Entwurfsbegleitende Leistungsanalyse für SDL-basiertes Design multimedialer Internet-Transportsysteme

Dissertation
zur Erlangung des Grades

Doktor der Naturwissenschaften (Dr. rer. nat.)

Vorgelegt beim
Fachbereich Mathematik und Informatik
der Universität GH Essen
von

Jörg Hintelmann, geboren in Dortmund

Datum der mündlichen Prüfung: 4. Juli 2000
Gutachter:

Prof. Dr. Bruno Müller-Clostermann
(Universität GH Essen)

Prof. Dr. Michael Goedicke
(Universität GH Essen)

Vorwort

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter in der Arbeitsgruppe Systemmodellierung im Fachbereich Mathematik und Informatik der Universität GH Essen. Sie wäre ohne die Unterstützung einer Reihe von Personen in der vorliegenden Form nicht möglich gewesen.

Mein größter Dank gilt Herrn Prof. Dr. Bruno Müller-Clostermann, der nicht nur die Anregung zum Thema dieser Arbeit gab, sondern auch ein Arbeitsklima geschaffen hat, in dem es Spaß machte, diese Arbeit zu erstellen. Die Diskussionen mit ihm und die Anregungen durch ihn haben mir immer wieder über größere und kleinere Hürden hinweggeholfen.

Herrn Prof. Dr. Michael Goedicke danke ich nicht nur für die spontane Zusage, das Zweitgutachten zu übernehmen, sondern auch für die Anregung, die QUEST-Methode unter dem Blickwinkel des Software-Engineerings zu betrachten.

Mein besonderer Dank gilt Herrn Dr. Marc Diefenbruch für die kooperative Zusammenarbeit bei der Entwicklung des QSDL-Ansatzes. Weiterhin danke ich ihm und dem gesamten QUEST-Team, Axel Hirche, Wolfgang Textor, Jörg Rühl und Christian Rodemeyer, für die Implementierung des Werkzeugs, ohne das diese Arbeit nicht existieren würde.

Weiterhin gilt mein besonderer Dank meiner Freundin Corinna, die nicht nur durch ihre kritischen Nachfragen zur Präzisierung vieler Sachverhalte beigetragen hat, sondern mir auch vor allem in der Endphase den Rücken frei gehalten hat.

An dieser Stelle möchte ich mich auch bei meinen Eltern bedanken, die mir die Ausbildung ermöglicht haben, die zur Erstellung dieser Arbeit notwendig war.

Kurzfassung

Neben der funktionalen Korrektheit von komplexen Kommunikationssystemen spielt eine ausreichende Performance dieser Systeme eine immer wichtigere Rolle. Dabei ist es notwendig, Performance-Aspekte bereits in frühen Entwurfsphasen und nicht erst nach der Implementierung der Systeme zu berücksichtigen.

Die vorliegende Arbeit präsentiert eine Methodik zur entwurfsbegleitenden, modellgestützten Leistungsanalyse von Kommunikationsprotokollen, die mit Hilfe der Sprache SDL formal spezifiziert wurden. Die vorgestellte Methodik bedient sich dabei Beschreibungsmechanismen, Verfahren und Werkzeugen, die bereits im Entwurfsprozeß dieser Systeme verwendet werden.

Für die wichtigsten Implementierungsansätze von Kommunikationsprotokollen werden Performance-Bausteine vorgestellt und deren Verwendung erläutert. Die Tragfähigkeit der entwickelten Methodik wird durch eine große Fallstudie im Kontext von Reservierungsprotokollen zur Unterstützung von Multimedia-Anwendungen im Internet demonstriert. Dabei werden nicht nur die informellen Angaben der vorhandenen RFCs in formale SDL-Beschreibungen umgesetzt, sondern auch als existent vorausgesetzte Ressource-Management-Funktionen entwickelt und analysiert. Die vorliegende Arbeit schließt so die bisher bestehende Lücke im SDL-basierten Entwurfsprozeß verteilter reaktiver Systeme, indem sie die modellgestützte Betrachtung von Performance-Aspekten in den frühen Phasen des Entwurfsprozesses ermöglicht.

Inhaltsverzeichnis

Kapitel 1

Einführung

1.1 Multimedia-Systeme

Moderne Kommunikationssysteme werden sich einer Vielzahl neuer Anforderungen durch Multimedia-Anwendungen zu stellen haben. Neue verteilte Anwendungen wie Videokonferenzen, multimediale Post, Video-on-demand oder die Verbreitung großer Graphiken führen zu immensen Herausforderungen, die beim Entwurf neuer Kommunikationssysteme gelöst werden müssen. Die Übertragung digitaler Audio- und Videodaten erfordert die Bereitstellung von Verarbeitungsfunktionen in allen beteiligten Knoten, die eine Weiterleitung und Verarbeitung der Daten unter strengen Realzeitanforderungen ermöglichen. Die neu in ein System zu integrierenden Funktionen sollten dabei die bestehenden Anwendungen insoweit unterstützen, daß diese zumindest mit der bisherigen Dienstgüte erbracht werden können. Das führt im Gegensatz zur gegenwärtigen Situation dazu, daß neue Kommunikationssysteme Anwendungen mit sehr unterschiedlichen Anforderungen an die benötigte Dienstgüte (Quality of Service, QoS) unterstützen müssen.

