Diplomarbeit

Entwurf einer Videokonferenz-Umgebung für das Projekt DROPS

Lukas Grützmacher
<lukas.gruetzmacher@iname.com>
Technische Universität Dresden
Professur für Betriebssysteme

31. August 2000

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Seit einigen Jahren wird an der Professur Betriebssysteme der TU Dresden an einem Echtzeitsystem geforscht: DROPS — *Dresden Realtime Operating System*. Die Grundlage bildet ein echtzeitfähiger Mikrokern: L4 bzw. die L4-kompatible Eigenentwicklung FIASCO.

Ein wesentliches Entwurfsziel von DROPS ist das umfassende Ressourcenmanagement. Dabei wird nicht nur die Nutzung der CPU überwacht, sondern auch die aller anderen Rechnerkomponenten: Arbeitsspeicher, Festplatten, Netzwerk etc. Die verschiedenen Anwendungen, die auf einem System laufen, können untereinander kommunizieren, um sich gegenseitig über kritische Lastsituationen zu informieren und entsprechend zu reagieren.

Eine Grundkomponente von DROPS ist das *DROPS Streaming Interface* (DSI). Es ist für die Übertragung von Datenströmen zwischen allen Anwendungen des Systems zuständig.

In Zusammenhang mit einem Sonderforschungsbereich der TU ist unter anderem die Entwicklung einer Videokonferenz geplant. Ein solches Konferenzsystem muß in der Lage sein, auf die Quantität aller beteiligten Ressourcen Rücksicht zu nehmen und dementsprechend die Qualität der zu übertragenden Daten anzupassen. DROPS erfüllt aufgrund seiner genannten Ziele in idealer Weise die Voraussetzungen, um auf ihm eine Videokonferenz zu betreiben.

In der vorliegenden Arbeit wird untersucht, wie ein Videokonferenzsystem nach dem Standard H.323 auf DROPS realisiert werden kann. Dabei wird besonders der Einsatz des DSI beschrieben.

Nach einer Einführung in den Standard und das zugrundeliegende System (Kapitel 2) wird ein Entwurf präsentiert (Kapitel 3). Ein Teil der Konferenz-Umgebung wurde implementiert (Kapitel 4) und anhand von Messungen bewertet (Kapitel 5). Die Arbeit schließt mit einem Ausblick auf das weitere Vorgehen (Kapitel 6).

Schreibweisen

In dieser Arbeit werde ich folgende Schreibweisen verwenden: **Fett** gedruckte Wörter sind Hervorhebungen allgemeiner Art insbesondere für wichtige Schlagwörter. *Kursiv* werden Worte geschrieben, die etablierte Fachbegriffe sind und deren Sinn entstellt würde oder zu umständlich ausgedrückt wären, wenn ich sie ins Deutsche übersetzt hätte. Letztlich symbolisiert die `Schreibmaschinenschrift`, daß es sich um Quelltext oder den Namen existierender Programme handelt.

Danksagung

An dieser Stelle möchte ich meinen Dank für die Unterstützung bei der Entstehung dieser Arbeit ausdrücken. Ganz besonders danke ich Herrn Prof. Hermann Härtig und Lars Reuther, die mir halfen, das Ziel nicht aus den Augen zu verlieren. Des weiteren danke ich allen Korrekturlesern und insbesondere meiner Verlobten Cornelia Hajek für ihre Geduld.

Erklärung

Hiermit erkläre ich, daß ich diese Arbeit selbständig erstellt und keine anderen als die angegebenen Hilfsmittel verwendet habe.

2. Stand der Technik

2.1. Konferenzsystem der Professur Rechnernetze

Seit einigen Jahren ist die Professur Rechnernetze mit der Entwicklung eines Systems zur Durchführung von Videokonferenzen beschäftigt. Es wurde ein Satz von Codec-und Protokoll-Modulen entwickelt und optimiert. Die Protokolle sind eine eigene Entwicklung. Für die Zukunft ist geplant, anerkannte Standards zur Kommunikation zu nutzen, insbesondere ist dabei an den ITU[1]-Standard H.323 (siehe Abschnitt 2.2) gedacht.

Aus diesem Grund wurde mir bei der Beratung mit Mirco Benz über die Möglichkeiten der Zusammenarbeit geraten, nicht auf die Entwicklung der Professur Rechnernetze zu bauen, sondern meine Arbeit an H.323 auszurichten.

Inzwischen wurde die Entwicklung an ein privates Unternehmen verkauft, das sich jetzt mit der Weiterentwicklung beschäftigt. Da das Unternehmen sämtliche Rechte erworben hat, kann die Professur keinen Zugang zu den Quellen ihrer Entwicklung gewähren. Es wäre nur ein Zugang zu den Schnittstellen möglich. Seit dem Verkauf wurde die Forschung in der Gruppe um Prof. Schill auf die Möglichkeiten einer Implementierung von Protokollen in Hardware fokusiert.

Aus den genannten Gründen habe ich mich entschieden, von der Teilaufgabe, dem Konferenzsystem der Professur Rechnernetze besondere Beachtung zu schenken, Abstand zu nehmen und das Konzept meiner Videokonferenz an H.323 zu orientieren.

2.2. H.323

H.323 [H323] ist ein Standard der ITU, welcher die Struktur eines Videokonferenzsystems über paketbasierte Netzwerke beschreibt und dabei auf verschiedene andere Standards [H225, H245] Bezug nimmt. Man kann somit bei H.323 von einer Sammlung von Standards sprechen.

Zusammen mit anderen Dokumenten bildet H.323 eine Standard-Familie für Multimediakommunikation über verschiedene Netzwerke. So gibt es Spezifikationen für die Kommunikation über ISDN (H.320), ATM (H.321) und PSTN (H.324). Alle diese Systeme sind darauf ausgelegt, daß ihre Komponenten mit Endpunkten der anderen Konferenzsystemklassen kooperieren können.

Seit einiger Zeit läuft ein Projekt mit Namen openH323, in dem die Standards, auf die H.323 Bezug nimmt, in OpenSource implementiert werden. Die Entwickler stützen

[1] International Telecomunication Union — ein Standardisierungsgremium

sich dabei, so wie ich es für die folgende Beschreibung getan habe, auf die Vorabversion der Standards, da die offiziellen Versionen noch nicht endgültig verabschiedet bzw. nur gegen Gebühr erhältlich sind. Zum aktuellen Status der Standardisierung sei gesagt, daß das Dokument zu H.323 zur Zeit in Version 3 vorliegt. Version 4 ist in Arbeit und soll Mitte November bestätigt werden. Auch diese Version wird wahrscheinlich nicht die letzte sein.

Die Spezifikation umfaßt Beschreibungen zu folgenden Themen:

Endpunkte Eine Konferenz nach H.323 wird zwischen zwei oder mehr Endpunkten hergestellt. Zur Laufzeit der Konferenz können weitere Endpunkte in die Konferenz aufgenommen werden.

Interoperabilität Es wird beschrieben, wie Teilnehmer anderer Konferenzsysteme mit H.323 zusammengeschlossen werden können.

Verschiedenartigkeit Der Standard schreibt eine Anzahl von Fähigkeiten (Protokolle, Codecs) vor, die jede teilnehmende Komponente beherrschen muß. Über die Nutzung weiterer Fähigkeiten wird beim Verbindungsaufbau verhandelt.

Kontrolle Eine Komponente in einer Konferenz-Umgebung hat das Privileg, die Kommunikation im Netzwerk zu steuern, um die Fähigkeiten des Netzwerkes optimal zu nutzen und es nicht zu überlasten. Außerdem gibt es einen weiteren Standard (H.235) mit Bezug auf H.323, welcher Regeln zur Gewährung von Sicherheit aufstellt.

2.2.1. Gesamtkonzept

Im Sinne von H.323 ist eine Videokonferenz ein Zusammenschluß von H.323-Komponenten. Diese werden aufgeteilt in *Terminals*, *Gateways*, *Gatekeeper* und *Multipoint Control Units* (MCU). Sie tauschen über ein gemeinsames Protokoll (H.225.0 — siehe Abschnitt 2.2.2) verschiedene Arten von Datenströmen aus:

- digitalisierte Video- bzw. Audio-Signale
- Daten, wie z. B. Textdokumente oder Bilder
- Kontroll- und Steuersignale

Die Übertragung eines Konferenz-Stromes ist in der Minimalvariante zwischen 2 **Terminals** möglich. Ein Terminal ist die Anwendung, die dem Nutzer als Schnittstelle zur Videokonferenz dient. Sie sorgt für Aufnahme und Wiedergabe der Audio- und Videodaten und stellt auf Anweisung des Nutzers Verbindungen zu anderen Teilnehmern her.

Für eine Konferenz mit mehr als 2 Teilnehmern ist eine Komponente notwendig, welche die Verteilung der Ströme übernimmt: Der *Multipoint Controller* (MC). Dieser ist Bestandteil der **Multipoint Control Unit** und ist als logische Komponente oft Bestandteil eines Terminal oder eines **Gatekeepers**. Letzterer ist für die Verwaltung aller

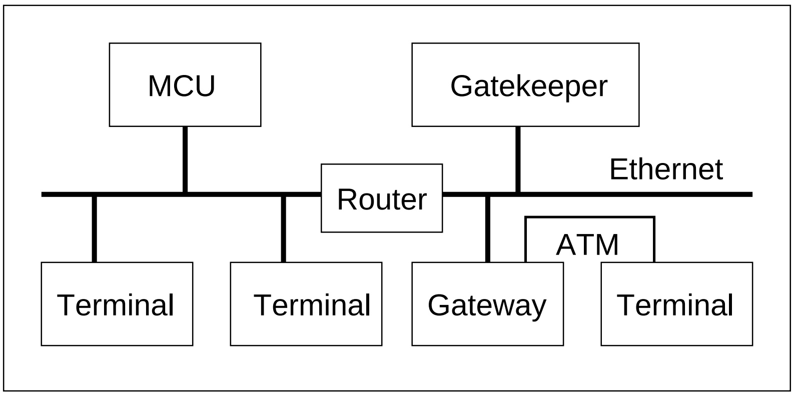

Abbildung 2.1.: Struktureller Aufbau eines möglichen H.323-Systems

Kommunikationsbeziehungen in einer Zone zuständig (siehe 2.2.4). Das Zusammenspiel der verschiedenen Endpunkte ist in Abbildung 2.1 dargestellt. Die Integration des Routers soll symbolisieren, daß eine Videokonferenz nicht auf ein Netzsegment beschränkt ist.

Der MC ist nur für die Verteilung der Medienströme von Mehrpunktkonferenzen zuständig und ist Vermittlungszentrale für deren Aufbau. Er entscheidet über die Nutzung von Uni- bzw. Multicast[2], je nach Fähigkeit des zugrundeliegenden Netzwerkes.

Dem MC können einige *Multipoint Prozessors* (MP) zur Seite gestellt werden. Ein MP ist für die Aufbereitung eines Stromes zuständig. So kann er beispielsweise alle eingehenden Audio-Ströme mischen und an die Empfänger weitersenden, die Lippensynchronität zwischen Audio- und Videostrom sicherstellen oder Mediendaten verschiedener Standards ineinander übersetzen.

Auf dedizierte *Multipoint Control Units* kann verzichtet werden, wenn jedes teilnehmende Terminal einen eigenen *Multipoint Controller* besitzt. Dann übernimmt jeder selbst die Verteilung der Ströme[3]. Dieses Vorgehen wird als *Decentralized Multipoint* bezeichnet im Gegensatz zu *Centralized Multipoint*. Auch die Mischung beider Varianten ist im Standard vorgesehen. Je nachdem, welche Ströme zentral und welche dezentral verteilt werden, spricht man von *Hybrid Multipoint — Centralized Audio* bzw. *Hybrid Multipoint — Centralized Video*.

2.2.2. H.225.0: Multiplexing und Verbindungskontrolle

H.225.0 beschreibt zum einen, wie die Datenströme für das Netzwerk paketiert und übertragen werden. Zum zweiten wird ein Protokoll definiert, das für den Verbindungsauf- und -abbau zwischen H.323-Komponenten zuständig ist. Die Protokoll-Komponente wird wiederum in zwei Teile unterteilt: Verbindungsaufbau *(Call Signalling)* und *Registration, Admission, Status*.

[2] Unicast: Adressierung eines einzelnen Rechners; Multicast: Adressierung mehrerer Rechner; Broadcast: Alle Rechner eines Segmentes sind adressiert

[3] Dazu ist es notwendig, daß das zugrundeliegende Netzwerk Multicast-Kommunikation unterstützt.

Die Signalisierung zum Verbindungsaufbau ist von der Spezifikation Q.931 abgeleitet, welche für die Signalisierung in *Integrated Service Digital Networks* (ISDN) entwickelt wurde. Sie wird entweder direkt zwischen den Endpunkten durchgeführt (es ist kein Gatekeeper vorhanden) oder durch den Gatekeeper vermittelt.

RAS — *Registration, Admission, Status* — ist der Teil des Protokolls, der die Kommunikation zwischen Endpunkten und dem Gatekeeper regelt. Ist in einer H.323-Implementierung kein Gatekeeper vorgesehen, wird dieser Teil von H.225.0 nicht gebraucht und muß folglich nicht realisiert werden[4].

H.225.0 ist darauf ausgelegt, in verschiedenen Arten von Netzwerken genutzt zu werden. Dazu zählen jedoch nur die nicht garantierenden wie z. B. Ethernet und Token Ring. Das Protokoll ist oberhalb einer Transportschicht wie TCP/UDP angesiedelt. Das bedeutet, daß H.323-Netzwerke nicht auf das lokale Netz beschränkt sind und somit beispielsweise Konferenzen über das Internet möglich sind.

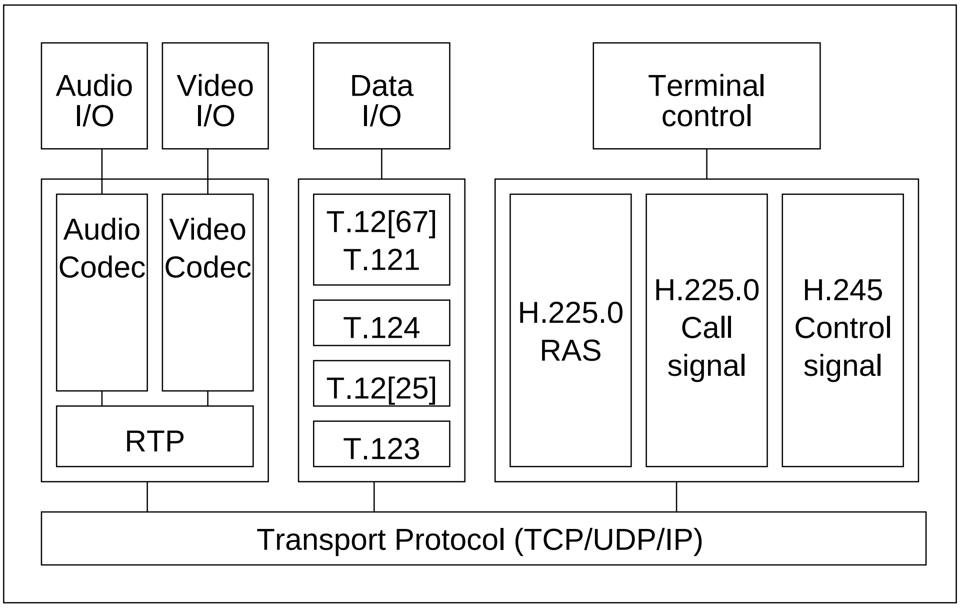

Abbildung 2.2.: H.323 Terminal Protokoll-Stack

Einige der Kontrollsignale — H.245 (siehe 2.2.3) und die Rufsignalisierung (H.225.0 *Call Control Messages*) — sowie die einzelnen Teile von Dateien[5] sind auf eine zuverlässige Übertragung angewiesen, da sie in der korrekten Reihenfolge ankommen müssen und kein Teil verloren gehen darf. Daher werden sie verbindungsorientiert übertragen.

