekannt sind. Um das Verhalten von RTCP beobachten zu können, schreibt das RTCP-Modul jedesmal, wenn ein zusammen gesetztes RTCP-Paket versendet wird und das RTCP-Intervall neu berechnet wird, einen `OutputVector` in eine Datei. Der Wert ist die berechnete theoretische Länge des RTCP-Intervall, vor einer Normierung und Multiplikation mit einer Zufallszahl. Er gibt einen Aufschluss darüber, wieviele Teilnehmer dem Endsystem bekannt sind und welche durchschnittliche Größe eines zusammen gesetzten RTCP-Paketes berechnet wurde. Nachdem die Simulation beendet ist, lassen sich damit die Zeitpunkte, an denen RTCP-Pakete versendet wurden, rekonstruieren und man kann erkennen, ob RTCP für die entsprechende Teilnehmerzahl richtig skaliert. Die Qualität der Datenübertragung mit RTP lässt sich anhand der durch die Empfänger-Module erzeugten Dateien untersuchen. Die im Rahmen dieser Arbeit entwickelten Module zum Empfang von simulierten MPEG-1-Video schreiben eine `.sim`-Datei, deren Format in [Barq00] beschrieben ist. Die Datei enthält für jedes MPEG-Bild die Menge der erhaltenen Daten und den Zeitpunkt, wann das letzte akzeptierte Paket für dieses Bild erhalten wurde. Mit Hilfe des in der erwähnten Arbeit modifizierten MPEG-Players lassen sich die Auswirkungen der durch die Übertragung möglicherweise entstandenen zeitlichen Schwankungen darstellen. Die Größe der erhaltenen Bilder ermöglicht durch einen Vergleich mit der ursprünglichen `.sim`-Datei eine Abschätzung, welcher Anteil der Nutzdaten bei der Übertragung verloren gegangen ist, sei es durch Paketverluste oder wegen zu spät erhaltener und deshalb vom Empfänger verworfener Pakete.

5.1 Das Multicast-Netzwerk 1

Das Multicast-Netzwerk 1 besteht aus vier Rechnern, die über einen Router sternförmig miteinander verbunden sind. Es findet nur Multicast-Kommunikation zwischen den Rechnern statt, wobei statische Routen verwendet werden. Als Besonderheit in diesem Netzwerk finden mehrere Datenübertragungen innerhalb einer Sitzung gleichzeitig statt, jeder Teilnehmer ist ein Sender.

Als erstes sei das Augenmerk auf die Vorgänge für den Beitritt zur Sitzung gerichtet: Die Nachricht von der Anwendung an RTP, das Erzeugen und Öffnen der Sockets durch das RTP-Modul, die Nachricht vom RTP- an das RTCP-Modul und das Erzeugen und Öffnen der Sockets durch das RTCP-Modul. Wie im Standard beschrieben, schätzt jeder Teilnehmer auf Grundlage der verfügbaren Bandbreite und der verwendeten Protokolle die Länge des RTCP-Intervalls im Voraus ab, halbiert diesen Wert und multipliziert ihn mit einer Zufallszahl zwischen 0,5 und 1,5. Der berechnete Wert ist dann die Zeit, die das RTCP-Modul wartet, bevor es selbst aktiv wird und das erste RTCP-Paket ans Netzwerk, sowie eine Rückmeldung an das RTP-Modul über den erfolgreichen Beitritt zur Sitzung sendet. Dieses sendet eine entsprechende Rückmeldung an die Anwendung. Eine Inspektion des RTP-Moduls fördert das neu erzeugte Profil-Modul zutage, das RTCP-Modul hat ein `RTPSenderInfo`-Objekt erzeugt, in dem der SSRC dieses Teilnehmers eingetragen ist. Bei der Datenübertragung sind folgende Dinge zu erwähnen: Jedes Profil-Modul erzeugt für jede Datenübertragung ein Empfänger-Modul, und da jeder Teilnehmer hier ein Sender ist auch ein Sender-Modul. Die erzeugten Module kann man durch Inspektion der Profil-Module als dessen Submodule anzeigen lassen. Als zweites sei hier das Senden der RTP-Pakete zu erwähnen: Dies erfolgt mit derselben Rate wie die Framerate des MPEG-Videos, wobei bei zu großen Bildern mehrere RTP-Pakete direkt hintereinander verschickt werden. Im Rahmen dieses Netzwerkes soll die Qualität der Nutzdaten-Übertragung nicht weiter untersucht werden, da insbesondere durch die gleichzeitigen Datenübertragungen die Übersicht vorloren ginge. RTCP ist bei diesem Netzwerk in erster Linie