Neue digitale Netzwerktechnologien wie Asynchroner Transfer Modus (ATM) [51], [77] können als erste Ansätze betrachtet werden, die Probleme multimedialer Kommunikation zu lösen. Allerdings ist nicht zu erwarten, daß es in naher Zukunft ein homogenes Multimedianetz geben wird. Im Gegenteil, die Liberalisierung des Telekommunikationsmarktes wird dazu führen, daß eine Menge parallel existierender Teilsysteme zu einem multimedialen Internet verschmelzen. Daher werden neue Realzeit-Netzwerk- u. Transportprotokolle zu entwickeln sein, die den Anforderungen der neuen Anwendungen Rechnung tragen. Im Bereich des Internets lassen sich drei große Richtungen ausmachen, diese Probleme anzugehen. Auf der einen Seite existieren Ansätze, die für adaptive Anwendungen geeignet sind, und die lediglich eine Beobachtung der erzielten Dienstgüte ermöglichen [97]. Das andere Extrem stellen Reservierungsprotokolle dar. Sie ermöglichen es, Teile der Netzwerk-Ressourcen für eine Verbindung zu reservieren, um so die geforderte Dienstgüte zu erlangen [20],[33],[111]. Als Mittelweg haben sich sog. *differentiated Services* in den Mittelpunkt der Diskussion gestellt [68]. Dabei werden QoS-Garantien nicht flußbasiert für eine Ende-zu-Ende-Sitzung gegeben, sondern über alle Flüsse innerhalb einer Domäne aggregiert.

1.2 Multimedia-Daten und ihre QoS-Anforderungen

Multimediale Daten, seien es Text, Hypertext, Graphiken oder digitale Audio- und Videodaten erfordern eine große Verarbeitungskapazität bzw. Speicherkapazität in den beteiligten Komponenten. Allerdings besteht bei den drei erstgenannten kein prinzipieller Unterschied zu klassischen Computerdaten.

Bei digitalen Audio- und Videodaten kommt neben dem immensen Datenaufkommen ihre Charakteristik als streng periodischer Datenstrom hinzu [105]. Das bedeutet, daß diese Daten mit derselben Geschwindigkeit abgespielt werden müssen, mit der sie von der Quelle eingespeist werden, obwohl sie i.d.R. unregelmäßig bei der Senke ankommen (siehe Abbildung 1-1).

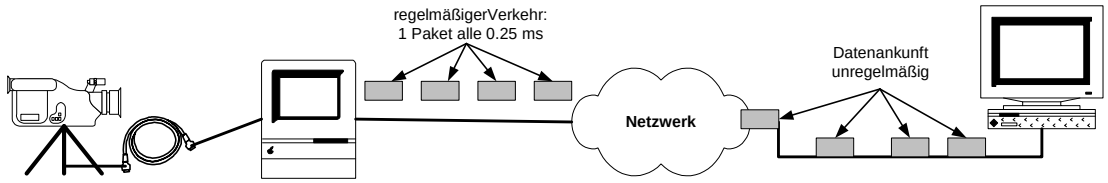


Abbildung 1-1 **Realzeitverkehr vor und nach Netzdurchlauf**

Im Falle digitaler Videodaten erfordern die heute gängigen Kodierungsverfahren Bandbreiten von mehreren hundert Mbit/s zur Übertragung von Filmen mit etwa 25 Bildern/s und selbst mit Kompressionstechniken werden noch Datenraten im Bereich von 2 Mbit/s erzielt [33]. Diese Datenraten liegen um Größenordnungen höher als die bisheriger Anwendungen. So reichen wenige Audio- oder Videoströme aus, um zum Beispiel ein lokales Netz (LAN) völlig auszulasten. Der sorgsame Umgang mit Systemressourcen ist daher in Zukunft sehr wichtig, selbst wenn neue, leistungsfähige Komponenten entwickelt werden.

Im Vergleich zu klassischen Computerdaten benötigen Bild- und Tondaten keine vollständig fehlerfreie Übertragung aller Datenpakete, da sowohl das menschliche Auge als auch das menschliche Gehör nur über ein begrenztes Auflösungsvermögen verfügen, bzw. Qualitätsverluste je nach Anwendung durchaus tolerieren. Im Falle eines Bildtelefongespräches kann man zum Beispiel bei zeitweiligen Engpässen nur jedes zweite Bild übertragen, oder falls erforderlich auch ein Standbild einblenden und lediglich den Sprachkanal aufrecht erhalten.

In der Literatur finden sich diverse Klassifikationen bzgl. der Dienstgüteanforderungen von Multimedia-Anwendungen. Die klassische Unterteilung sieht *asynchrone*, *synchrone* bzw. *isochrone* Ströme vor [105]. In [33] werden multimediale Datenströme als *non-realtime*, *realtime, non-interactive* bzw. als *realtime, interactive* bezeichnet. Zitterbart [116] unterteilt in 4 Klassen:

- *unzuverlässiger Realzeitdienst,*
- *zuverlässiger Realzeitdienst,*
- *zuverlässiger Dienst ohne Realzeitgarantien*
- *unzuverlässiger Dienst ohne Realzeitgarantien*

Dabei ist zu beachten, daß die Zuordnung der Anwendungen zu den einzelnen Klassen durch die Art und Weise ihrer Implementierung festgelegt wird [33]. Eine Anwendung, bei der ein Video-Client einen Film zunächst vollständig vom Server lädt, bevor er mit dem Abspielen beginnt, ist keine Anwendung mit Realzeitanforderungen. Wird diese Anwendung so implementiert, daß der

Client nach einer gewissen Vorlaufzeit mit dem Abspielen des Filmes beginnt, während weitere Daten geladen werden, so handelt es sich um eine Anwendung mit Realzeitanforderungen.