Audio- und Videoströme sowie RAS-Signale können dagegen verbindungslos übertragen werden. Für die Medienströme nutzt H.225.0 zusätzlich das Echtzeit-Transportprotokoll RTP [IETF96]. Dabei fügt RTP Informationen über die Reihenfolge der Pakete in

[4]So geschehen in der Anfangsphase des openH323-Projektes. Mittlerweile ist ein Programm namens "opengate" im Entstehen.

[5]Textdokumente oder einfache digitale Bilder sowie Daten eines *Distributed Whiteboard* werden nach dem Standard T.120 verteilt.

den UDP-Strom ein. Dies ist wichtig, da UDP nicht dafür Sorge trägt, daß alle Pakete in der korrekten Reihenfolge beim Empfänger abgeliefert werden. RTP wird nicht auf TCP betrieben, da dieses Protokoll, zusätzlich zur Sicherstellung der Reihenfolge, verlorene Pakete neu anfordert. Dies ist bei der Übertragung von Mediendaten jedoch nicht sinnvoll. Bis das Paket auf erneute Anforderung beim Empfänger eintrifft, ist es bereits veraltet. Die Anwendung muß mit dem Verlust einzelner Daten rechnen.

Um einen Kommunikationsendpunkt eindeutig identifizieren zu können, muß dieser mit einem *Transport Layer Service Access Point* (**TSAP**) bezeichnet werden. TSAPs sind teilweise dauerhaft spezifiziert (vgl. Anhang D in [H225]) oder werden dynamisch erzeugt. Ein festgelegter TSAP ist beispielsweise der *Call Signalling Channel TSAP*. An diesen sendet ein Terminal den Aufruf zum Verbindungsaufbau (*Setup*) mit einem zweiten Terminal, wenn beide bei keinem Gatekeeper registriert sind.

Es ist nicht zwingend, den festgelegten TSAP zum Verbindungsaufbau zu nutzen. Es wird sogar empfohlen, nur dynamisch erzeugte Adressen zur Kommunikation zu nutzen, da sonst pro Netzwerkinterface nur eine Verbindung aufgebaut werden kann.

2.2.3. H.245: Protokollsyntax

H.245 beschreibt die Syntax der Nachrichten, die zwischen den Teilnehmern einer Konferenz ausgetauscht werden. Sie ist in ASN.1 definiert, einer hardwareunabhängigen Beschreibungssprache für Datenstrukturen (siehe Beispiel in Abbildung 2.3). Nach dem Bekanntmachen zweier oder mehrerer Endpunkte durch *Call Signalling* (siehe 2.2.2) wird ein entsprechender Kanal zur Verbindungskontrolle aufgebaut. So werden H.245-Nachrichten genutzt, um genaue Informationen bezüglich der zukünftigen Übertragung zwischen zwei Endpunkten[6] auszuhandeln. Dazu übermittelt jeder Endpunkt eine Liste seiner Fähigkeiten. Aus der Schnittmenge werden dann die Eigenschaften der Verbindung abgeleitet.

Ohne Gatekeeper muß der Kanal direkt zwischen den Endpunkten aufgebaut werden. Wenn ein Gatekeeper vorhanden ist und die Endpunkte bei diesem registriert sind, hat er die Wahl, den H.245-Kanal direkt zwischen den Endpunkten oder durch sich selbst zu etablieren[7]. Letzteres hat den Vorteil, daß der H.245-Kontrollkanal an einen *Multipoint Controller* weitergereicht werden kann, wenn aus der Punkt-zu-Punkt-Verbindung eine Mehrpunkt-Konferenz werden soll.

Für jeden Strom von Daten wird ein **logischer Kanal** geöffnet. Die Aushandlung, welche Nummer mit welchem Strom in Verbindung stehen soll, wird ebenfalls mittels Nachrichten aus H.245 geführt. Für den H.245-Kanal selbst ist die Kanalnummer 0 festgelegt und gilt auch ohne entsprechende Interaktion als aufgebaut.

Es gibt eine Liste weiterer Protokolle. Hier seien nur noch die genannt, die jeder H.323-Endpunkt neben den beschriebenen beherrschen muß: *Master-Slave*-Auswahl sowie Zustandsprüfung des entfernten Terminals (*Round Trip Delay*) oder einzelner logischer Kanäle (*Maintenance Loop*).

[6] H.245 Signalisierung besteht immer zwischen zwei Endpunkten, einem Endpunkt und einem MC oder einem Endpunkt und einem Gatekeeper.

[7] Sind Endpunkte bei zwei verschiedenen Gatekeepern registriert, müssen sich diese beiden über das Vorgehen einigen.

```
VideoCapability ::=CHOICE
{
  nonStandard NonStandardParameter,
  h261VideoCapability H261VideoCapability,
  h262VideoCapability H262VideoCapability,
  h263VideoCapability H263VideoCapability,
  is11172VideoCapability IS11172VideoCapability,
  ...
}

H261VideoCapability ::=SEQUENCE
{
  qcifMPI INTEGER (1..4) OPTIONAL, -- units 1/29.97 Hz
  cifMPI INTEGER (1..4) OPTIONAL, -- units 1/29.97 Hz
  temporalSpatialTradeOffCapability BOOLEAN,
  maxBitRate INTEGER (1..19200), -- units of 100 bit/s
  stillImageTransmission BOOLEAN, -- Annex D of H.261
  ...
}
```

Abbildung 2.3.: Beispiel zweier Definitionen in ASN.1

2.2.4. Gatekeeper und Quality of Service (QoS)

Der Gatekeeper ist eine H.323-Komponente, welche den Service in einer Konferenz verbessern kann. So ermöglicht er neben der zentralen Registrierung der Terminals auch *Admission Control*. Über Nachrichten aus H.225.0 können der Zugang und die Übertragungsbandbreite einzelner Komponenten ausgehandelt und so in gewissem Maße QoS-Eigenschaften realisiert werden. Mittels Austausch von Zustandsberichten während der Laufzeit einer Verbindung kann eine Neuverhandlung der Parameter ausgelöst werden. So kann der Bericht des Empfängers beispielsweise die Anzahl der verlorenen Pakete enthalten. Daraus resultierend kann entweder die reservierte Bandbreite für diese Verbindung erhöht werden, um die Verlustrate zu reduzieren, oder die Mediendaten werden in einer geringeren Qualität gesendet, was die erforderliche Bandbreite senkt.

Wenn ein Gatekeeper in einem Netzwerk vorhanden ist, müssen sich alle Komponenten (Terminals, Gateways, MCUs) bei ihm registrieren. Er übernimmt dann die Vermittlung im Verbindungsaufbau. Ohne einen Gatekeeper müssen alle Endpunkte direkt miteinander kommunizieren.

Die Reservierung von Leitungskapazität soll nach H.323 auf der Ebene der Transportschicht erfolgen. Dazu wird das *Resource Reservation Protocol* (RSVP) vorgeschlagen, welches für IP-basierte, nicht garantierende Netzwerke spezifiziert wurde. Das Management übernimmt der Gatekeeper. Das Terminal muß jedoch selbst dafür Sorge tragen, die reservierte Bandbreite nicht zu überschreiten. Dazu muß es zuerst die notwendige Bandbreite für alle seine Verbindungen berechnen. Beim Öffnen eines logischen Kanals für die

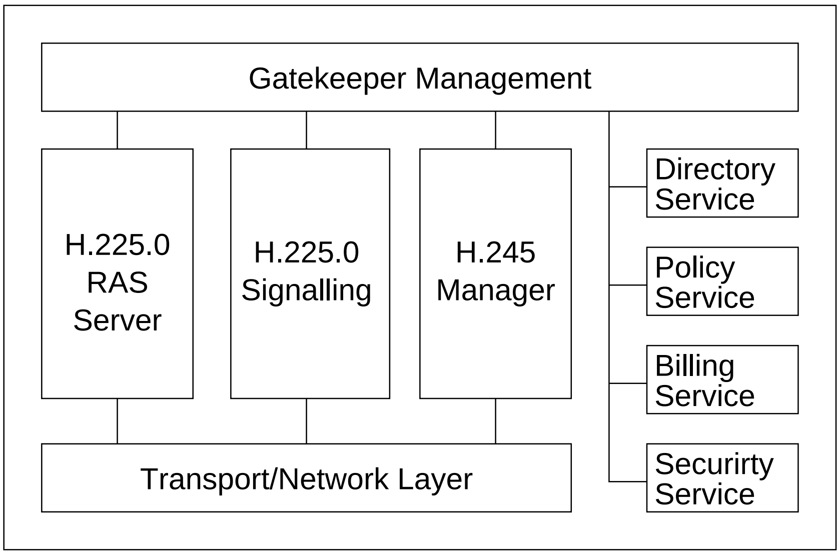

Abbildung 2.4.: Struktur eines Gatekeepers mit einigen Service-Einheiten für verschiedene Dienste

Daten werden nun die Werte für die notwendige Bandbreite mitgeliefert. Die Bestätigung über den neuen Kanal enthält auch eine Mitteilung über die reservierten Ressourcen der beteiligten Router. Beim Abbau des Kanals sollte dementsprechend auch der Hinweis zur Freigabe der Ressourcen mitgeliefert werden.

Eine optionale Fähigkeit eines Gatekeepers ist das Routing von Signalen zum Verbindungsaufbau. Dies ermöglicht es, den Überblick über alle Verbindungen zentral zu halten und so beispielsweise die Last zwischen verschiedenen Wegen zu teilen oder Buch zu führen, um den Kunden die Leistung in Rechnung stellen zu können.

Der Gatekeeper kann auch entscheiden, die Anzahl der Teilnehmer zu begrenzen oder bestimmte Endpunkte auszuschließen. So kann die Qualität der Konferenz verbessert werden.

Wie in Abschitt 2.2.2 erwähnt, erfolgt die Kommunikation zwischen Endpunkten und dem Gatekeeper mit RAS-Nachrichten. So steht beispielsweise am Anfang der Registrierungsprozedur die Suche nach einem verfügbaren Gatekeeper: *Gatekeeper Request* (GRQ). Dafür sind eine Multicast-Adresse und eine Portnummer festgelegt [H225, Anhang D]. Unterstützt das Netzwerk kein Multicast, so wird die Anfrage als Broadcast gesendet.

Nun kann sich der Endpunkt mit seinen Adressen und Alias-Namen registrieren. Der Gatekeeper antwortet darauf mit einer Bestätigung, wenn er diesen Endpunkt verwalten will. Ansonsten muß er mit einer Ablehnung antworten, weil beispielsweise die Maximalzahl der verwaltbaren Endpunkte erreicht ist. Andernfalls könnte ein Endpunkt das Ausbleiben einer Antwort als Verlust einer Nachricht interpretieren und darf seine Anfrage periodisch wiederholen.

H.323 sieht vor, mehr als einen Gatekeeper in einer Verwaltungszone zu installieren.

Dadurch kann die Verfügbarkeit des Systems erhöht werden. Dazu schickt der erste Gatekeeper mit der Bestätigung der Registrierung eine Liste alternativer Gatekeeper, an die sich der Client wenden kann, wenn der erste nicht mehr erreichbar ist.

2.2.5. Video-Kodierung

Videodaten weisen eine hohe Redundanz auf. Es bietet sich daher an, überflüssige Teile wegzulassen. Diese Kompression kann verlustbehaftet oder -frei erfolgen. Daten, die wieder exakt hergestellt werden müssen (z. B. Textdokumente), sind verlustfrei zu komprimieren. Dagegen können beispielsweise Videodaten mit Verlust komprimiert werden, da dieser vom Menschen nicht wahrgenommen wird, wenn eine bestimmte Verfälschungsgrenze nicht überschritten wird.

Zusammen mit der Art und Weise, einzelne Bildteile als Bestandteil des Stromes zu kennzeichnen, ergeben sich verschiedene Möglichkeiten der Kodierung, die sich in unterschiedlichen Standards manifestieren: z.B. MPEG oder H.261/263.

Die Video-Codecs der H-Serie aus der Standard-Reihe der ITU sind besonders auf die Erfordernisse einer Videokonferenz ausgerichtet. Vor allem H.263 ist auf die Übertragung mittels schmalbandiger Medien optimiert.

H.261

H.261 ist ein Standard, der beschreibt, wie Video-Daten — optimiert für Videokonferenzen — über Netzwerke mit Bandbreiten von 40 kBit bis 2 MBit übertragen werden können. Ausgangspunkt ist hierbei, daß die verbreiteten Fernsehnormen PAL/SECAM und NTSC mit 525 bzw. 625 Zeilen pro Bild, also einer unterschiedlichen Auflösung, arbeiten. Daher hat man sich geeinigt, die Eingangsdaten für den Kodierer im *Common Intermediate Format* (CIF) zu erwarten. Dieses schreibt eine Auflösung von 352 Punkten in 288 Zeilen vor. Ein zweites, abgeleitetes Format ist *quarter-CIF* (qCIF) mit horizontal und vertikal halber Auflösung. Der Standard schreibt vor, daß jede Implementierung eines H.261-Codec qCIF verarbeiten können muß, die Unterstützung von CIF ist optional.

Die Kodierung erfolgt hierarchisch in den Ebenen **Bild**, **Blockgruppe** (GOB), **Makroblock** (MB) und **Block** (siehe Abbildung 2.5). Dazu wird ein CIF-Bild in 12, ein qCIF-Bild in 3 Blockgruppen zerlegt. Eine Gruppe — nun immer 176x48 Pixel groß — wird wiederum in 33 Makroblöcke zerlegt.

Ein Makroblock enthält die Informationen von 16x16 Pixeln und wird unterteilt in 4 Blöcke zu 8x8 Pixeln mit Helligkeitsinformationen und 2 darüberliegenden Blöcken mit Farbinformationen, wobei die Farbinformation von 4 Pixeln des Quellbildes zu einem zusammengefaßt werden. An dieser Stelle erkennt man die Nähe zur Farbkodierung der Fernsehnormen: Es werden vorrangig Helligkeitsinformationen übertragen, die Farbinformation ist deutlich reduziert. Man spricht hier von einer 4:1:1-Kodierung.

Für jeden Block wird entschieden, ob er übertragen werden muß oder ob er sich aus anderen, bereits übermittelten, ableiten läßt. Zuletzt wird jeder Block verschiedenen

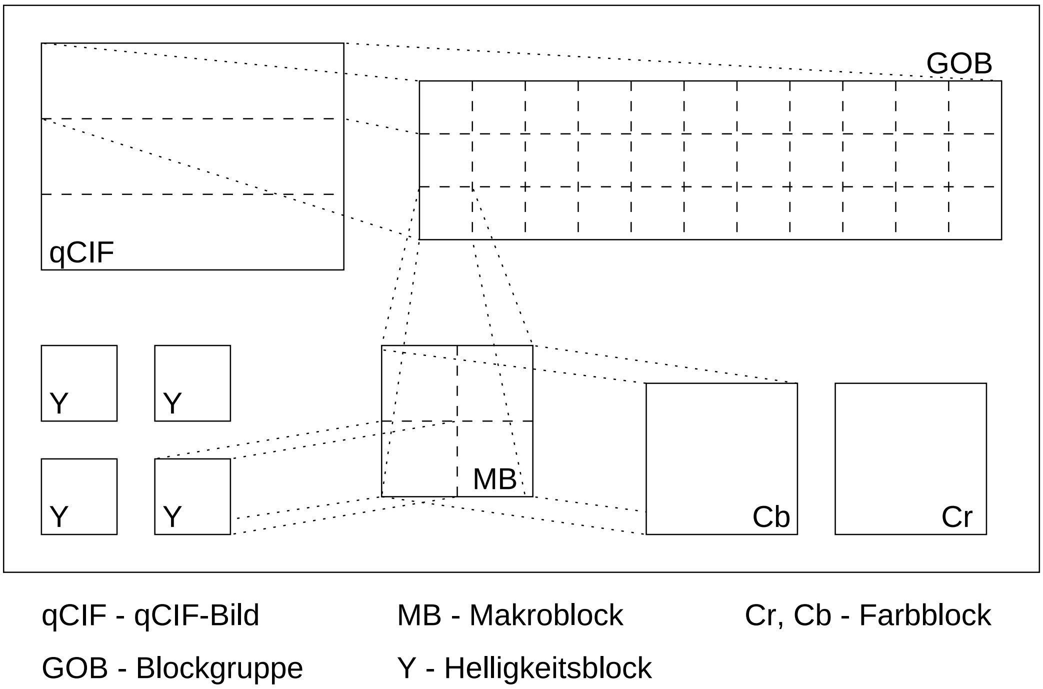

Abbildung 2.5.: Hierarchischer Aufbau eines Bildes im Format qCIF

Transformationen unterzogen (unter anderem der diskreten Cosinus-Transformation) und wird so, wie bei MPEG, verlustbehaftet komprimiert.