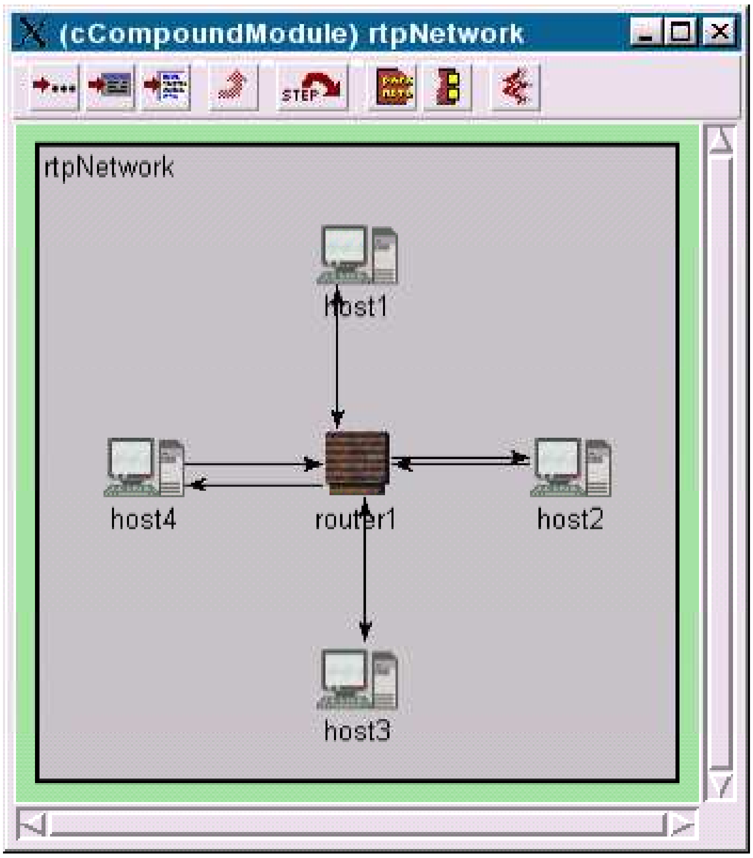

Abbildung 5.1: Das Multicast-Netzwerk 1.

im Bezug auf die Receiver Report-Pakete interessant, da in diesem mehrere Reception Reports enthalten sind, wie es sich bei Inspektion eines Paketes beobachten lässt.

5.2 Das Multicast-Netzwerk 2

Das Multicast-Netzwerk 2 dient zur Untersuchung des Verhaltens von RTCP bei vergleichsweise vielen Teilnehmern an einer Sitzung. Grundsätzlich besteht das Netzwerk aus einem zentralen Router, an den sternförmig weitere Router angeschlossen sind, die die Endsysteme anbinden. Die Anzahl sekundärer Router und die Anzahl Endsysteme pro sekundärem Router lässt sich dadurch variieren, dass in einem Skript, welches die Netzwerk-Definitions- Datei und die Routing-Tabellen erzeugt, die entsprechenden Werte eingetragen werden. Alle folgenden Angaben beziehen sich auf ein Netzwerk mit vier sekundären Routern und vier Hosts jeweils daran, also insgesamt 16 Teilnehmern an der RTP-Sitzung. Dabei erfolgt keine Datenübertragung mit RTP, es wird nur RTCP beachtet.

Die Teilnehmer treten direkt zu Beginn der Simulation der Sitzung bei, allerdings warten die einzelnen Teilnehmer zunächst einen zufällig bestimmten Zeitraum ab, in dem sie Pakete anderer Systeme empfangen, um die Wahrscheinlichkeit eines SSRC-Konflikts zu verringern. Bedingt durch diesen Wartezeitraum, werden die einzelnen Teilnehmer erst nacheinander aktiv.

Jeder Messpunkt der Kurve stellt das berechnete RTCP-Intervall zu dem Zeitpunkt dar, an dem ein Teilnehmer ein RTCP-Paket sendet. Der erste Teilnehmer, der ein

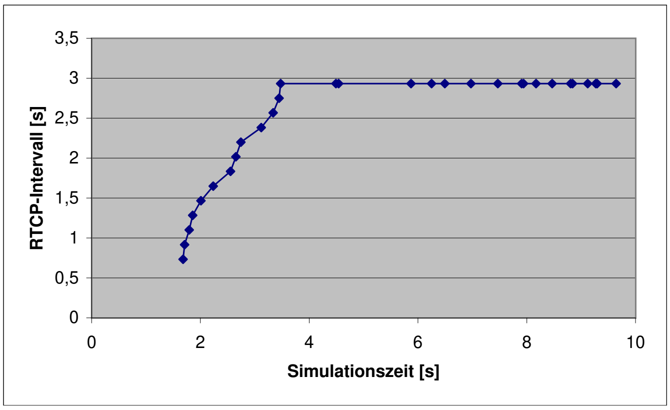

Abbildung 5.2: Die berechneten RTCP-Intervalle.

RTCP-Paket sendet, weiß zu diesem Zeitpunkt noch nichts über andere Teilnehmer außer sich selbst und kann deshalb annehmen, die gesamte RTCP-Bandbreite für sich zu beanspruchen, was sich an dem kleinen RTCP-Intervall zeigt. Dem nächsten Teilnehmer, der das RTCP-Intervall berechnet, ist der erste Teilnehmer durch dessen RTCP-Paket bereits bekannt. Er geht also von zwei Teilnehmern aus, es ist für ihn weniger Bandbreite verfügbar, entsprechend ist das Intervall größer. Dem darauf folgenden sind dann drei Teilnehmer bekannt, entsprechend ist das Intervall noch größer. Dieser Vorgang wiederholt sich solange bis sich alle Teilnehmer bekannt sind, das RTCP-Intervall ist ab diesem Zeitpunkt jedesmal gleich.