Eine weitere wichtige Anforderung von multimedialen Anwendungen besteht darin, ihre benötigten Dienstgüten adäquat an den entsprechenden Dienstschnittstellen zu formulieren, diese zu verhandeln und sie während der Verbindung zu beobachten und gegebenenfalls neu zuverhandeln. Dabei sollten die Parameter der Dienstgüte in der Semantik der Anwendung verhandelt werden und von sog. QoS-Mappern auf entsprechende Transportsystem, bzw. Netzwerkparameter abgebildet werden. Das erfordert zumindest neue Dienstschnittstellen, wenn nicht neue Kommunikationsarchitekturen [116].

1.3 Multimedia-Transportsysteme

Multimediale Ströme erfordern den Entwurf neuer Transportsysteme, die deren erweiterte Anforderungen erfüllen können. Heutige Transportsysteme sind aus vielfältigen Gründen nicht in der Lage, multimediale Ströme mit Realzeitanforderungen in ausreichender Qualität zu übertragen. Ein Problem stellt die fehlende Leistungsfähigkeit heutiger Computernetze dar. Durch den Einsatz glasfaserbasierter Übertragungstechnologien, durch den Entwurf ATM-basierter Netze wie DQDB, B-ISDN oder lokalen ATM-Netzen und durch die Verwendung von Kompressionstechniken werden die Anforderungen der Anwendungen sowie die potentielle Leistungsfähigkeit der Netze einander angenähert. Dies ist jedoch nur möglich, falls die Anwendungen eine Reduktion von Quantität bzw. Qualität tolerieren. Den Engpaß in diesem Gesamtsystem stellen die Transportsysteme, bestehend aus Transport- und Netzwerkprotokollen und deren Hardware dar.

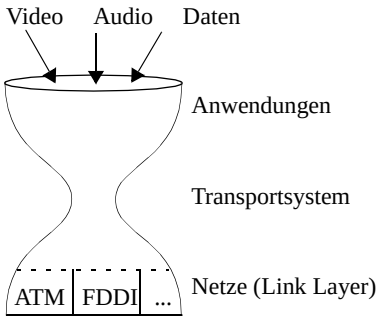


Abbildung 1-2 **Flaschenhals Transportsystem modifiziert nach [55]**

Insbesondere reicht die *Best-Effort-Semantik* bezüglich QoS und die FIFO-Bedienstrategie in allen beteiligten Subsystemen nicht aus, um die vorhandenen Ressourcen effizient zu nutzen. Weiterhin besteht keine Zugangskontrolle zu vorhandenen Ressourcen. Jede Anwendung belegt solange Ressourcen, wie freie Kapazitäten vorhanden sind. Darüberhinaus trägt der demokratische Ansatz bei der Verteilung von Ressourcen wie z.B. CPU-Zeiten den unterschiedlichen zeitlichen Bedürfnissen multimedialer Anwendungen keine Rechnung.

Zur effizienten Nutzung aller vorhandenen Ressourcen ist ein sog. *Ressource-Management* notwendig. Die Ressourcen zerfallen in zwei Klassen:

- sog. passive Ressourcen: Sie nehmen keine Datenverarbeitung vor und benötigen daher auch keine Scheduling-Strategien. Beispiele solcher Ressourcen sind Speicher.
- sog. aktive Ressourcen: Sie verarbeiten Daten und ihre Scheduling-Strategien haben wesentlichen Einfluß auf die Gesamtverzögerung einer Verbindung.

Passive Ressourcen erfordern kein aufwendiges Management, d.h. eine Reservierung für eine Verbindung ist ausreichend. Für aktive Ressourcen werden intelligentere Managementfunktionen benötigt. Diese Funktionen beinhalten eine Zugangskontrolle, eine QoS-Berechnung, gegebenfalls eine Reservierung der Ressourcen und zur Laufzeit die Erzwingung der ausgehandelten Dienstgüten durch geeignete Scheduling-Strategien.

1.4 Entwurfstechniken für Multimedia-Protokolle

Die Komplexität von Multimedia-Protokollen erfordert den Einsatz moderner Entwurfstechniken und Werkzeuge, die es ermöglichen, für entworfene Komponenten und Systeme zu überprüfen, ob sie den Anforderungen gewachsen sind. Diese Überprüfung sollte entwurfsbegleitend sein und in möglichst frühen Phasen des Entwicklungsprozesses einsetzen. In der Telekommunikationsindustrie haben sich zur Anforderungsanalyse und Spezifikation die Sprachen *Message Sequence Charts* (MSC) [73] und die *Specification and Description Language* (SDL) [72] durchgesetzt.

Eine Vielzahl von Werkzeugen unterstützt den Entwurfsprozeß durch Editoren und Analysatoren, die eine Überprüfung funktionaler Eigenschaften durch Erreichbarkeitsanalyse ermöglichen [98], [112]. Weiterhin ist es möglich, zu prüfen, ob ein Protokoll den Zieldienst erbringt, falls dieser in Form von MSCs spezifiziert ist. Die Werkzeuge ermöglichen darüberhinaus die Ableitung von Testfällen und eine Unterstützung der Implementierung durch Code-Generatoren. Da im Kontext multimedialer Kommunikation die funktionale Korrektheit eng mit der Leistungsfähigkeit der Systeme verknüpft ist, lassen sich die bisher verwendeten Methoden und Werkzeuge nur noch sehr begrenzt einsetzen, bzw. sind die mit diesen Werkzeugen erzielbaren Aussagen zur abschließenden Bewertung solcher Systeme nicht ausreichend.