Zusätzlich werden Informationen zur Fehlerkorrektur nach BCH (Bose, Chaudhuri und Hocquengham) eingefügt, die vom Dekoder aber nicht genutzt werden müssen.

H.263

H.263 ist eine abwärtskompatible Weiterentwicklung von H.261, die zum einen die Qualität deutlich verbessert und zum zweiten durch eine Huffmann-Code-Tabelle eine Optimierung für schmalbandige Verbindungen (<64kBit) mit sich bringt.

Des weiteren wurden drei neue Bildformate eingeführt: sub-qCIF (128x96), 4CIF (702x576) und 16CIF (1408x1152).

2.2.6. Audio-Kodierung

In der gleichen Weise wie bei der Kodierung eines Videostromes gibt es verschiedene Verfahren, Audio-Signale auszutauschen.

G.711

Die einfachste Form der Kodierung von Audio-Signalen ist die *Pulse Code Modulation* (PCM). Dazu wird das bandbreitenbegrenzte Eingangssignal periodisch abgetastet und quantisiert. G.711 schreibt die Abtastung des Signales, welches auf 100 Hz bis 3.5 kHz begrenzt ist, mit einer Frequenz von 8 kHz und Quantisierung auf 13 Bit vor.

Jeder Einzelwert wird logarithmisch auf 8 Bit heruntergerechnet. So wird die Übertragung auf besondere Eigenheiten des menschlichen Gehörs optimiert.

G.722

G.722 verbessert G.711 vor allem dadurch, daß die Bandbreite des Eingangssignals nicht so stark begrenzt wird. Die obere Grenze liegt hier bei 7 kHz. Des weiteren wird jeder Kanal in 2 Frequenzbänder unterteilt, die unabhängig voneinander kodiert werden. Die Abtastrate beträgt 16 kHz bei einer Quantisierungsgenauigkeit von 14 Bit.

G.722 nutzt des weiteren *Adaptive Differential Puls Code Modulation* (ADPCM). Dieses Verfahren nutzt die Ähnlichkeit benachbarter Signalwerte und kann so durch Vorhersage und Differenzkodierung die Datenmenge reduzieren.

G.728

G.728 ist ein Verfahren, welches für die Übertragung von Sprache optimiert ist. Das digitalisierte Signal wird in kurze Kurvenstücke zerlegt und mit denen einer Tabelle verglichen. In den Datenstrom wird dann ein Identifikator für das Stück geschrieben, welches am besten mit der Vorlage übereinstimmt. So ist es möglich, die Datenrate auf 16 kBit/s zu reduzieren.

2.2.7. H.235: Sicherheit

Mit dem Standard H.235 wurde H.323 um Sicherheitsaspekte erweitert. Er beschreibt verschiedene Möglichkeiten, um Authentifizierung (der Kommunikationspartner ist wirklich der, für den er sich ausgibt), Integrität (Daten werden unverfälscht übertragen) und Vertraulichkeit zu unterstützen. Der Standard ist dabei unabhängig von den genutzten kryptographischen Mitteln.

Der Verbindungsaufbau kann sicherer gemacht werden, indem die IP-Erweiterung IPSEC zur Kommunikation genutzt wird. Anschließend kann bei der Aushandlung der Konferenzeigenschaften eine Authentifizierung vorgenommen werden (z. B. mittels Austausch von Zertifikaten). Des weiteren wird beschrieben, wie Datenströme (logische Kanäle) verschlüsselt werden können.

Um in einer Mehrpunktkonferenz Sicherheit zu gewährleisten, wurde festgelgt, daß der *Multipoint Controller* auch hier eine zentrale Rolle spielt. Er wird als vertrauenswürdig angenommen und handelt mit allen Teilnehmern die Bedingungen zum Verbindungsaufbau aus.

H.235 legt auch fest, wie Schlüssel bzw. Zertifikate ausgetauscht werden und wie sich einzelne Endpunkte verhalten sollen, wenn sie eine Sicherheitsverletzung erkennen.

2.3. DROPS-Komponenten

Ansatz meiner Arbeit ist die Verbindung einiger der bereits entwickelten Komponenten für das DROPS-System, welches auf den Mikrokern L4 [LIE96] aufbaut. Im folgenden Abschnitt gehe ich auf alle verwendeten L4-Komponenten ein. Ich beschreibe ihren Einsatz und Zweck und beurteile sie bezüglich der Echtzeitfähigkeit, soweit dies möglich ist.

2.3.1. meteor

`meteor` ist ein Server, der den Linux-Treiber zur Ansteuerung der Meteor-Framegrabber-Karte für das L4-System kapselt.

Dieser Server wurde von Jork Löser ohne Rücksicht auf Echtzeitfähigkeit entwickelt, um erst einmal eine Möglichkeit zu haben, eine Folge von Live-Bildern in das System einspeisen zu können.

Da die Meteor-Karten nicht mehr weiterentwickelt werden und es zudem mittlerweile leistungsfähigere Karten gibt, sollte in Zukunft ein neuer Treiber (für die Meteor- oder eine andere Karte) entwickelt werden, bei dem die Echtzeitfähigkeit berücksichtigt wird.

2.3.2. console

Torsten Paul hat in seiner Diplomarbeit [PAU98] einen Konsolen-Treiber auf L4 portiert. Dieser erlaubt es, eine Grafikkarte in seine verschiedenen Grafik-Modi zu schalten. Anschließend kann der Grafikspeicher, der in den Hauptspeicher des Rechners eingeblendet ist, direkt manipuliert werden.

Der Einsatz in einem Echzeitsystem ist einfach möglich, da nur die Initialisierung der Hardware ein nicht abschätzbarer Vorgang ist und während des Systemstarts ausgeführt wird. Durch die direkte Manipulation des Grafikspeichers ist das Verhalten zur Laufzeit sehr gut vorhersagbar.

2.3.3. ether

Dieser Server [HOH] ist ein einfacher Hardware-Treiber. Er nutzt die Funktionalität des OSKit [OS96] zur Steuerung der Netzwerkkarte. Die zusätzliche Funktionalität des Servers besteht in der Bereitstellung einer Schnittstelle für andere L4-kompatible Anwendungen.

Der Ethernet-Server ist nicht auf Echtzeitfähigkeit hin gestaltet. Da jedoch das einfache Ethernet mit seinem nicht deterministischem Zugriffsverfahren generell nur schwer echtzeitfähig zu machen ist, spielt diese Herangehensweise keine große Rolle.

2.3.4. tcpip

Während der Entstehung dieser Arbeit beendete Marco Rudolph seine Diplomarbeit [RUD00], aus der ein L4-Server hervorging, der sich den Ethernet-Server zunutze macht und auf ihm einen TCP/IP-Stack implementiert. UDP wird ebenfalls unterstützt, auch wenn der Name dieses Servers dies nicht zeigt.

Der Server bietet ein typisches Socket-Interface. Zusätzlich wurde die Festlegung von Prioritäten implementiert, was die Bevorzugung wichtiger Pakete ermöglicht. Damit können in begrenztem Maße Zusagen für die Übertragung gemacht werden.

Da am Ende meiner Planungszeit der Server nicht vollständig zur Verfügung stand, konnte ich ihn nicht in meine Implementierung nutzen.

2.3.5. ATM

In seiner Dissertation [BOR98] hat Martin Borriss einen QoS-fähigen ATM-Server entwickelt. Auf dem ATM-Treiber von Uwe Dannowski [DAN99] basierend, verwaltet diese Komponente sämtliche Datenströme zwischen zwei Rechnern. Jedem einzelnen Strom können bestimmte Bandbreiten zugesichert werden. Die Einhaltung der angeforderten Datenmengen wird überwacht und, wenn es notwendig wird, erzwungen.

2.3.6. rtfs

Das Echzeit-Dateisystem ist eine Entwicklung von Lars Reuther [REU98]. Es wurde das *Extended Filesystem 2* (ext2) von Linux um Reservierungsoptionen erweitert. Diese Ergänzung erlaubt das Reservieren von Übertragungsbandbreiten für Lesen und Schreiben. Das wurde möglich durch die Nutzung eines SCSI-Treibers, der Zusagen über die Ausführungszeit einzelner Aufträge an die Hardware machen kann.

Zur Nutzung des Dateisystems existiert neben dem Standard POSIX-Interface für die normale Nutzung — beispielsweise durch Linux — ein zusätzliches Echtzeitinterface. Aus der Erfahrung mit der Nutzung dieser Schnittstelle ist inzwischen das *DROPS Streaming Interface* (siehe Abschnitt 2.4) hervorgegangen.

Es ist mittlerweile eine vollständige Neuentwicklung in Arbeit. Dabei sind die Verwaltungsstrukturen nur noch teilweise an die von ext2 angelehnt.

2.3.7. Weitere

rmgr

Der Ressourcenmanager ist eine der zentralen Komponenten eines L4-Systems. Ab dem Systemstart verwaltet er alle wichtigen Ressourcen: Speicher, Interrupts, IO-Bereich.

buffermgr

Der Buffer-Manager ist ein einfacher Speicherverwalter, der es zwei Anwendungen ermöglicht, einen gemeinsamen Speicher zu nutzen.

names

`names` realisiert einen einfachen Namensdienst, der eine Zeichenkette auf eine Prozeßnummer abbildet.

Da diese Anfragen wärend des Systemstarts erfolgen sollten, ist die Echtzeitfähigkeit dieses Servers nicht relevant.

2.4. DSI — DROPS Streaming Interface

Innerhalb eines Medienservers[8] oder eines Videokonferenzsystems müssen Datenströme möglichst übersichtlich und effizient übertragen werden. So soll beispielsweise eine „unendliche“ Folge von Bildern, welche eine Framegrabber-Karte liefert, kodiert über ein Netzwerk übertragen werden. Dies ergibt das Zusammenspiel von 3 Komponenten mit insgesamt 2 Berührungspunkten. Die Bilder werden von dem Treiber der Framegrabber-Karte an den Kodierer geliefert, der diese bearbeitet und dann an die Netzwerkkomponente weiterreicht. Dabei ist es prinzipiell für die Bildquelle uninteressant, ob sie die Bilder an den Codec liefern oder zuvor von einem Filter bearbeiten lassen soll. Dieser Filter soll nun einfach und transparent in die Kette eingegliedert werden, die dann 4 Komponenten mit 3 Berührungspunkten enthält.

Daraus ergab sich die Entwicklung des *DROPS Streaming Interface* (DSI) [RLG00]. Dieses stellt über eine Sammlung von Schnittstellen einen Puffer-Pool bereit, in den eine Komponente ihre Daten schreiben kann und aus der sie eine zweite Komponente wieder entnehmen kann, ohne daß die Komponenten direkt miteinander kommunizieren müssen. Das übernimmt die Implementierung in DSI für sie.

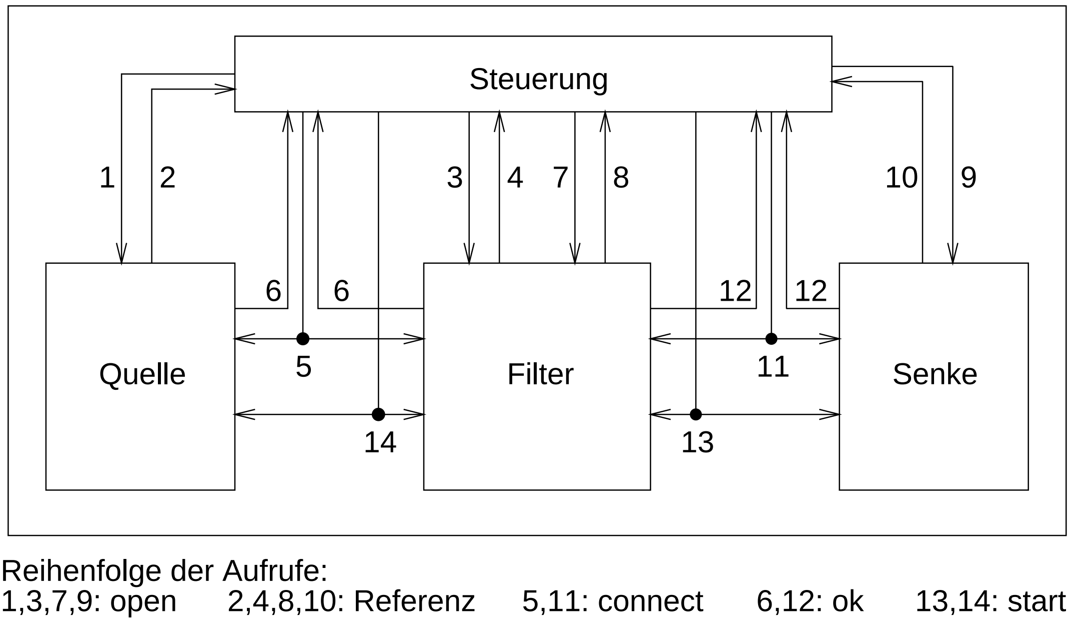

Abbildung 2.6.: 3 Komponenten und eine Steuerungsanwendung sowie deren Interaktion beim Aufbau des Stromes

[8] laufende Entwicklung an der Professur Betriebssysteme — eine Art *Video on Demand* (siehe Abschnitt 3.3)

2.4.1. Ansatz

Für die Realisierung eines DSI-basierten Systems wird die Komponenten-Kette auf einen paarweisen Zusammenschluß reduziert. Eine Anwendung, beispielsweise das Benutzerinterface, kontaktiert zwei streamingfähige Komponenten, die daraufhin eine Referenz für den Anschluß an einen eindeutigen Strom zurückliefern. Die Anwendung konfiguriert beide Seiten und verbindet die Enden, welche nach dem Startsignal selbständig die Daten austauschen. Die Anwendung ist nicht weiter beteiligt, es sei denn, sie will den Strom unterbrechen oder neu konfigurieren.

So läßt sich die Kette beliebig erweitern. Die beschriebene oder eine neue Anwendung kann weitere Komponenten-Paare zu einem Datenaustausch verbinden. Die Abbildung 2.6 zeigt das genannte Beispiel aus drei Komponenten und die Reihenfolge der einzelnen Nachrichten.

2.4.2. Implementierung

Das *DROPS Streaming Interface* wurde in Form einer Bibliothek realisiert. Sie wird Teil aller Anwendugen und Komponenten, die DSI nutzen wollen. Abbildung 2.7 zeigt die Zusammensetzung zweier Komponenten und einer Anwendung. Die beiden zusätzlichen, bisher nicht genannten Bibliotheken, welche Bestandteil der Anwendung werden müssen, sind von den einzelnen Komponenten bereitzustellen.