5.3 Das Unicast-Netzwerk

Das Unicast-Netzwerk besteht aus zwei Endsystemen, die direkt miteinander verbunden sind. Neben dem Aufzeigen der Eignung des Modells für Unicast-Übertragungen kann man hier auch sehr gut das rhythmische Senden der RTP-Pakete erkennen: Es werden alle RTP-Pakete eines MPEG-Frames direkt nacheinander verschickt, danach folgt eine Pause, die genau dem zeitlichen Abstand zweier Frames entspricht.

Als Besonderheit dieses Netzwerks wird hier ein unzuverlässiger Übertragungsweg verwendet: Die Verbindung zwischen den Rechnern hat eine Fehlerrate 0.00001. Aufgrund verwendeter Prüfsummen-Mechanismen unterer Schichten werden fehlerhafte Pakete erkannt und verworfen, die darin enthaltenen RTP-Pakete gehen verloren. Dadurch sind einzelnen Frames entweder unvollständig oder nicht vorhanden, was man durch den Vergleich zwischen `.gdf`-Datei und `.sim`-Datei erkennt.

6. Zusammenfassung und Ausblick

Im Rahmen dieser Studienarbeit wurde ein Modell des Real-time Transport Protokolls (RTP) in der Simulationsumgebung OMNeT++ entworfen und implementiert. Ergänzend dazu wurden Module zur Übertragung von simuliertem MPEG-1-Video entwickelt. Das Modell ist bereits auf Erweiterbarkeit und Wiederverwendbarkeit ausgelegt, was sich einerseits durch die Struktur zur Einbindung von Modulen für weitere Profile und Nutzdaten-Typen, andererseits durch eine flexible Schnittstelle zwischen Anwendung und RTP zeigt.

Als eine auf diesem Modell aufsetzende Arbeit wird bereits an der TU Braunschweig ein Modell des *Real-time Streaming Protokolls* (*RTSP*) entwickelt. Ebenso sind Arbeiten zur Weiterentwicklung des Modells durch Module zur Übertragung von echtem MPEG-1- und MPEG-2-Video in Planung. Am Institut für Nachrichtentechnik an der Universität Karlsruhe soll das Modell im Rahmen einer Simulation eines Videoservers verwendet werden, um die Tauglichkeit von RTP für diesen Anwendungsfall zu testen.

Eine weitere Erweiterungsmöglichkeit für dieses Modell besteht in der Unterstützung von RTP-Zwischensystemen. Des Weiteren ergeben sich auch interessante Fragestellungen, wenn man nicht nur höhere Schichten und Anwendungen betrachtet, sondern auch die zugrunde liegenden Netzwerk-Strukturen variiert. So ließe sich mit Hilfe des Modells die Qualität von Video-Übertragungen in mobilen Ad-hoc-Netzwerken untersuchen.

Unabhängig von den konkreten Szenarien, in denen dieses Modell weiterverwendet werden wird, lässt sich absehen, dass die Übertragung von Echtzeitdaten weiter an Bedeutung zunehmen wird und dementsprechend RTP noch häufig unter OMNeT++ ablaufen wird.

Literatur

[Barq00] G. N. Barquero. Simulation von MPEG-Datenströmen: Parameterextraktion und Visualisierung. Studienarbeit am Institut für Nachrichtentechnik, Universität Karlsruhe, Mai 2000.

[Grou96a] Audio-Video Transport Working Group (Hrsg.). RTP: A Transport Protocol for Real-Time Applications. Request For Comments, The Internet Engineering Task Force, 1996.

[Grou96b] Audio-Video Transport Working Group (Hrsg.). RTP Profile for Audio and Video Conferences with Minimal Control. Request For Comments, The Internet Engineering Task Force, 1996.

[Grou98] Network Working Group (Hrsg.). RTP Payload Format for MPEG1/MPEG2 Video. Request For Comments, The Internet Engineering Task Force, 1998.

[Rebe00] J. Reber. Entwurf und Implementierung einer Simulation des Internet Protokolls und von Dienstgütemechanismen unter OMNeT++. Diplomarbeit am Institut für Telematik, Universität Karlsruhe, 2000.

[Stei00] R. Steinmetz. *Multimedia-Technologie : Grundlagen, Komponenten und Systeme*. Springer. 2000.

[Varg00] A. Varga. *OMNeT++ – Discrete Event Simulation System, User Manual*, 2000.

[Zhu02] M. Zhu. Conception, implementation and evaluation of a socket interface layer under OMNeT++. Diplomarbeit am Institut für Nachrichtentechnik, Universität Karlsruhe, 2002.