Beim Entwurf multimedialer Transportsysteme spielt neben der funktionalen Korrektheit die Leistungsfähigkeit der Systeme eine entscheidende Rolle, da die korrekte Funktion des Systems u.a. an die Einhaltung zeitlicher Schranken geknüpft ist. Der klassische Weg mit den Zwischenschritten *Problemidentifikation*, *Anforderungsanalyse*, *Konzept Design*, *Detail Design*, *Implementierung*, *Testen der Funktion und der Leistung* und dann gegebenfalls *Redesign* führt dazu, daß Leistungsaspekte erst (zu) spät berücksichtigt werden. Es entstehen hohe Kosten durch Re-Design, und möglicherweise wird durch die Einführung von *Bypässen* zur Leistungssteigerung die Systemarchitektur zerstört.

Es sollte daher das Ziel sein, Leistungsaspekte so früh wie möglich zu berücksichtigen, um die aufgezeigten Probleme zu umgehen und so Zeit und Kosten zu sparen, vgl. Abbildung 1-3.

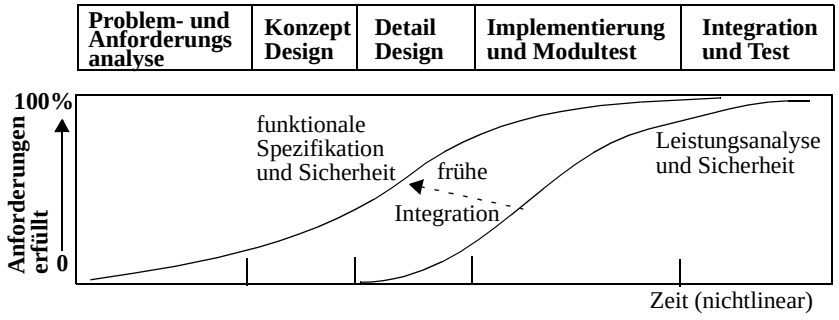


Abbildung 1-3 **Leistungsanalyse im Entwicklungsprozeß nach [79]**

Frühe Integration bedeutet modellgestützte Leistungsbewertung, die parallel zum Fortschreiten der Systementwicklung durchgeführt wird. Daher ist die isolierte Entwicklung und Pflege eigenständiger, quantitativ bewertbarer Modelle zu vermeiden, da die Vergangenheit gezeigt hat, daß dies in der Praxis aus Aufwandsgründen nicht durchgeführt wird. Zurecht findet man daher in der Literatur den Begriff der *insularity of performance evaluation* [43].

Außerdem erscheint der Wert der mit separater Performance Modellierung prognostizierbaren Leistungsmaße insbesondere deswegen zweifelhaft, weil die zu Zwecken der Leistungsanalyse eingesetzten Formalismen, Modelle und Analysetechniken mit den gängigen Software-Entwurfstechniken nicht verträglich sind. Insbesondere erlauben es die derzeit verfügbaren Techniken und Werkzeuge zur Leistungsanalyse nicht, eine formale Systemspezifikation als Grundlage (oder Bestandteil) eines quantitativ bewertbaren Modells zu integrieren.

Unter diesen Gegebenheiten ist es praktisch nicht möglich, einem sich in Entwicklung befindlichen Systementwurf ein konsistentes Leistungsanalysemodell zur Seite zu stellen. Aufgrund der wachsenden Komplexität moderner Rechen- und Kommunikationssysteme und der immer kürzeren Entwicklungszyklen ist es erfolgversprechender, Techniken zur modellgestützten Leistungsanalyse mit anderen entwurfsbegleitenden Methoden, insbesondere mit formalen Spezifikationstechniken zu integrieren, d.h. existierende formale Systementwürfe sind um Informationen zu erweitern, die für die Leistungsanalyse relevant sind, vgl. [17],[18] und [53]. Zugleich erlaubt ein Modell, das auf der Basis einer formalen Spezifikation entstanden ist, bei Bedarf eine sehr detaillierte Analyse von Performance-Problemen.

Der normalerweise mit der Erstellung eines solch detaillierten Modells verbundene Aufwand, der zurecht in klassischen separaten Modellierungsstudien gescheut wird, entfällt in diesem Fall, da die bereits existierende Spezifikation des funktionalen Verhaltens wesentliche Teile des Modells enthält. Damit verliert die Hauptkritik an der Verwendung detaillierter Leistungsmodelle an Wirkung.

1.5 Ziel und Inhalt der Arbeit

Der Ausgangspunkt für die vorliegende Arbeit ist die Beobachtung, daß entwurfsbegleitende Performance-Analysen von verteilten Systemen heute entweder gar nicht oder erst in sehr späten Entwurfsphasen stattfinden. Das liegt unter anderem daran, daß die Methoden, Werkzeuge und Vorgehensweisen zum Systementwurf sich fundamental von den Methoden und Werkzeugen zur Performance-Analyse unterscheiden. In der Praxis wird daher zu Recht der zusätzliche Aufwand zur entwurfsbegleitenden Erstellung und Pflege von Performance-Modellen gescheut.

Das erste Ziel der Arbeit besteht daher in der Ergänzung der SDL-Methode, die sich als gängige Methodik zum Entwurf von reaktiven, verteilten Systemen herauskristallisiert hat, um Sprachelemente, die die Spezifikation der leistungsrelevanten Informationen überhaupt ermöglichen. Das Resultat dieses ersten Schrittes ist die Sprache QSDL und das Werkzeug QUEST, siehe Kapitel 3.