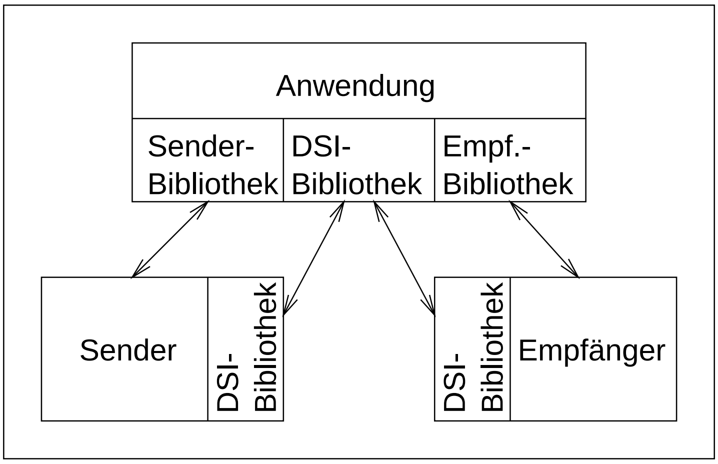

Abbildung 2.7.: Aufbau von DSI-Komponenten

Die DSI-Bibliothek enthält folgende Teile:

Bereitstellung von Strombeschreibungsstrukturen

Der Endpunkt eines Stromes, welcher von einer Komponente zur Verfügung gestellt werden soll, wird als *Socket* bezeichnet, die eindeutige Verbindung zweier Sockets als *Stream*. Für beides ist je eine Verwaltungsstruktur notwendig, um die zu übertragenden Daten richtig zuordnen zu können. Diese Strukturen werden von der DSI-Bibliothek bereitgestellt und initialisiert.

Verwaltung der Daten

Der Datenaustauch wird über einen gemeinsamen Speicherbereich abgewickelt, der beim Verbindungsaufbau in Sender und Empfänger eingeblendet wird. Der physische Speicher, der hinter diesem sogenannten *Shared Memory* steht, wird von einem beliebigen Speichermanager bereitgestellt. Diese Manager können Speicher mit unterschiedlichen Merkmalen verwalten, z. B. „DMA-fähig“ oder „einem bestimmten Cachespeicher-Bereich zugeordnet“. Der Anwender der DSI-Bibliothek kann Speicher eines von ihm favorisierten Managers anfordern.

Die Bereistellung der Daten für die Partner-Komponente erfolgt über Kontrollstrukturen. Sie informieren über Größe und Adresse innerhalb des gemeinsamen Speichers. Diese Datenstrukturen werden in der Bibliothek verwaltet und sind durch Funktionsaufrufe nutzbar.

Ebenso wie die Initialisierung dieser Strukturen wird auch das Einblenden des gemeinsamen Speichers durch die Bibliothek implementiert.

Bearbeitung einzelner Datenpakete

Eine einzelne Übertragung wird als ein Paket bezeichnet. Auch diese Datenstruktur wird von der Bibliothek zur Verfügung gestellt und kann mittels entsprechender DSI-Funktionen bearbeitet werden.

Synchronisationsmechanismen

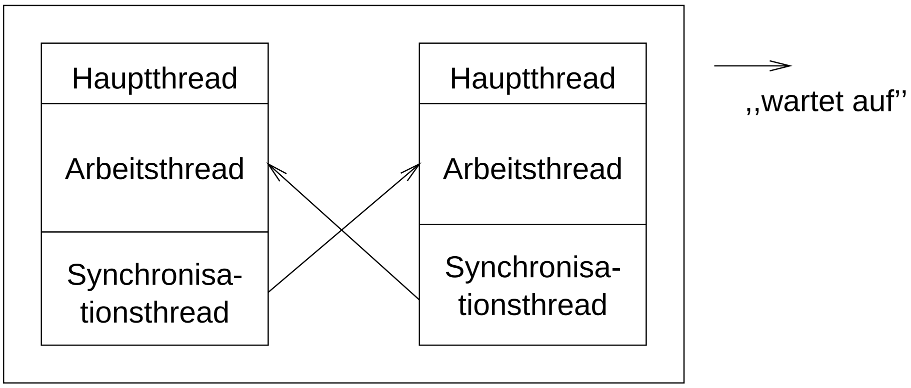

Abbildung 2.8.: Interaktion der Threads, die DSI vorschreibt

Das Konzept von DSI sieht eine bestimmte Thread-Struktur vor, unter anderem ist ein Synchronisationsthread notwendig (siehe Abbildung 2.8). Da dieser für jede Komponente gleich ist, wird er von der Bibliothek implementiert und automatisch gestartet.

Jede Komponente muß je zwei Threads für jeden Strom bereitstellen. Der eine ist der Arbeitsthread, der Pakete erzeugt und bereitstellt bzw. anfordert und verarbeitet. Der zweite ist der Synchronisationsthread, welcher von der Gegenseite aufgerufen wird, wenn ein Unter- oder Überlauf an Paket-Deskriptoren eintritt. Je Strom existieren also 4 Threads, die paarweise aufeinander warten.

Es ist nicht erforderlich, den Arbeitsthread für jeden Strom neu zu erzeugen. Einer kann die Bearbeitung mehrerer Ströme übernehmen. Nur der Synchronisationsthread ist eindeutig einem Strom zugeordnet.

3. Entwurf

Nach der Einführung in die Grundlagen werde ich nun beschreiben, wie ein Videokonferenzsystem aussehen kann, welches auf L4 als Systemkern aufsetzt. In einem zweiten Abschnitt gehe ich genauer auf den Entwurf einer Videoübertragung mittels des *DROPS Streaming Interface* (DSI) ein.

Ein dritter Abschnitt beschreibt, wie der Medienserver in das Gesamtsystem integrierbar ist.

3.1. Videokonferenz

In Abschnitt 2.2 habe ich ausführlich die Bestandteile eines H.323-Systems beschrieben. Hier gehe ich nun auf eine mögliche Realisierung eines solchen Systems ein. Zur Bezeichnung der Komponenten nutze ich die Termini des Standards.

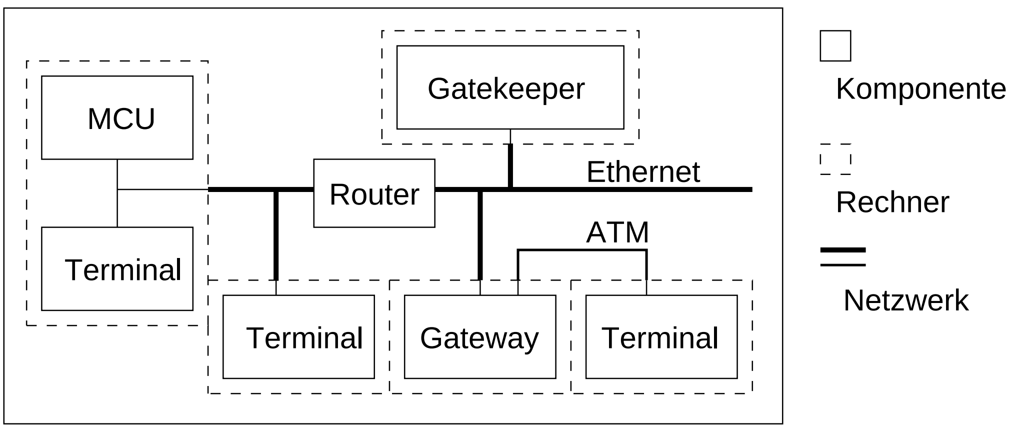

Abbildung 3.1.: H.323-Videokonferenz und deren mögliche Aufteilung auf 5 Rechner

Die Abbildung 3.1 stellt das Beispiel eines H.323-Systems dar, wie es bereits in der ausführlichen H.323-Beschreibung von Abschnitt 2.2 (Abb. 2.1) beschrieben ist. Hier ist die Grafik um eine mögliche Rechneraufteilung erweitert. Andere Kombinationen sind nicht ausgeschlossen, da die H.323-Komponenten nur logisch voneinander getrennt sind. Das Beispiel enthält alle H.323-Grundkomponeten und soll für die folgende Beschreibung der Realisierung auf L4 als Grundlage dienen. Das System enthält *Terminals* im lokalen Netz und, mittels der Vermittlung durch das *Gateway*, auch über eine ATM-Verbindung. Des weiteren sind *Multipoint Control Unit* (MCU) und *Gatekeeper* vorgesehen.

3.1.1. Protokolle

Ein Hauptbestandteil jeder Komponente eines H.323-Konferenzsystems ist eine Protokollbibliothek, welche die Kommunikation abwickelt. Da sie von allen zu realisierenden Endpunkten gebraucht wird, soll ihr zuerst Beachtung geschenkt werden.

Die Protokolldefinitionen aus H.245 und H.225.0 liegen in der Datenbeschreibungssprache ASN.1 vor. Für diese ist eine Anwendung (Compiler) zu entwickeln, welche zwei Aufgaben auszuführen hat. Zum ersten die Übersetzung der verschiedenen Nachrichten in eine hardwareunabhängige Folge von Zeichen, die über das Netzwerk zu anderen Endpunkten übertragen wird. Andererseits muß sie eine empfangene Zeichenfolge als eine Nachricht mit den einzelnen Parametern interpretieren.

Im Rahmen des Projektes openH323 wird ein Programm (`asnparser`) entwickelt, welches aus den Definitionen, die in ASN.1 vorliegen, C++-Implementierungen generiert. Dieses Programm kann als Ausgangspunkt für die Nutzung unter L4 dienen. Für den direkten Einsatz auf einem L4-basierten System ist dieses Programm nicht geeignet, da die generierten Funktionen die Nutzung einer Bibliothek voraussetzen, welche nur für Linux- und MS Windows-Systeme existiert. Es ist daher eine Anpassung der Codegenerierung oder eine Nachbildung der Bibliothek auf L4 notwendig.

Die Mediendaten werden mittels des auf UDP/IP aufsetzenden *Real Time Protocol* (RTP) übertragen. Die L4-Server, welche diese Daten liefern, also Audio- und Video-Codecs, sollen jedoch nicht selbst RTP implementieren. Sie sollen unabhängig von ihrem hier beschriebenen Einsatz bleiben, um beispielsweise den kodierten Strom in einer Datei sichern zu können.

Es gibt nun noch zwei Möglichkeiten der RTP-Umsetzung. In bisher existierenden Projekten wird RTP in der Regel von der Anwendung selbst realisiert, da die UDP-Implementierungen der meisten Betriebssysteme RTP nicht unterstützen. Auf die Realisierung im L4-System übertragen erfordert dieses Vorgehen die Entwicklung eines einzelnen Servers, der die Daten in RTP kapselt und mittels der DROPS-Standardschnittstellen (DSI) eingebunden wird. Dies hat den Vorteil der klaren Trennung vom Netzwerk-Server (`tcpip`). Allerdings verursacht dies auch erhöhten Kommunikationsaufwand bei der Vermittlung der Daten von RTP- zum `tcpip`-Server. Daher ist die zweite mögliche Implementierung vorzuziehen, bei der RTP in den existierenden TCP/UDP/IP-Stack integriert und über zusätzliche Schnittstellen zugänglich gemacht wird.

Diese Erweiterung wird von verschiedenen Komponenten gleichzeitig genutzt und muß deshalb mit mehreren unabhängigen Strömen arbeiten können.

Alle Protokolle setzten auf TCP/UDP/IP auf. Daher ist es die Aufgabe der Bibliotheken, mit dem Netzwerkserver zu kommunizieren, um die Nachrichten ins Netzwerk zu übertragen.

3.1.2. Terminal

Das Terminal (Abbildung 3.2) ist die Schnittstelle zum Anwender der Videokonferenz. Es stellt die gelieferten Videodaten und Statusmeldungen auf dem Bildschirm dar und leitet die Audiodaten an die Hardware weiter. Gleichzeitig müssen die eigenen, von einer Kamera und einem Mikrofon gelieferten, Mediendaten für die Übertragung aufbereitet und über das Netzwerk verschickt werden. Das Terminal bietet eine Bedienungsschnittstelle an, welche es dem Nutzer ermöglicht, Verbindungen zu anderen Endpunkten aufzubauen. Zum Aufbau einer Verbindung gehören unter anderem: Registrierung beim Gatekeeper, Anfrage nach möglichen Konferenzpartnern, Verbindung mit einem bestimmten Terminal, Akzeptieren des Verbindungswunsches eines anderen Terminals, Festlegung von Verbindungsparametern, etc.

Mit Hilfe der im vorigen Abschnitt beschriebenen Bibliotheken wird das Terminal in die Lage versetzt, die Nachrichten entsprechend der Nutzeranweisung generieren und versenden zu können (markiert mit 1 in der Abbildung 3.2).

Während des Aufbaus der H.323-Konferenz müssen die Medienströme bereitgestellt werden. Der Standard H.323 schreibt vor, daß jeder Endpunkt Audio-Ströme nach G.711 liefern bzw. verarbeiten können muß. Der Codec (2) wandelt ein Audio-Signal (3) in einen G.711-Datenstrom (4) und umgekehrt. Diese beiden Datenströme werden mit Hilfe des DSI ausgetauscht. Aufgabe des Terminals ist es nun, die L4-Server untereinander und dann mit der Netzwerkkomponente (5) zu verbinden (6 — DSI-`connect`).

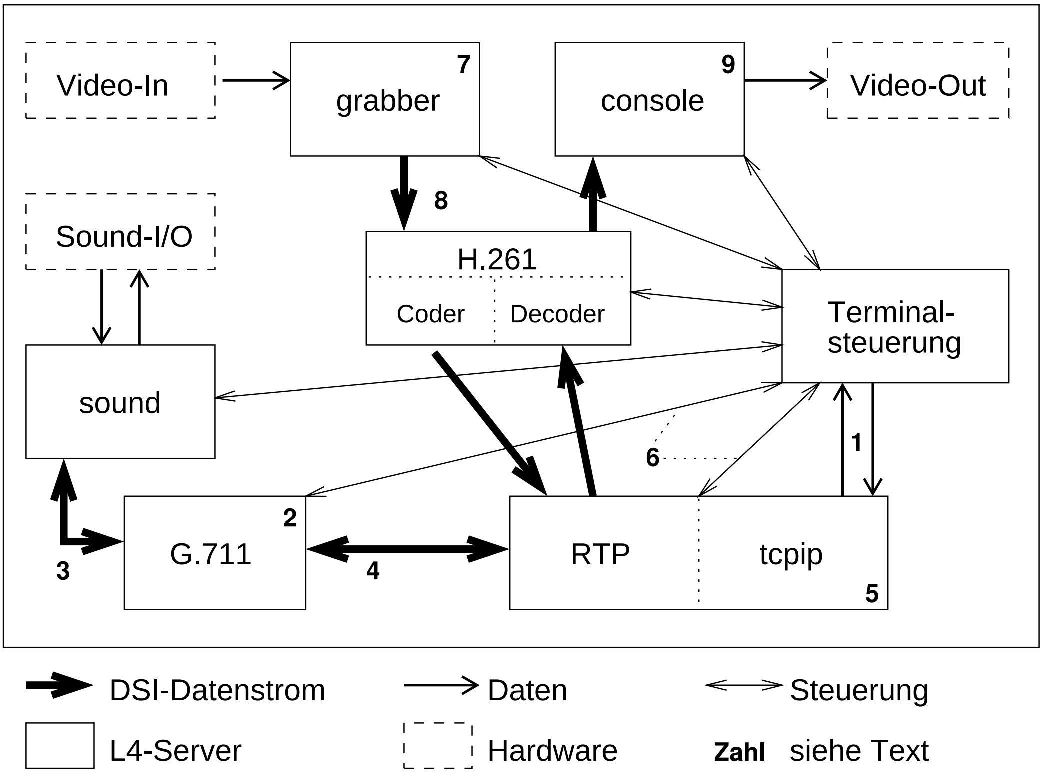

Abbildung 3.2.: Aufbau eines Nutzerterminals auf L4

Für die Bereitstellung eines Videostromes ist ein Server (7) notwendig, der die Framegrabberkarte steuert und über die DSI-konforme Schnittstelle (8) mit dem Codec für H.261 verbunden wird. Dieser wiederum wird, wie der Audio-Codec, mit dem L4-Server verbunden, der die Videodaten ins Netzwerk leitet. Andererseits müssen die vom Netzwerk gelieferten Daten dekodiert und dargestellt werden. Dazu wird der H.261-Server ein zweites Mal, nun jedoch in entgegengesetzter Richtung, mit der RTP-Komponente verbunden. Zur Darstellung ist ein Grafikkarten-Treiber (9) notwendig, der mit dem H.261-Server verbunden wird.