Mit der Spracherweiterung und der Implementierung des gewählten Ansatzes sind die notwendigen **Vorarbeiten** für die Durchführung einer entwurfsbegleitenden Performance-Analyse geschaffen. Darüberhinaus gilt es, eine Methodik zu erarbeiten und beispielhaft zu demonstrieren, wie in der Praxis verwendete Implementierungsansätze für Kommunikationsprotokolle in Leistungsmodelle überführt werden können und wie mit Hilfe des Werkzeugs QUEST die unterschiedlichen Analysen ermöglicht werden. Diese Methodik und die Analyseformen werden in Kapitel 4 vorgestellt.

Um die Leistungsfähigkeit eines Transportsystems bewerten zu können, müssen Angaben über die Belastung durch die Anwendungen und Angaben über die unterliegenden *Link Layer* Protokolle vorliegen, da diese von den Instanzen des Transportsystems benutzt werden, siehe Abbildung 1-4.

Dabei ist es nicht ausreichend, wie bei einer funktionalen Analyse, das logische Verhalten der unterliegenden Schichten zu kennen. Vielmehr sind detaillierte Kenntnisse über das **zeitliche** Verhalten unterliegender Schichten notwendig. Damit der Vorteil eines homogenen Modells zur funktionalen und zur quantitativen Bewertung, der durch die QUEST-Methodik erreicht werden soll, nicht verloren geht, ist es notwendig, für wichtige Schicht-2-Protokolle Performance-Bausteine zu entwickeln und deren zeitliches Verhalten zu analysieren. Weiterhin müssen auch für die wichtigsten Multimedia-Anwendungen Lastbausteine entwickelt werden, um ein durchgängiges Modell für Last und System zu erhalten. Das Lastverhalten typischer Multimedia-Anwendungen sowohl für kontinuierliche Ströme wie Sprache, Audio und Video als auch für diskrete Ströme wie FTP, Telnet oder WWW wird mit Hilfe der Sprachelemente von QSDL in Kapitel 5 beschrieben.

Die Schwierigkeiten, die bei der Beschreibung der Schicht-2-Protokolle auftreten, und deren Lösung wird in Kapitel 6 beschrieben.

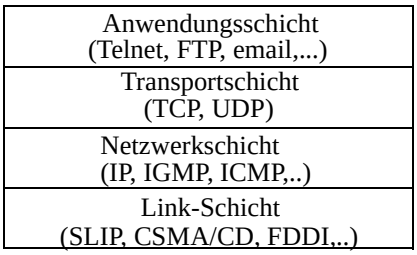


Abbildung 1-4 **Hierarchie der TCP/IP Protokoll-Suite**

Für die Unterstützung von Multimedia-Strömen im Internet liegen Dienstspezifikationen und teilweise Umsetzungs-Frameworks in Form von RFCs[a] vor. Diese RFCs sind informelle Beschreibungen der zu unterstützenden Dienste und machen Vorschläge zu einer möglichen Implementierung der notwendigen Protokollfunktionen. Die publizierten Dokumente sind insofern unvollständig, als daß wichtige Funktionen zur Unterstützung von Realzeitströmen in den Knoten nicht festgelegt sind, deren Existenz aber vorausgesetzt wird. Eine Übersicht über die vorhandenen Protokolle zur Unterstützung von Multimedia-Strömen wird in Kapitel 7 gegeben.

Ein weiterer Schwerpunkt dieser Arbeit liegt im Entwurf von Ressource-Management-Funktionen im Kontext von Reservierungsprotokollen sowie der Umsetzung der informellen Angaben aus den RFCs in formale SDL-Spezifikationen, um eine Bewertung des resultierenden Gesamtsystems zu ermöglichen. In Kapitel 8 werden der Entwurf und die Analyse lokaler Ressource-Manager beschrieben und in Kapitel 9 wird die Tragfähigkeit der entwickelten Methode an einer größeren Fallstudie zum Reservierungsprotokoll $ST2^+$ demonstriert.
Kapitel 10 faßt die wichtigsten Ergebnisse zusammen und zeigt auf, in welchem Umfang die gesetzten Ziele erreicht werden konnten. In Kapitel 11 wird ein Ausblick auf mögliche Ergänzungen und Fortführungen der Arbeit gegeben.

a. Request for Comment sind offizielle Standards der Internet Community. Einige sind “offizielle” Standards, viele werden nur zu informellen Zwecken veröffentlicht.

Kapitel 2 SDL

2.1 Einleitung

Der korrekte Entwurf komplexer, verteilter Systeme, wie z.B. Kommunikationsprotokolle, wird durch Verwendung formaler Beschreibungstechniken (Formal Description Techniques FDTs) erleichtert. Die dem Formalismus zugrundeliegende Semantik ermöglicht es, das Verhalten der neu zu entwickelnden Systeme eindeutig und unmißverständlich zu beschreiben. Weiterhin kann das zugrundeliegende formale Modell dazu benutzt werden, Entwurfsfehler bereits in frühen Entwicklungsphasen zu entdecken und zu beseitigen.

Eine weit verbreitete formale Beschreibungstechnik stellt SDL [72] dar. Die erste offizielle Version wurde 1976 verabschiedet. 1988 wurde der Sprache eine formale Basis gegeben, so daß ab diesem Zeitpunkt SDL als formale Beschreibungstechnik angesehen werden kann. Letzte wesentliche Ergänzungen stellen die objekt-orientierten Erweiterungen in SDL'92 dar. In SDL'96 sind kleinere Änderungen an der Sprache vorgenommen worden. SDL ist in der Telekommunikationsindustrie weit verbreitet. Anwendungsgebiete sind beispielsweise ISDN-Systeme [92], Komponenten von Intelligenten Netzen [7],[29], Flugnavigationssysteme [50] oder Kontrollsysteme für Industriemaschinen [81]. Neben der weiten Industrieverbreitung hat sich SDL in den Arbeitsgruppen der *International Telecommunication Union* (ITU) sowie im *European Telecommunications Standards Institute* (ETSI) als bevorzugte Standardisierungssprache durchgesetzt [40].

Die wichtigsten Elemente werden hier kurz zusammengefaßt. Eine ausführliche Einführung über SDL findet man in der Literatur [15],[40]. Weiterhin wird dargelegt, welche Informationen in einem SDL-System fehlen, um eine Leistungsanalyse durchzuführen und welche Ansätze existieren, diese Lücke zu schließen.

2.2 Sprachelemente von SDL

SDL stellt Konzepte und Sprachmittel zu Verfügung, die es erlauben, die Struktur und das Verhalten von Telekommunikationssystemen, bzw. allgemeiner formuliert, das Verhalten von reaktiven Systemen als Stimuli/Response-Zusammenspiel zu beschreiben. Dabei sind sowohl Stimuli

als auch Response diskrete Ereignisse (Signale), die Träger von Informationen sind. Als Konzept zur Beschreibung sehr großer und komplexer Systeme wird die Partitionierung, bzw. Dekomposition unterstützt, die es ermöglicht, eine System-Hierarchie festzulegen. So ist es möglich, ein System *top down* zu entwerfen und durch Verfeinerungen die einzelnen Komponenten zunehmend detaillierter zu beschreiben.

Die handelnden Einheiten in einem SDL-System sind die Prozesse. Das Verhalten von SDL-Prozessen wird durch asynchron kommunizierende, erweiterte, endliche Automaten beschrieben. Alle in einem System existierenden Prozeßinstanzen laufen autonom und nebenläufig ab.

Die asynchrone Kommunikation geschieht durch Signalaustausch zwischen beteiligten Prozessen. Die ausgetauschten Signale können in den (unbegrenzten) Eingangspuffern der empfangenden Prozesse gespeichert werden, bis diese in der Lage sind, sie weiterzuverarbeiten, vgl. Abbildung 2-1.

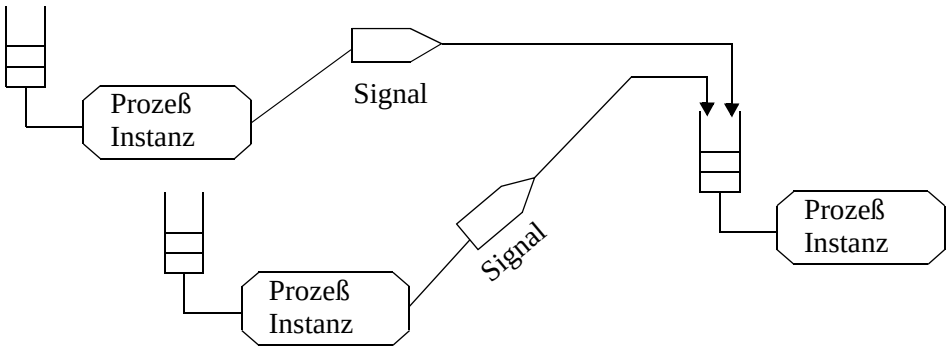


Abbildung 2-1 **Prozeßkommunikation in SDL mod. nach [15]**

Jede Prozeßinstanz besitzt einen Eingangspuffer, aus dem Signale standardmäßig in der Reihenfolge entnommen werden, in der sie eintreffen. Durch Verwendung des Save-Konstrukts kann diese FIFO-Reihenfolge ebenso verändert werden, wie durch die Verwendung von Priority-Input, siehe Abbildung 2-2. Durch Verwendung von Priority-Input können Signale gegenüber solchen bevorzugt behandelt werden, die durch das normale SDL-Input konsumiert werden. Die Erstellung einer mehrstufigen Prioritätshierarchie, wie sie aus der Welt der Warteschlangen bekannt ist, erlaubt SDL nicht.

Weiterhin stellt SDL abstrakte, algebraische Datentypen zur Verfügung, die äquivalent den Konzepten der ACT ONE [39] und LOTOS-Datentypen [70] sind. Es wird eine Menge von Operatoren und Sorten definiert. Die Wirkung der Operatoren wird durch algebraische Gleichungen festgelegt, d.h. es wird das Verhalten der Operatoren spezifiziert, und jede Implementierung ist korrekt, die alle Gleichungen erfüllt.

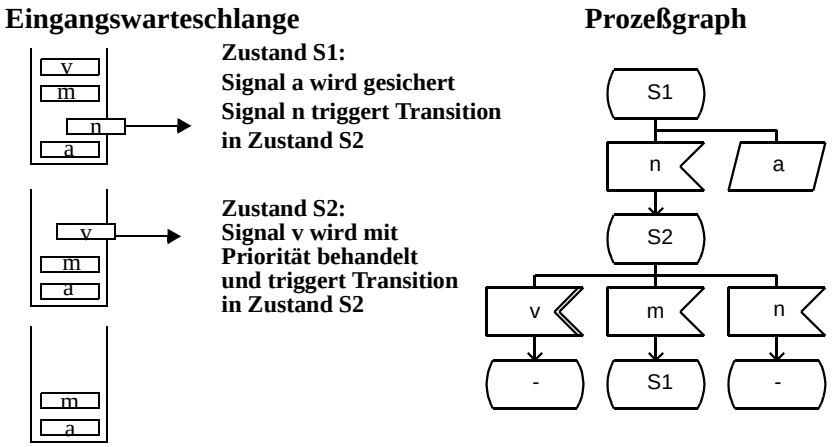


Abbildung 2-2 **Mechanismen der Eingangswarteschlange nach [40]**

2.3 SDL und Leistungsbewertung

SDL-Spezifikationen legen die Struktur eines Systems sowie das funktionale Verhalten der beteiligten Komponenten fest. Diese Beschreibung sollte so allgemein sein, daß keinerlei Einschränkungen bzgl. der Implementierung vorliegen [40].