3.1.3. Gatekeeper

Die Entwicklung eines Gatekeepers ist zwar nicht essentiell für die Funktion einer Videokonferenz, im Rahmen des Projektes DROPS jedoch besonders interessant. Wie bereits beschrieben bietet er neben dem Namensdienst die Möglichkeit der Qualitätssicherung für die kommunizierenden Endpunkte.

Voraussetzung für *Qualitiy of Service* in einem nicht garantierenden Netzwerk ist, daß alle im Netzwerk kommunizierenden Endpunkte die Reservierungen, welche sie beim Gatekeeper vollzogen haben, nicht überschreiten. Um dies gewährleisten zu können, müssen auch alle Komponenten eines Rechners in der Lage sein, eine Abschätzung der Dauer einer zuverlässigen Ausführung einzelner Aktionen durchführen zu können. Dies ist aber für den Netzwerktreiber nur möglich, indem alle Treiber in einem Segment gemeinsam über die zeitliche Nutzung ihrer Ressource entscheiden. Hat ein Treiber die Nutzungszeit oder Datenmenge, die im zusteht, überschritten, ist er verpflichtet, alle weiteren Daten zurückzuweisen.

Die Reservierungsmechanismen, welche DROPS bietet, erlauben es nun, alle beteiligten Komponenten zur Einhaltung der genannten Zusagen zu zwingen.

Abbildung 2.4 (siehe Seite 13) zeigte schon den strukturellen Aufbau eines Gatekeepers. Er ist wie alle anderen H.323-Endpunkte eine logische Einheit, die auf einem Rechner mit anderen Komponenten zusammenarbeiten kann. In einem Verbund vieler Teilnehmer ist jedoch eine hohe Last zu erwarten, und der Gatekeeper sollte somit einzige Anwendung eines Rechners sein.

Grundlegend für die Funktion eines Gatekeepers ist der *RAS-Manager*. Er ist für die Kommunikation des Gatekeepers mit anderen H.323-Endpunkten verantwortlich. Basierend auf den beiden Bibliotheken für die Protokolle entsteht so eine Anwendung, welche die Netzwerkpakete vom `tcpip`-Server entgegennimmt, auswertet und beantwortet bzw. an einen Service-Thread weiterreichen kann.

Vom RAS-Manager selbst zu beantworten sind die Registrierungsanfragen (GRQ, ARQ, etc.). Dazu muß er einen Namensdienst implementieren, der einem Namen einen *Transport Layer Service Access Point* (TSAP) zuordnet.

Daneben sind Module notwendig, die den Verbindungsaufbau für alle registrierten Endpunkte übernehmen. Dabei kann ein solcher Prozess vom RAS-Manager initiiert werden, sollte dann jedoch von einem separaten Thread abgewickelt werden, um den RAS-Manager nicht zu blockieren.

Ein weiteres Modul ist der H.245-Manager. Er vermittelt die Kontrollkanäle verschiedener Verbindungen. Diese Funktionalität sollte in einen oder mehrere Threads ausgelagert werden.

Der wichtigste Service-Thread ist der *Policy-Thread*. Er ist für die Verwaltung der vorhandenen Netzwerkressourcen zuständig. Er weist auf entsprechende Anfragen Teile der Ressourcen einer Kommunikationsbeziehung zu. Es muß also ein Verzeichnis der vorhandenen Netzwerkverbindungen implementiert werden, zu denen die Bandbreiten verwaltet werden.

So sind auch weitere Service-Threads in den Gatekeeper integrierbar, die für die Verarbeitung weiterer Dienste zuständig sind, wie z. B. die Sammlung von Verbindungsstatistiken zur Abrechnung beim Nutzer. Es müssen aber nicht alle möglichen Fähigkeiten implementiert werden. Einige, wie *Admission* und *Bandwidth Control,* fordern nur, daß die Anfrage nach diesem Service zumindest verstanden und mit einer Zusage beantwortet wird. Andere Services (*Call Control Signalling, Call Authorization, Bandwidth Management*) sind generell fakultativ.

3.1.4. MCU

Eine *Multipoint Control Unit* (MCU) besteht aus einem *Multipoint Controller* (MC) und einigen *Multipoint Porzessors* (MP). Dabei werden die Daten vom MC empfangen, an den oder die MP zur Verarbeitung weitergereicht und wieder vom MC ins Netzwerk übertragen. Die Daten werden in einer bestimmten Reihenfolge von den verschiedenen Komponenten bearbeitet, welche aber alle unabhängig voneinander sind. Damit ist die MCU von sich aus so strukturiert, daß das DSI die Verbindung übernehmen kann. In der Startsequenz werden die einzelnen Komponenten aus MCs und MPs zu einem Strom verkettet, die fortan autonom arbeiten.

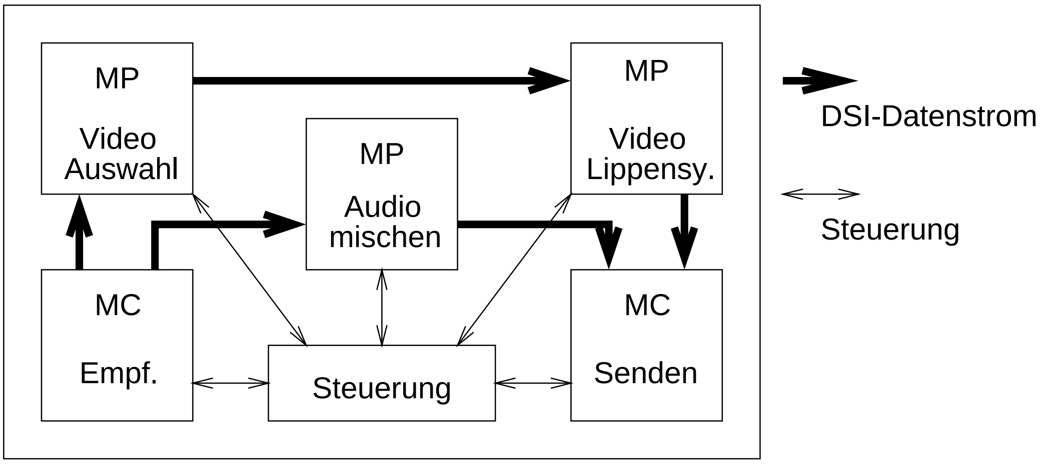

Abbildung 3.3.: Möglicher Aufbau eines MCU unter L4

Die Verarbeitung eines Konferenzstromes nimmt eine längere Zeit in Anspruch. Wenn beispielsweise ein MP die Lippensynchronität zwischen Audio- und Video-Daten sichern soll, ist eine Pufferung von Daten notwendig. Damit ist die Trennung von Daten, die nicht zur gleichen Konferenz gehören, aber über diese MCU ausgetauscht werden sollen, kompliziert. Wenn jedoch für jede neue Konferenz eine neue Instanz einer MCU erzeugt wird, können alle Konferenzen differenziert voneinander behandelt werden. Daraus folgt, daß die MCU-Applikation instanziierbar sein muß.

3.1.5. Gateway

Die Anbindung von Endpunkten anderer Konferenzstandards erfordert die Entwicklung eines Gateways. So ist für die Nutzung einer reinen ATM-Verbindung eine Vermittlung zum Standard H.321 notwendig.

Von einem H.323/H.321-Gateway werden folgende Fähigkeiten verlangt:

- Video- und Audioformat-Konvertierung (beidseitige Nutzung von z. B. H.261 ist möglich)
- Konvertierung des Datenstromes auf Netzwerkebene (H.221 vs. H.225.0)
- Vermittlung der Kontrollsignale (H.245 vs. H.242)

Somit sind Komponenten zu entwickeln, welche, wie bei H.323, die Protokolle kapseln. Sie können anschließend zu einer Kette in einem Datenstrom verbunden werden: Der Ethernet-Server empfängt die Daten und leitet sie weiter an die Komponente zur Konvertierung der Protokolle und Medienformate. Diese gibt die Daten, welche nun für H.321 aufbereitet sind, in das ATM-Netzwerk weiter. Die Gegenrichtung ist äquivalent aufzubauen.

3.2. Videostrom über DSI

Das *DROPS Streaming Interface* ist die Standardschnittstelle zum Datenaustausch zwischen L4-Servern und somit wesentlicher Bestandteil der Videokonferenz. Im folgenden möchte ich daher näher auf die Realisierung eines Datenaustausches mit DSI eingehen. Ich werde am Beispiel einer Videoübertragung zwischen zwei Rechnern den Entwurf von Komponenten beschreiben, welche die DSI-Bibliothek nutzen wollen. Das Ziel ist, mit Hilfe der DSI-Mechanismen zur Synchronisation der Kommunikationspartner die Rate zu regulieren, mit der Bilder auf dem Senderrechner generiert werden. Dabei soll sowohl auf unterschiedliche Netzwerklast als auch auf die Aufnahmefähigkeit des Abnehmers Rücksicht genommen werden.

Ausgangspunkt sind drei der vorhandenen L4-Server: `meteor` (Treiber für die Meteor Framegrabber-Karte), `ether` (Treiber für Ethernet-Karten) und `console` (Treiber für die Grafikkarte S3 968). Auf einem der beiden Rechner werden Bilder von einer Live-Kamera durch `meteor` erzeugt und zu einem zweiten Rechner übertragen, der die Daten mittels `console` darstellt. Für die Übertragung der Daten zwischen den Rechnern ist `ether` zuständig. Es werden also direkt Ethernet-Frames erzeugt und an den Treiber geliefert.

> Anmerkung: Im folgenden Abschnitt werde ich "Bild" als Bezeichnung für ein Datenpaket verwenden, welches nicht im Ganzen über das Netzwerk übertragen werden kann. Das bedeutet jedoch nicht, daß die beschriebene Entwicklung auf die Übertragung von Bilddaten beschränkt ist.

3.2.1. Rollen

Für DSI existiert in Bezug auf einen Strom immer ein Sender und ein Empfänger. Somit fungiert `meteor` immer als Sender und `console` immer als Empfänger. `ether` muß beide Rollen ausfüllen können, auf der Erzeugerseite als Empfänger (von `meteor`) und auf der anderen Seite als Sender.

Es ist im allgemeinen davon auszugehen, daß die beschriebenen Treiber gleichzeitig eine DSI-konforme Datenschnittstelle besitzen. Daher werde ich im Weiteren nicht mehr zwischen dem eigentlichen Treiber und der DSI-Erweiterung unterscheiden. Auch wenn es zwei Anwendungen sein können, bilden sie doch eine Einheit, und ich bezeichne sie daher mit `meteor`, `ether` und `console`.

Das äußere Design der Komponenten ist recht einfach. `meteor` wartet auf das nächste Bild, welches die Hardware liefert, kopiert es in den DSI-Puffer und übergibt den Speicherbereich mittels eines neuen Kontrollblockes. `console` wartet auf die Freigabe eines neuen Datenblockes und kopiert ihn in den Grafikbereich.

`ether` muß in die Kette zwischen `meteor` und `console` gestellt werden. Die Schwierigkeit dabei ist, daß die Paketgröße des Ethernet mit 1514 Byte deutlich kleiner ist als ein einzelnes Bild von `meteor` mit beispielsweise 10800 Byte (120x90 Punkte, 8 Bit Farbtiefe). Es gibt nun verschiedene Möglichkeiten, dieses Problem zu lösen:

1. Es wird ein Protokoll festgelegt, daß zwischen `meteor` und `console` die Zusammengehörigkeit einzelner Teile kennzeichnet. Die Protokollsteuerung wird in den Datenstrom eingewoben.
   Dies hat den Vorteil, daß `ether` nichts über die zu übertragenden Daten wissen muß und daher einfach und schnell arbeiten kann. Der große Nachteil ist jedoch, daß zumindest `meteor` Informationen über die zwischengeschalteten Module braucht, um die Stückelungsgröße festlegen zu können. Dies widerspricht aber dem Anspruch auf Transparenz von DSI.

2. Die Stückelung übernimmt `ether`. `meteor` liefert komplette Bilder an einen für ihn beliebigen Empfänger, in diesem Falle ist dies `ether`. `console` bekommt die einzelnen Teile eines Bildes als (Scatter-Gather-)Liste innerhalb eines DSI-Paketes übergeben.
   Vorteil: Das Netzwerk kann optimal genutzt werden, da `ether` (oder auch eine andere Netzwerk-Komponente) die maximale Übertragungsgröße genau kennt. Anhand fehlender Teile in der Liste kann `console` Verluste in der Übertragung erkennen und unnötiges Umkopieren in den Grafikspeicher vermeiden.
   Nachteil: `console` muß in der Lage sein, zerstückelte Pakete zu verarbeiten. Soll eine andere Komponente der `console` vorgeschaltet werden (z. B: ein H.261 Codec), muß die Fragmentierung von dieser behandelt werden. Auch hier geht Transparenz verloren.

3. `ether` übernimmt komplett Fragmentierung und Reassemblierung der Pakete. Der Vorteil besteht darin, daß die gesamte Intelligenz für die Übertragung in einer Komponente konzentriert wird. `meteor` und `console` können einfach aufgebaut werden.

Somit wird maximale Transparenz im System erreicht.
Nachteil: Verlorene Pakete müssen `console` als vorhanden erscheinen, da sie immer vollständige Bilder erwartet Es bleiben die Möglichkeiten, den entsprechenden Teil des vorausgegangenen Paketes erneut zu vermitteln oder einen leeren Bereich zu übergeben. In jedem Fall muß `console` diese Daten erneut in den Grafikspeicher kopieren, obwohl sie keine (zusätzliche) Information bieten.

4. Eine zusätzliche Komponente übernimmt vollständig die Aufteilung und Reassemblierung der Paket für die Netzwerkkomponente. Sie ist parametrisierbar zu gestalten, um sie unabhängig von der in dieser Demonstration notwendigen Teilungsgröße nutzen zu können.
Der Vorteil dieses Verfahrens liegt in der verfeinerten Strukturierung des System. `meteor`, `ether` und `console` sind nur mit jeweils einer Aufgabe betraut: Erzeugung, Übertragung bzw. Darstellung von Daten.
Als Nachteil kann sich aber der erhöhte Kommunikationsaufwand erweisen. In dem Fall, daß der Abnehmer nicht bei der Verarbeitung nach kommt, verlängert sich die Zeit, in welcher der Erzeuger davon Kenntnis erlangt.

Wegen der hohen Transparenz für `meteor` und `console` und einem genügenden Demonstrationseffekt in Bezug auf DSI habe ich mich für die Variante 3 entschieden. Der Mehraufwand beim Kopieren ist Dank schneller Kopierimplementierungen kaum spürbar.

Bei dieser Variante, die beiden Instanzen von `ether` (je eine auf Sende- und Empfangsrechner) als eine Einheit nach außen darzustellen, ist es notwendig, daß beide Instanzen Informationen austauschen. Sie unterrichten beispielsweise die Senderseite, daß der Abnehmer nicht so viele Daten verarbeiten kann, wie gerade übertragen werden. Um die Nutzrate des Netzwerkes maximal zu halten, sollen sich diese Nachrichten auf das absolute Minimum beschränken.

3.2.2. Synchronisation

Die Synchronisationsmechanismen erfordern eine genauere Betrachtung. Ziel ist es, dem Server `console` genau so viele Bilder zu liefern, wie Netzwerk und Rechner verarbeiten können. Die erste Möglichkeit ist, einen zweiten, entgegengesetzt zum Datenstrom gerichteten Strom zu etablieren, über den jedes verarbeitete Bild bestätigt und mit der Information über freie Ressourcen versehen wird. Damit ließe sich sehr genau die Rate regeln, was als Vorteil zu werten ist. Diese Herangehensweise verursacht jedoch einen erhöhten Aufwand für den Anwender. Er allein kennt die beteiligten Komponenten und muß somit diesen Bestätigungskanal aufbauen.