Das dynamische Verhalten eines SDL-Systems kann durch die Erzeugung des globalen Zustandsraumes beschrieben werden. Dieser Zustandsraum kann in Form eines gerichteten Graphen dargestellt werden, wobei die Knoten den Systemzustand enthalten und als Kantenbeschriftung K_{ij} die Transition gewählt wird, die das System von Zustand *i* in den Zustand *j* überführt. Jeder Weg ausgehend vom globalen Startzustand durch den Zustandsraum stellt einen möglichen Ablauf des beschriebenen Systems dar.

Mit diesem Ansatz lassen sich prinzipiell Sicherheits- und Lebendigkeitseigenschaften wie Deadlocks durch Zustandsraumexploration untersuchen. In der Literatur sind jedoch Beispiele veröffentlicht worden, die gezeigt haben, daß diese Eigenschaften bei der Einführung von zeitlichen Restriktionen verloren gehen können [10],[26],[38].

Aussagen über die Anzahl der übertragenen Signale von A nach B, über Verbindungsaufbauzeiten oder über die Anzahl abgelehnter Verbindungswünsche können allein auf der Grundlage einer SDL-Spezifikation nicht gemacht werden. Dazu fehlen in einer solchen Spezifikation Angaben über vorhandene Systemressourcen wie Prozessoren, Speicher, I/O-Kanäle oder Netzwerkkarten. Allgemeiner formuliert ist es nicht vorgesehen, in SDL ein Maschinenmodell zu formulieren, um die Leistung als Reaktion eines Systems auf eine übergebene Last zu bestimmen.

Weiterhin ist es nicht vorgesehen, eine für die Leistungsbewertung notwendige quantitative Lastbeschreibung zu formulieren. Lediglich mögliche Stimuli und ihre Parametertypen sind beschrieben. Wie häufig und in welchen Abständen diese Stimuli auftreten und welche konkreten Ausprägungen die Parameter bei den einzelnen Auftreten haben, läßt sich in SDL nicht ausreichend beschreiben. Insbesondere lassen sich die typischen Lastmuster digitaler Sprache oder MPEG-codierter Videoströme nicht als SDL-Signalsequenz bzw. MSC beschreiben.

Ein weiteres Problem ergibt sich bei der Frage, ob die SDL-Spezifikation als reine Lastbeschreibung anzusehen ist, oder ob sie auch Angaben zum Maschinenmodell enthält. So ist es zum Bei-

spiel möglich, die Struktur des SDL-Systems als Indiz für das unterliegende Architekturmodell anzusehen. Allerdings besteht das Problem, daß in SDL eine vielstufige Hierarchie von Blöcken und Sub-Blöcken möglich ist. Man müßte daher willkürlich eine Ebene als sog. Referenzebene festlegen, die als Beschreibung der Maschinenarchitektur angesehen werden soll.

Weiterhin besteht das Problem, daß sich eine Zerlegung in der Spezifikation an funktionalen Teilsystemen orientiert und in der Regel nicht an physikalischen Aufteilungen. Außerdem ist eine Spezifikation so allgemein, daß aus derselben Spezifikation unterschiedliche Realisierungen des Systems erstellt werden können, die funktional verhaltensäquivalent sind, sich aber fundamental in der Implementierungsarchitektur und somit in ihrem Leistungsverhalten unterscheiden. Für eine aussagefähige, flexible Leistungsbewertung sollte es möglich sein, die verschiedenen Realisierungen im Leistungsbewertungsmodell adäquat zu formulieren.

2.4 Lösungsansätze in der Literatur

Die Notwendigkeit, Leistungsanalyse mit formalen Beschreibungstechniken zu kombinieren, findet man seit Mitte der 80er Jahre in der Literatur. Erste Ansätze dazu befinden sich in den Arbeiten von Rudin [96], Hogrefe, Zorn [63] und Conway [28]. Eine Übersicht, die diverse Ansätze vorstellt, findet man in [53]. Nachfolgend werden die bekanntesten Ansätze dargestellt, die eine SDL-basierte Leistungsbewertung durchführen.

2.4.1 SDL-Simulationsmodell nach Wohlin

Der Autor stellt in [114] eine Methode vor, die ausgehend von einer SDL-Beschreibung der Anwendungssoftware sowie Modellen der Hardware-Architektur und Anforderungsmodellen der Systemumgebung einen Performance-Simulator erstellt, mit dessen Hilfe dann das Gesamtsystem analysiert werden kann. Die Transformation der SDL-Beschreibung in den Performance-Simulator wird durch spezielle Transformatoren unterstützt. Der Ansatz erfordert aber sowohl die Beschreibung des Architekturmodelles als auch die der Anforderungsmodelle als SDL-basiertes Simulationsmodell. Dies erscheint als Mißbrauch der Sprache SDL, da diese zur Beschreibung von verteilten Systemen entwickelt wurde ist und nicht als Simulationssprache. Positiv ist zu bewerten, daß versucht wurde, die formale Spezifikation der Anwendungssoftware als wesentlichen Bestandteil in das Leistungsmodell zu integrieren.