Zudem werden so nicht alle Fähigkeiten des DSI genutzt. Dieses implementiert, wie beschrieben, einen Synchronisationsthread, der die Gegenseite über Ereignisse informiert, die erkennen lassen, daß eine Seite mit der Verarbeitung überfordert bzw. nicht ausgelastet ist. Es läßt sich folgende Reaktionskette realisieren:

console Der Server entnimmt fortlaufend ein Paket und blockiert, wenn kein Deskriptor vorhanden ist, bis der nächste bereitgestellt wird. Damit wird `ether` signalisiert, daß zu wenig Pakete geliefert werden.

ether Bekommt `ether` beim Bereitstellen des nächsten Paketes für `console` die Mitteilung, daß auf dieses Paket bereits gewartet wurde, kann er dem `meteor`-Server signalisieren, daß er die Rate erhöhen kann, wenn das Netzwerk nicht schon voll ausgelastet ist.
Stellt er dagegen fest, daß er keinen neuen Deskriptor belegen kann, so heißt das, daß `console` bei der Abarbeitung der Pakete nicht folgen kann. Die Rate muß gedrosselt werden.

meteor Je nachdem, ob ein Über- oder Unterlauf an Deskriptoren festgestellt wird, kann `meteor` die Rate senken oder erhöhen.

Da die Synchronisation nur über das Eintreten eines Ereignisses informiert, nicht jedoch über das Maß der Überlast bzw. freien Ressourcen, kann die Regelung nicht so genau vorgenommen werden: Die Senderseite muß versuchen, die optimale Rate zu finden, was ein Nachteil ist. In dieser reinen Videoübertragung soll jedoch das DSI demonstriert werden. Daher habe ich diese Variante implementiert.

3.3. Integration Medienserver

Seit einigen Jahren gibt es an der Fakultät Informatik einen Sonderforschungsbereich (SFB), der sich in einem Teilbereich die Entwicklung eines Medienservers zum Ziel gesetzt hat. Der Medienserver speichert eine Sammlung von Videofilmen in komprimierter Form. Einzelne Clients können das Abspielen eines Films zu einer nahezu beliebigen Zeit anfordern. Eine mögliche Anwendung des Servers ist die Nutzung als Teil der Videokonferenz. So können sich die Konferenzteilnehmer gemeinsam einen Film ansehen oder der Vortrag eines Teilnehmers soll als Film abgespeichert werden. Die Integration mit dem hier beschriebenen Konferenzsystem kann unter bestimmten Voraussetzungen umgesetzt werden, und ich werde sie im folgenden benennen. Zunächst müssen dazu jedoch die Unterschiede herausgestellt werden.

Während der Laufzeit des Films wird ein Ausfall einzelner Bilder weniger toleriert, als bei einer Videokonferenz. Der Ausfall läßt sich aber nicht ausschließen, wenn ein nicht garantierendes Netzwerk zum Einsatz kommt. Er läßt sich jedoch begrenzen, wenn man den Service der Bandbreitenreservierung des Gatekeepers nutzt.

Wenn man den reinen Medienstrom betrachtet, ist ein abgespeicherter Film nichts anderes, als ein nicht interaktiver Konferenzteilnehmer. Das Ende des Films ist mit dem einseitigen Abbau der Verbindung gleichzusetzen. Der Verbindungsaufbau nimmt eine gewisse Zeit in Anspruch. Daher sollte der Medienserver erst mit dem Senden des Filmes beginnen, wenn die Bandbreitenreservierung vollzogen ist und die Verbindung zum Klienten endgültig steht. Ansonsten muß damit gerechnet werden, daß der Anfang des Filmes verloren geht. Bei einer Videokonferenz ist dies nicht relevant, da man hier den Beginn der Besprechung durch die menschliche Kommunikation festlegt.

Der Hauptanwenundungsfall des Medienservers soll das gemeinsame Betrachten eines Videofilms mit allen Teilnehmern einer Konferenz sein. Daraus folgt, daß von ihm Mehrpunktfähigkeit verlangt wird.

Um einen bestimmten Film zu wählen, kann auch die H.323-Semantik genutzt werden: Jedem Film wird ein TASP zugeordnet und mit einem Namen beim Gatekeeper registriert.

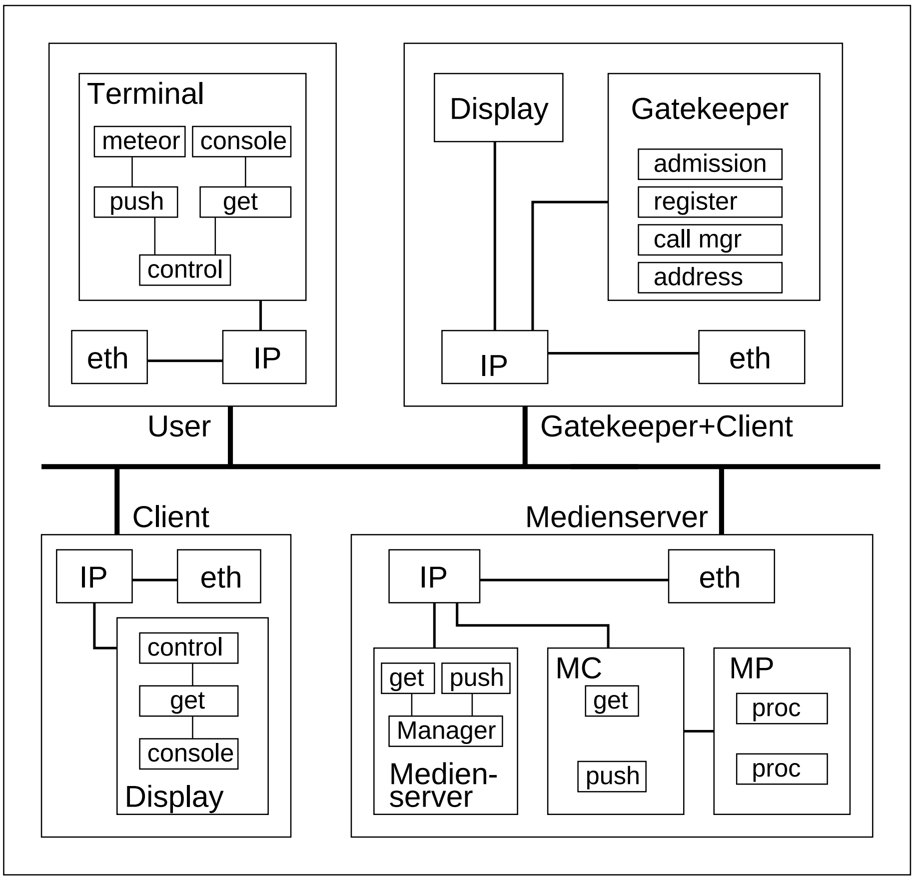

Abbildung 3.4.: Rechneraufteilung für eine mögliche Realisierung eines Videokonferenzsystems mit integriertem Medienserver

Die Abbildung 3.4 zeigt eine mögliche Aufteilung der notwendigen Anwendungen und deren schematische Verbindung auf mehrere Rechner. Dabei habe ich kurze Bezeichnungen gewählt, die im folgenden erläutert werden:

Client/Display Rechner/Darstellungskomponente für ausschließlichen Empfang eines Videostromes (Video on Demand)

Gatekeeper Komponente im Sinne von H.323 — Kontrollinstanz

Gateway Komponte im Sinne von H.323 — Vermittlung zu anderen Netzwerken

Medienserver Rechner, der die Komponenten betreibt, die für die Bereitstellung verschiedener Videofilme notwendig sind

MC/MP Komponente im Sinne von H.323 — Anwendung für die Aufarbeitung paralleler Ströme

MCU Komponente im Sinne von H.323 — Rechner für die Nutzung einer Mehrpunktkonferenz

Terminal Komponente im Sinne von H.323 — Nutzerinterface

User ein Rechner, der nur ein Terminal für einen Nutzer betreibt

4. Implementierung

Im folgenden Kapitel beschreibe ich den Implementierungsteil meiner Arbeit. Zuerst lege ich dar, warum die ursprüngliche Aufgabe, ein Videokonferensystem zu entwickeln, nicht im gewünschten Umfang realisiert werden kann, sondern auf den Entwurf beschränkt bleiben muß. Danach gehe ich auf Details der DSI-Demonstration ein.

4.1. openH323

Nachdem sich herausgestellt hatte, daß das Konferenzsystem der Professur Rechnernetze für meine Arbeit nicht mehr maßgebend sein kann, konzentrierte ich mich auf den Standard H.323 und dessen frei verfügbare Implementierung openH323.

Zuerst beschäftigte ich mich eingehend mit den umfangreichen Dokumenten der Standards H.323, H.261, H.225.0, H.245 und H.235 (siehe Literaturverzeichnis und Abschnitt 2.2).

Anschließend habe ich einige Zeit die Entwicklung des Projektes openH323 verfolgt und die Quellen daraufhin untersucht, ob sie sich auf L4 portieren lassen. Leider wird eine sehr komplexe Bibliothek genutzt, die nur für Linux- und MS Windows-Systeme verfügbar ist. Sie bietet neben dem vereinfachten (objektorientierten) Zugriff auf Strings, Listen etc. vor allem eine Threadabstraktion. Auf meine Anfrage in der (sehr aktiven) Mailingliste zu einem Überlick über die Bibliothek und deren Systemabhängigkeit bekam ich keine einzige Antwort.

Die Mechanismen, die in den Standards beschrieben werden, sind als Bibliothek implementiert. Diese bauen wiederum auf die Systembibliothek auf und sind somit nicht direkt nutzbar.

Einige Beispielanwendungen bauen auf die genannten Grundlagen auf und demonstrieren den noch recht geringen Funktionsumfang. Mir ist es im Laufe der Zeit nur wenige Male gelungen, die Original-Quelltexte komplett zu kompilieren und zu starten. Trotz offensichtlich fehlerloser Kompilation wurde die Anwendung wegen Speicherverletzung beendet.

Wie gesagt ist der Funktionsumfang noch eingeschränkt. So ist das Terminal bislang nur zu einer Punkt-zu-Punkt-Verbindung fähig. Eine Zusammenarbeit mit den noch jungen Projekten `opengate` (Gatekeeper) und `openmcu` (MCU) ist mir nicht gelungen.

4.2. DSI-Demonstration

In Abschnitt 3.2 habe ich den Aufbau der DSI-Demonstration beschrieben. Im folgenden gehe ich auf Details der Realisierung ein. Sie sollen die Nutzung des *Streaming Interface* verdeutlichen.

4.2.1. meteor

Für die Bereitstellung eines Videostromes ist ein Server notwendig, der die Framegrabberkarte steuert und Bilder im Speicher bereitstellt. Dafür existiert `meteor`. Dieser Server ist jedoch im Moment so implementiert, daß die Bilder im Speicher einer anderen Task bereitgestellt werden. Diese Task kann nun ihrerseits das DSI implementieren. Dieses Vorgehen hat aber den Nachteil, daß jedes Bild kopiert werden muß, vom Bereich, der vom Meteor-Treiber bereitgestellt wird, in den Bereich, welchen DSI zur Datenübertragung nutzt. Beide Speicherbereiche werden zwar von ein und demselben Speichermanager bereitgestellt, jedoch nicht auf Anforderung vom Anwender direkt, sondern intern von den Bibliotheken zu `meteor` und DSI.

Es ist aber auch möglich, daß der Meteor-Treiber selbst um das DSI erweitert wird. Dies erspart das Umkopieren der Daten, erfordert jedoch eine Restrukturierung des Meteor-Servers. Er muß veranlaßt werden, seine Daten in den vom DSI bereitgestellten Puffer zu schreiben. Für die Realisierung im Videokonferenzsystem ist auf jeden Fall die zweite Variante zu wählen. Aus Zeitgründen habe ich jedoch für meine Implementierung die einfache Variante gewählt.

Die DSI-Erweiterung zu `meteor` ist also ein L4-Server mit einem Hauptthread und je zwei Arbeitsthreads pro Strom, wie es DSI vorschreibt. Der Hauptthread registriert sich beim L4-Namensdienst und arbeitet anschließend als Server für die entfernten Aufrufe, die in der IDL[1] spezifiziert und offengelegt sind. Die angebotenen Funktionen heißen `open`, `connect` und `start`.

`open` fordert den gemeinsamen Speicher von Sender und Empfänger an und erzeugt einen neuen Arbeitsthread[2] mit dem notwendigen Arbeitsspeicher.

`connect` erwartet Informationen über die Gegenstelle des Stromes, damit beiden Komponenten die Kommunikationspartner bekannt sind.

Der erzeugte Arbeitsthread initialisiert den Meteor-Treiber und wartet bis zum Aufruf der Funktion `start`, bevor er mit dem Versenden der Pakete beginnt. Dies ist notwendig, um bei einer längeren Initialisierungsphase des Empfängers diesen nicht mit Daten zu beliefern, die er noch nicht verarbeiten kann.

Die Initialisierung des Treibers im Arbeitsthread ist ungünstig, da so nur ein Arbeitsthread erzeugt werden kann. Bei dem Ansatz, DSI-Erweiterung und Treiber zu trennen, ist jedoch keine andere Lösung machbar, da der Meteor-Treiber nur an genau

[1] *Interface Description Language* — IDL erlaubt die Festlegung von Schnittstellen, die wie normale Funktionsaufrufe genutzt werden können. Ein externes Programm erzeugt zu der Schnittstellendefinition Module für Client und Server, die den Funktionsaufruf in einer Task (Client) annehmen und in einer anderen Task (Server) die eigentliche Implementierung der Funktion aufrufen.

[2] Der zweite Arbeitsthread (Synchronisationsthread) für diese Verbindung wird von der DSI-Bibliothek automatisch erzeugt und gestartet.

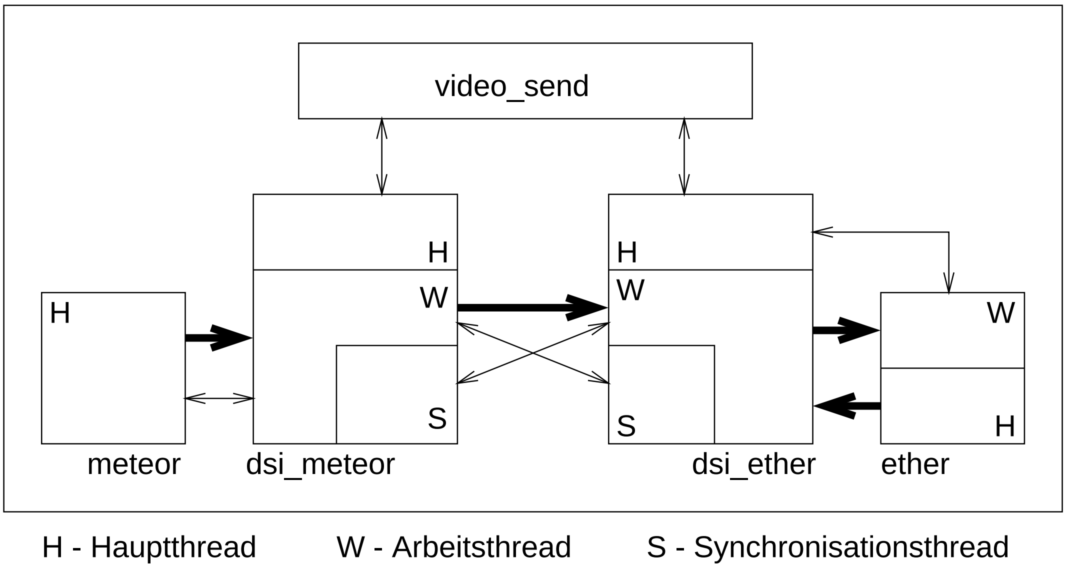

Abbildung 4.1.: Task- und Threadstruktur eines Rechners am Beispiel der Senderseite der DSI-Demonstration

einen Thread Daten liefert, und zwar an den, der ihn initialisiert hat. Für die Demonstration sind parallele Stöme aus `meteor` aber auch nicht notwendig, und daher ist diese Implementierung vertretbar, bis die Integration des DSI in den Treiber erfolgt ist.

Nach dem Empfang des Start-Signals ermittelt der Thread die Adressen des gemeinsamen Datenbereiches, der jetzt in beiden Komponenten initialisiert ist, und betritt die Arbeitsschleife.

Der Aufruf der Funktion `meteor_wait_for_signal` kehrt zurück, sobald das nächste Bild von der Hardware bereitgestellt wurde. Anschließend wird ein neuer Paket-Deskriptor angefordert. Enthält dieser die Information, daß die Gegenstelle bereits auf das Paket wartet, und tritt dies mehrfach in Folge auf, wird die Karte über den Treiber angewiesen, die Rate zu erhöhen, mit der die Bilder generiert werden[3]. Danach werden die Bilddaten in den DSI-Puffer geschrieben und für die Gegenstelle freigegeben.

console

Der Aufbau der Erweiterung für `console` ist der des Meteor-Treibers ähnlich. Im Gegensatz zu `meteor` ist die Initialisierung des Displays aber unabhängig vom Aufrufer und wird daher vom Hauptthread ausgeführt, bevor sich dieser beim Namensdienst registriert und die IDL-Server-Schleife betritt. Die Schnittstellen hier heißen `open` und `connect`.

[3]Hier habe ich eine Obergrenze von 19 Bildern/Sekunde (dies entspricht bei der verwendeten Bildgröße ca. 3,2 MBit/s) eingebaut, da sonst der Ethernet-Treiber des Empfängers zu oft versagte. Es besteht das Problem, daß der Treiber feststellt, daß ein neues Paket angekommen ist, bevor das letzte bearbeitet wurde. Passiert dies oft nacheinander, bleibt der Server sporadisch stehen, und es ist keine Übertragung mehr möglich.

open erzeugt einen Arbeitsthread und dessen Arbeitsspeicher. connect initialisiert die gemeinsamen Daten und startet den Arbeitsthread.

ether

Auch die Erweiterung des Ethernet-Treibers besteht aus Haupt- und Arbeitsthreads. Auch hier sind Initialisierung der Hardware und IDL-Sever im Hauptthread implementiert.

Da ether sowohl als Sender als auch als Empfänger arbeitet, muß er auch zwei verschiedene Schnittstellen anbieten. Senderseitig heißen die Funktionen open, connect und start. Als Empfänger bietet er open und connect an.

Es gibt für die beiden Rollen ebenso je eine Implementierung eines Arbeitsthreads. Auch wenn eigentlich auf jeder Seite nur eine Rolle ausgefüllt werden muß, so ist doch immer (auf beiden Seiten) das Starten des Senderthreads notwendig, da ein Rückkanal für den Datenabgleich zwischen den Instanzen notwendig ist. Der Senderthread im Sinne von DSI ist der Empfangsthread für meteor: Er **empfängt** die Daten vom Netzwerk und **sendet** sie weiter (siehe auch Abbildung 4.2).

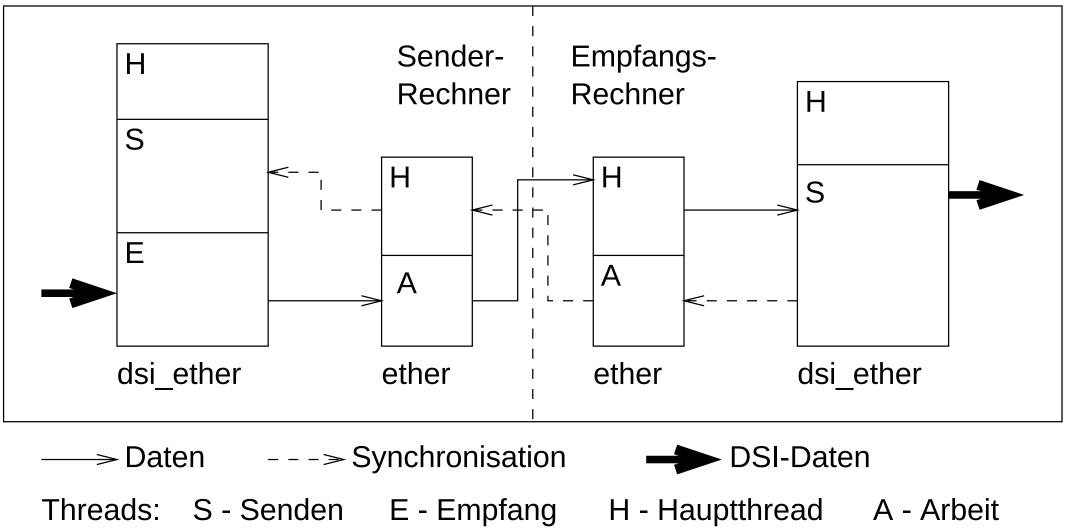

Abbildung 4.2.: Kommunikation zwischen den Instanzen der Ethernet-Komponenten

Die Implementierung der Schnittstellen entspricht im wesentlichen der von meteor und console, und ich gehe deshalb nicht weiter darauf ein.

Der Ethernet-Treiber bietet ebenso ein IDL-Interface für Initialisierung und Datenaustausch. Der Empfänger von Daten muß eine Funktion unter dem Namen stream_io_server_push anbieten, die aufgerufen wird, wenn Daten vorliegen. Im Treiber schickt diese Funktion die Daten in das Netzwerk. In der DSI-Erweiterung kommen so die Daten aus dem Netzwerk an.

video

Das Konzept des DSI sieht vor, daß eine Anwendung bei zwei oder mehr Komponenten den Anschluß für einen Strom anfordert und paarweise verbindet. In meiner Demonstration sind zwei solcher Anwendungen notwendig, je eine auf Sende- und Empfangsrechner. Auf Senderseite werden `meteor` und `ether` verbunden, auf Empfangsseite `ether` und `console`.

4.2.2. Synchronisation

Der Ethernet-Treiber ist so implementiert, daß die Funktion zum Senden ins Netzwerk zurückkehrt, wenn das Paket versandt werden konnte oder in eine Warteschlange gestellt wurde. Im letzten Fall kehrt die Funktion schneller zurück als im ersten. Um den Aufwand einer Zeitmessung und entsprechender Auswertung zu sparen, habe ich das OSKit und den Treiber versuchsweise erweitert, so daß ein eindeutiger Fehlercode zurückgeliefert wird. So ist es mir sogar möglich, weitere Fälle zu unterscheiden. Ist die Warteschlange bereits gefüllt, wird das zu sendende Paket verworfen. Als Anwender zeigt mir dies, daß das Netzwerk total überlastet ist. Das erlaubt mir nun, differenziert auf diese Situationen zu reagieren. Bemerkt die Komponente, daß eine Überlast herrscht, wird der Arbeitsthread für eine gewisse Zeit schlafen gelegt. Die Zeit wird länger gewählt, wenn Pakete bereits verworfen werden.

Daraus resultiert, daß `ether` nicht mehr in der Lage ist, so viele Pakete wie bisher zu verarbeiten. Der Sender, also `meteor`, bemerkt dies durch das Blockieren beim Anfordern eines neuen Paket-Deskriptors und kann entsprechend mit der Absenkung der Rate reagieren; mehr dazu im Abschnitt 4.2.4.

Der eben beschriebene Mechanismus reagiert allein auf die Netzlast. Die Reaktion auf die Nachrichten über die Belastung des Abnehmers (`console`) bedarf weiterer Maßnahmen.

Auf der Abnehmerseite blockiert die DSI-Erweiterung des Ethernet-Treibers für eine gewisse Zeit, wenn kein Deskriptor frei ist, da `console` noch mit der Bearbeitung eines Paketes beschäftigt ist. Diese Zeit wird der Erzeugerseite mitgeteilt. Wenn sich diese für genau diese Zeit schlafen legt, werden vorerst genau im richtigen Maße Daten gesendet. Nur ist der Erzeugerseite nicht bekannt, ob die aktuelle Last das Maximum für den Abnehmer ist. Es wäre daher sinnvoll, entsprechende Nachrichten zur Sender-Instanz von `ether` zu schicken. Dabei ist aber zu bedenken, daß bei dem Versuch, ständig in optimaler Menge Daten zu senden, nahezu jedes Daten-Paket durch eine Nachricht bestätigt wird, was meinem Ansatz, die Kommunikation zwischen den Instanzen zu minimieren, widerspricht.

Ich habe eine einfachere Variante gewählt: Nach einer gewissen Zeit wird die Rate einfach wieder schrittweise erhöht, bis die Gegenstelle einen Überlauf meldet. Die Ergebnisse zweier Meßreihen mit diesem Verhalten werden im Kapitel 5 vorgestellt.

4.2.3. Datenstrukturen

Für die interne Kommunikation zwischen den `ether`-Instanzen sind Protokolle notwendig. Als erstes muß ein Netzwerk-Paket in die Folge der Strom-Pakete einzuordnen sein. Dazu habe ich die folgende Struktur definiert, die an den Beginn jedes Netzwerk-Paketes geschrieben wird.

```
typedef struct ether_packet
{
  struct ether_header eth;
  unsigned packet;
  unsigned frames;
  unsigned frame;
  unsigned offset;
  unsigned char body;
} ether_packet_t;
```

`packet` ist eine fortlaufende, von der Sendeinstanz vergebene, Nummer, welche für die Teile eines Bildes gleich ist. `frame` kennzeichnet die Reihenfolge des Teiles innerhalb des Bildes, `frames` die Anzahl der Teile des Bildes. Der Einfachheit halber wird diese Information in jedem Netzwerkpaket (Frame) mitgeliefert. Andere Varianten, sie nur im ersten Frame eines Bildes mitzuliefern oder nur dann, wenn sich die Anzahl ändert, bergen die Gefahr des Verlustes dieser Information. Es wäre also notwendig, diese Nachricht der Gegenseite zu bestätigen, was einen vergleichsweise hohen Aufwand bedeuten würde (Bestätigung der Bestätigung, usw.) und zeitlich nur schwer abschätzbar wäre.

Der Parameter `offset` übermittelt zur Sicherheit die Position des Frames innerhalb des Bilders. Dies läßt sich einsparen, da sich dieser Wert berechnen läßt. Er sollte in einer optimierten Version folglich nicht mehr mit übertragen werden.

`body` dient als Adressreferenz für den Datenteil des Frames.

Zur Synchronisation der Instanzen sind entsprechende Nachrichten notwendig. Sie werden als einzelne Ethernet-Frames gesendet. Für sie habe ich folgende Datenstruktur definiert:

```
typedef struct ether_sync
{
  struct ether_header eth;
  unsigned char type;
  unsigned long long time;
} ether_sync_t;
```

`type` kennzeichnet einen der beiden Fälle „zu wenig Bilder“ oder „zu viele Bilder“.

`time` übermittelt die Zeit, die das System blockiert, bis ein weiterer Paket-Deskriptor belegt werden konnte bzw. von der Gegenstelle bereitgestellt wurde.

4.2.4. Regulierung der Rate in meteor

Kurzzeitige Belastungen in einem Netzwerk sind sehr wahrscheinlich. Daher wartet `meteor` erst mehrere Synchronisationsnachrichten ab, bevor die Rate gesenkt wird. Dabei ist die Anzahl der auslösenden Nachrichten von der Zeit abhängig, die zwischen zwei Nachrichten liegt. Im Versuch mit verschiedenen Werten haben sich die folgenden als diejenigen erwiesen, die am besten auf den erwarteten Verlauf folgten.

1. Beträgt der Abstand mehr als 3 Sekunden, reicht eine Nachricht aus, um die Rate zu senken.
2. Bei einem Abstand von 1–3 Sekunden sind 3 Nachrichten notwendig,
3. ansonsten 6.

Die Rate wird wieder erhöht, wenn 5 Mal in Folge festgestellt wurde, daß auf das Paket, welches jetzt bereitgestellt wird, schon gewartet wurde.

4.2.5. Regulierung in ether

Wie bereits beschrieben, wird der Instanz auf der Sender-Seite eine Zeitspanne mitgeteilt, die zwischen zwei Bildern vergehen muß. Diese Zeit wird auch beim Senden des nächsten Bildes auf die einzelnen Frames aufgeteilt.

Anschließend wird die Zeit bestimmt, die seit dem Eintreffen der letzten Synchronisationsnachricht vergangen ist. Alle 1,5 Sekunden wird die Wartezeit durch 4 geteilt. Dies erhöht schließlich schrittweise die Rate der Bildgenerierung in `meteor`.

Die Zeit von 1,5 Sekunden und der Teiler von 4 sind, wie bei `meteor`, erprobte Werte, die ich an Hand der Heftigkeit bzw. Trägheit der Reaktion im Gegensatz zur erwarteten Reaktion festgelegt habe.

5. Leistungsmessung und -bewertung

Im folgenden werde ich die Ergebnisse meiner Messungen an der entwickelten DSI-Demonstration nennen und bewerten. Ziel war es dabei, die Möglichkeiten des DSI zu zeigen.

5.1. Reaktion auf Netzlast

Für diesen Versuch wurden vier Rechner in einem eigenen Segment betrieben, um unkalkulierbare Störungen zu vermeiden. Je ein Rechner diente als Sende- bzw. Empfangsrechner mit den in dieser Arbeit beschriebenen Servern auf Fiasco, dem L4-kompatiblen Mikrokern. Die beiden anderen Rechner dienten als gezielte Störer.

Ausgangspunkt ist hier die ungestörte Übertragung eines Videostromes. Nach dem Start mit einer Voreinstellung von 10 Bildern pro Sekunde steigt die Rate auf 17 Bilder/s[1] und bleibt nahezu unverändert.

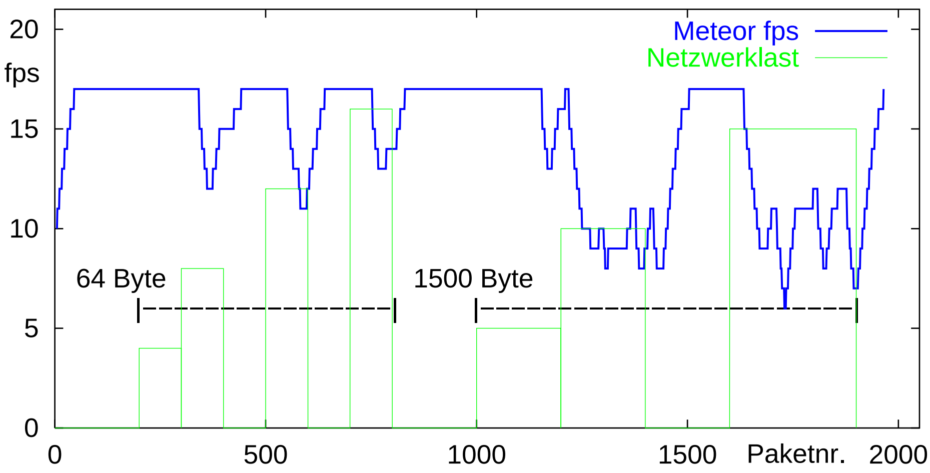

Abbildung 5.1.: Einfluß verschieden starker Netzlasten auf die Bildrate

Nun wurde das Netz verschieden starken Belastungen ausgesetzt. Zuerst mit 1–4 Instanzen eines Ping zwischen den Störrechnern mit normaler Paketgröße (64 Byte —

[1] Die voreingestellte Grenze wurde nicht mehr erreicht, nachdem ich die Wartezeiten bei Netzwerklast geändert hatte. Ich habe die Begrenzung aber beibehalten, da eine Datenrate von 3–4 MBit/s offensichtlich die momentan erlaubte Höchstlast für `ether` ist.

Paketnummern 200–800). Anschließend mit 1–3 Instanzen eines *Flood-Ping* bei einer Paketgröße von 1500 Byte (Paketnummern 1000–1900). Der Einsatz zweier zusätzlicher Rechner bewirkt hier, daß die beiden Rechner unter L4 nicht direkt mit dem Empfang von Paketen beschäftigt werden, sondern nur ein Teil der Netzwerkressourcen für die Videoübertragung übrig bleibt.

Die Grafik in Abbildung 5.1 zeigt das Ergebnis eines Meßdurchgangs. Die Störung wird dabei schematisch als verschieden hoher Balken dargestellt. Der Graph der Meteor-Karte zeigt die aktuelle Rate über der Nummer des Paketes, welches gerade übertragen wird.

Es zeigt sich, daß bei erhöhter Netzlast nicht mehr alle Pakete eines Bildes ausgeliefert werden können. Tritt dies gehäuft auf, beginnt der beschriebene Mechanismus mit der Absenkung der Senderate. Nach dem Wegfall der Störung wird die Rate langsam wieder erhöht.

Es ist auch deutlich die Verzögerungszeit zwischen Einsatz bzw. Wegfall der Störung und der Reaktion darauf zu sehen. Dies ist auf die Pufferung zurückzuführen, die durch das Vorhandensein mehrerer Paket-Deskriptoren sowie dem beschriebenen Algorithmus (Ignorieren einzelner Synchronisationsnachrichten) verursacht wird.

5.2. Reaktion auf die Leistungsfähigkeit

In einem zweiten Versuch habe ich die sich möglicherweise ändernde Leistungsfähigkeit eines Abnehmers simuliert. In der Praxis kann dies z. B. durch schwankende Prozessorlast hervorgerufen werden.

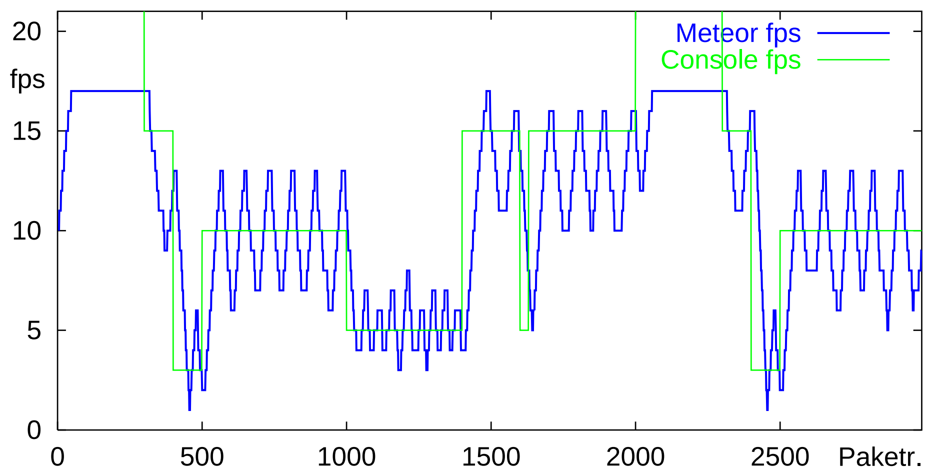

Abbildung 5.2.: Reaktion des Bildgenerators auf die Bildrate, die der Empfänger verarbeiten kann

Die `console` ändert ihre Aufnahmefähigkeit nach einem bestimmten Muster, indem sie sich nach der Verarbeitung eines Paketes für eine gewissen Zeit schlafen legt. Diese

Zeiten wurden so gewählt, daß sie die Aufnahme in etwa auf eine bestimmte Framerate beschränkt. In der Grafik (siehe Abbildung 5.2) wird der Verlauf dieser Beschränkung mit „console“ bezeichnet. Die Reaktion des Meteor-Servers wird vom zweiten Graph beschrieben.

Auch hier zeigt sich die gewünschte Reaktion. Kurz nach dem Einsetzen der Beschränkung wird die Rate so lange heruntergeregelt, bis keine Synchronisationsnachrichten mehr empfangen werden. Danach versucht der Algorithmus die Rate wieder zu erhöhen, bis erneut die Senkung der Rate angefordert wird. Dies wiederholt sich periodisch, was das Auf und Ab in der Grafik erklärt.

5.3. Bewertung der Demonstration

Das beschriebene und getestete System leistet im wesentlichen das Gewünschte: Auf eine Last im Übertragungsweg wird die Datenrate gesenkt. Anschließend wird versucht, die Rate zu erhöhen, so daß sie jederzeit möglichst nahe am Maximum liegt. Man kann in gewisser Näherung davon sprechen, daß sich das System ständig im Gleichgewicht befindet.

Es muß aber auch verdeutlicht werden, daß der Algorithmus noch nicht optimal ist. Die starke Schwankung um das Optimum der Datenrate ist unerwünscht und muß in einer veränderten Version zumindest reduziert werden.

6. Fragen und Ausblicke

Dieser Abschnitt geht kurz auf meine Erfahrungen im Umgang mit den vorhandenen L4-Servern ein und faßt zusammen, was in nächster Zeit im Zusammenhang mit meiner Arbeit zu tun ist.

6.1. L4-Komponenten

6.1.1. DSI

Die aktuelle Implementierung des DSI hat voll und ganz die Entwicklung des gewünschten Systems unterstützt und ließ nur wenig zu wünschen übrig: bislang muß ein bestimmtes Bit in der Paket-Beschreibungsstruktur getestet werden, um zu erfahren, ob auf das Paket bereits gewartet wurde. Dies sollte durch eine zusätzliche Funktion implementiert werden. Dieser Vorschlag wurde bereits zur Diskussion angenommen.

Ein weiteres Problem ist die statische Festlegung von virtuellen Adressen, die von der Bibliothek genutzt werden. Dies führte zu Kollisionen mit den Erfordernissen anderer Komponenten. Es wäre gut, eine Möglichkeit zu finden, welche eine dynamische Adressvergabe ermöglicht.

In seiner bisherigen Form hat die DSI-Bibliothek noch nicht alle geplanten Möglichkeiten implementiert. Sie ist viel mehr Grundlage für die weitere Entwicklung. So ist vorgesehen, die Verwaltung des Pufferbereiches von der Bibliothek ausführen zu lassen. Dies erhöht die Transparenz für die Nutzer. Damit wäre es z. B. möglich, die Datei, welche einen Videofilm gespeichert hat, einem solchen Puffer zuzuordnen. Je nachdem, welchen Teil der Datei das Dateisystem an seinen Partner schicken will, wird dieser automatisch in den Puffer kopiert und über entsprechende Paket-Deskriptoren für den Empfänger nutzbar gemacht.

Einige Anwendungsbereiche der Bibliothek erforden die Möglichkeit des Hinzufügens von Speicherbereichen zum gemeinsamen Pufferbereich. So kann beispielsweise eine Netzwerkkomponente erst nach dem Empfang eines Paketes entscheiden, für welchen Strom diese Daten bestimmt sind. Ist diese Entscheidung getroffen, wird der entsprechende Datenbereich in den Adressraum des Empfängers eingeblendet und so für beide Seiten nutzbar gemacht.

Eine wichtige Option für die Zukunft ist, daß jedes Paket mit einem Zeitstempel versehen werden kann. Damit kann die Gültigkeit einzelner Pakete zeitlich begrenzt werden.

6.1.2. meteor

Wie bereits beschrieben, ist die Meteor-Karte veraltet und nicht mehr lieferbar. Der Treiber für L4 ist eine Ad-hoc-Portierung, die auf Dauer nicht optimal ist.

Video4Linux ist eine generische Schnittstelle zur Ansteuerung von Videokarten unter Linux. Es ist bereits geplant, diese Umgebung auf L4 zu portieren, um einen einfachen Zugang zu aktueller Hardware und deren Treibern zu erlangen. Diese Aufgabe sollte schnell bearbeitet werden. Dabei muß von vornherein eine DSI-kompatible Schnittstelle eingeplant werden.

6.1.3. console

Die von mir genutzte Version des Display-Treibers war sehr unhandlich und statisch. Sie erfüllte jedoch nach dem Überwinden einiger Hürden ihren Zweck.

Seit einiger Zeit ist die Aufarbeitung der Implementierung im Gange. Es bleibt zu hoffen, daß der damit erreichte Stand in Zukunft pflegeleichter ist und auch auf andere Hardware anwendbar wird.

6.1.4. ether

Dieser Treiber ist ordentlich konzipiert und sehr gut nutzbar. Der Ansatz, von Klienten die Implementierung einer Server-Schnittstelle zu fordern, ist jedoch schwer eingängig. Da es jedoch ungewöhnlich ist, Ethernet-Pakete von Hand zu erstellen und zu versenden, wiegt dieser Makel nur wenig. Zudem existiert mit der jungen Entwicklung des TCP/IP-Stack ein deutlich einfacherer Zugang zur Netzwerkkommunikation.

6.2. Videoübertragung

Mit der beschriebenen Implementierung dieser Arbeit existiert nun ein Anfang einer Videokonferenz unter Nutzung des DSI zur effektiven Kanalisierung der Mediendaten. Die Entwicklung ist nicht perfekt und bislang nicht auf die parallele Verarbeitung mehrerer Ströme getestet. Im Konzept habe ich aber eine entsprechende Nutzung so weit wie möglich berücksichtigt.

Wie bereits in der Leistungsbewertung beschrieben, bedarf der Adaptionsalgorithmus einer Überarbeitung. Dabei ist zu beachten, daß der Einsatz von Codecs wie H.261 eine Veränderung der Stromeigenschaften mit sich bringt: Es ist zu erwarten, daß ein kodierter Strom eine zeitlich veränderliche Bandbreite besitzt.

6.3. Videokonferenz

Im Kapitel Entwurf habe ich eine mögliche Implementierung einer Videokonferenz nach dem H.323-Standard auf L4 beschrieben. Es ist zuerst ein ASN.1-Parser für das Projekt DROPS anwendbar zu machen, um die Protokolle, die zur Kommunikation zwischen H.323-Endpunkten notwendig sind, nutzen zu können.

Ich habe beschrieben, wie die Endpunkte eines solchen Systems aufgebaut werden können. Auf der Basis dieser Beschreibung können in Zukunft die Aufgaben an Studenten oder Mitarbeiter verteilt werden.

7. Zusammenfassung

In dieser Arbeit wurde der Videokonferenzstandard H.323 daraufhin analysiert, wie er in einem auf L4-basierten System implementiert werden kann. Aus dem Studium der verschiedenen Quellen enstand eine zusammenfassende Beschreibung der Merkmale eines H.323-Systems.

Aus der Kenntnis des Standards und dem Wissen über die vorhandenen Anwendungen für L4 entstand ein grundlegendes Konzept für die schrittweise Umsetzung.

Das *DROPS Streaming Interface* (DSI) soll in Zukunft in die Entwicklung von Medienanwendungen auf L4 (z. B. eine Videokonferenz) einfließen und die Kanalisierung der Datenströme übernehmen. Daher wurde in dieser Arbeit eine Grundkomponente einer Videokonferenz mit Nutzung des DSI entwickelt und getestet: Ein Videostrom wird zwischen zwei Rechnern übertragen. Diese Implementierung demonstriert die grundlegenden Mechanismen des DSI, die durch Meßreihen und deren Bewertung veranschaulicht werden.

A. Glossar

ARQ, ACF, ARJ Admission Request, Confirmation, Reject

ASN Abstract Syntax Notation

ATM Asynchronous Transfer Mode

BRQ, BCF, BRJ Bandwidth Change Request, Confirmation, Reject

B-ISDN Broadband-ISDN

CIF Common Intermediate Formate

DTMF Dual Tone Multi-Frequency

GK Gatekeeper

GRQ, GCF, GRJ Gatekeeper Request, Confirmation, Reject

GSTN General Switched Telefone Network

GW Gateway

IRQ, IRR Information Request, Request Resonse

IP Internet Protocol

ISDN Integrated Service Digital Network

ITU-T International Telecomunications Union — Telecomunications Standardisation Sector

LAN Local Area Network

LRQ, LCF, LRJ Location Request, Confirmation, Reject

LCN Logical Channel Number

MC Multipoint Controller

MCS Multipoint Communications System

MCU Multipoint Control Unit

MP Multipoint Processor

N-ISDN Narrow Band-ISDN

RAS Registration, Admission, Status

RRQ, RCF, RRJ Registration Request, Confirmation, Reject

PSTN Public Switched Telephone Network

RTP Real Time Protocol

RTCP Real Time Control Protocol

SCN Switched Circuit Network

TCP Transport Control Protocol

TSAP Transport Layer Service Access Point

URQ, UCF, URJ Unregistration Request, Confirmation, Reject

UDP User Datagram Protocol

Literaturverzeichnis

[LIE96] Jochen Liedtke
L4 Reference Manual
GMD/ IBM Watson Technical Report
http://os.inf.tu-dresden.de/L4/l4refx86.ps.gz

[OS96] The OSKit — Framework for Building Operating Systems
Flux Research Group, University of Utah
http://www.cs.utah.edu/flux/oskit/

[HOH] Michael Hohmuth
Using the OSKit as a base for L4 applications
Auf Anfrage erhältlich: hohmuth@innocent.com

[BOR98] Martin Borriss
Operating System Support for Predictable High-Speed Communication
Shaker Verlag, Aachen
ISBN 3-8265-6827-3

[DAN99] Uwe Dannowski
An ATM Driver for DROPS
Großer Beleg: http://os.inf.tu-dresden.de/~ud3/beleg/
ATM Firmware for DROPS
Diplomarbeit: http://os.inf.tu-dresden.de/~ud3/diplom/

[RLG00] Lars Reuther, Jork Löser, Lukas Grützmacher
DSI — DROPS Streaming Interface
http://os.inf.tu-dresden.de/drops/
TU Dresden, 2000

[DBC98] A Primer on the H.323 Series Standard
Version 2.0, (C)1997,1998 DataBeam Corporation
http://www.packetizer.com/iptel/h323/primer/

[H323] Draft Recommendation H.323:
Packet-based multimedia communication systems
International Telecomunication Union (ITU-T)
Draft v4 (05/2000)

[H245] Recommendation H.245:
Control Protocol for Multimedia Communication
International Telecomunication Union (ITU-T)
June 3, 1999

[H225] Draft Recommendation H.225.0
Call signalling protocols and media stream packetization for packet-based multimedia communication systems
International Telecomunication Union (ITU-T)
Draft v4 (02/2000)

[WEB-1] Liste der aktuellen Versionen frei verfügbarer Dokumente in Bezug zu H.323
http://www.packetizer.com/iptel/h323/

[IETF96] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson
RTP: A Transport Protocol for Realtime Applications
RFC 1889, Internet Engeneering Task Force, 1996
http://www.cis.ohio-state.edu/htbin/rfc/rfc1889.html

[WEB-2] RTP-FAQ
http://www.cs.columbia.edu/~hgs/rtp/faq.html

[PAU98] Torsten Paul
Entkopplung von Echtzeit- und Timesharing-Aktivitäten am Beispiel einer echtzeitfähigen Darstellungskomponente in DROPS
Diplomarbeit

[REU98] Lars Reuther
Entwicklung eines echtzeitfähigen Dateisystems
Diplomarbeit: http://os.inf.tu-dresden.de/papers_ps/real-time-fs.ps

[RUD00] Marco Rudolph
Eine reservierungsfähige Version von IP über Ethernet für DROPS
Diplomarbeit

[WEB-3] H.323 Tutorial
http://www.webproforum.com/h323/index.html

[WEB-4] H.323 and Associated Protocols
ftp://ftp.netlab.ohio-state.edu/pub/jain/courses/cis788-99/h323/index.html

Index