2.4.2 Timed-SDL

In diesem Ansatz werden Zeiten mit der Ausführung von Transitionen assoziiert [11]. Die Schaltdauern von SDL-Transitionen werden als exponential verteilt oder zeitlos angenommen. Falls mehrere zeitlose Transitionen in Zuständen möglich sind, müssen Schaltwahrscheinlichkeiten angegeben werden. Das so erweiterte TSDL-System läßt sich in eine Semi-Markov-Kette transformieren. Mit den bekannten numerischen Analysetechniken können die stationären Zustandswahrscheinlichkeiten und aus diesen weitere stationäre Leistungsmaße abgeleitet werden. Eine transiente Analyse von TSDL-Systemen ist ebenfalls auf der Grundlage der Semi-Markov-Ketten möglich. Der gewählte Ansatz ist auf Markov-Prozesse beschränkt, und die Grenzen der Anwendungsmöglichkeit werden durch die Beherrschung der Zustandsraumexplosion gesteckt. Der Ansatz ermöglicht es ebenfalls nicht, das zugrundeliegende Maschinenmodell adäquat zu beschreiben. Somit können auch Phänomene, die beim konkurrierenden Zugriff auf beschränkte Systemressourcen auftreten, nicht detailliert untersucht werden, bzw. die Auswirkungen müssen in die Angaben der Transitionsdauern und -wahrscheinlichkeiten eingerechnet

werden. Daher scheint dieser Ansatz nicht geeignet zu sein, Leistungsbewertung in den Entwurfsprozeß zu integrieren, da eine *von-Hand-Transformation* der SDL-Spezifikation in das Leistungsbewertungsmodell durchgeführt werden muß.

2.4.3 SDL-Net

Bei SDL-Net handelt es sich um eine Unterklasse von *Queueing Petri-Nets* (QPN), die wiederum *Coloured Generalized Stochastic Petri Nets* (CGSPN) mit erweiterten Stellen darstellen [11],[12].

Der gewählte Ansatz ermöglicht die Transformation einer SDL-Spezifikation in ein SDL-Netz. Diese Transformation ist durch den Systementwickler durchzuführen. Er kann exponentiell verteilte Raten für Transitionen, Kanalverzögerungen sowie für Timer-Werte angeben. Besonders wird die Modellierung von SDL-Prozeß-Warteschlangen, des SDL-Save-Mechanismus, des Timeout-Mechanismus sowie die FIFO-Kanal-Warteschlange durch entsprechende SDL-Netze vorgestellt. Die Leistungsanalyse kann entweder numerisch durch eine stationäre Analyse der eingebetteten Markov-Kette oder durch Simulation durchgeführt werden.

Die Beschränkungen des vorgestellten Ansatzes liegen zum einen darin, daß die dem SDL-System zugrundeliegende Systemarchitektur und insbesondere das Konzept geteilter Ressourcen nicht berücksichtigt wird. Weiterhin verhindert die Beschränkung auf die Exponentialverteilung, die eine Markov-Ketten-Analyse erlaubt, den Einsatz des SDL-Net-Ansatzes im Entwurfsprozeß realer Kommunikationsprotokolle. Eine weitere Schwäche besteht in der starken Einschränkung der verwendbaren Datentypen, die durch die verwendete Modellwelt der Petri Netze begründet ist. Insgesamt erscheint daher der SDL-Net-Ansatz nicht geeignet, in der Praxis die modellgestützte Leistungsanalyse in den Systementwurfsprozeß zu integrieren.

2.4.4 SPECS

Die Grundidee von SPECS besteht darin, mit der Ausführung von SDL-Aktionen Zeitdauern zu assoziieren [24]. Typische Aktionen sind Input und Output von Signalen, Zuweisungen in Tasks, Setzen und Rücksetzen von Timern, et cetera. Weiterhin wird die SDL-Zeitsemantik dahingehend geändert, daß der Signaltransfer über Kanäle eine vom Entwickler zu spezifizierende Zeit dauert, die Anzahl der von einem Prozeß ausführbaren Aktionen pro Zeit endlich ist, und daß jeder Prozeß seine eigene, von anderen Prozessen unabhängige Ausführungsgeschwindigkeit hat.

In SPECS wird ein einfaches Maschinenmodell eingeführt. Jedem Systemblock wird eine virtuelle Maschine zugeordnet, die mit einer spezifischen Geschwindigkeit die Aktionen aller im Block aktiven Prozeßinstanzen ausführt. Jedem Prozeß i in einem Block j wird ein Gewicht w_{ij} zugeordnet. Diese Gewichte legen den Anteil an Prozessorkapazität q_{ij} fest, der einem Prozeß zugeteilt wird.

Prozesse innerhalb eines Blocks laufen quasi parallel im *multitasking*-Modus, wie er von realen CPUs bekannt ist, während Prozesse in unterschiedlichen Blöcken echt parallel ausgeführt werden können. Die um Gewichte und Maschinen ergänzte Spezifikation kann nun simuliert werden. Während des Simulationslaufes werden Traces erzeugt, aus denen typische Leistungsmaße wie Kanaldurchsätze, Wartezeiten oder maximale und minimale Warteschlangenlängen der SDL-Prozesse berechnet werden können.

Im Gegensatz zu den beiden vorherigen Ansätzen bietet SPECS die Möglichkeit, ein Maschinenmodell anzugeben und somit Phänomene, wie sie beim konkurrierenden Zugriff auf Maschinen

auftreten, zu analysieren. Allerdings ist der Ansatz, jedem Block eine Maschine zuzuordnen, nicht flexibel genug, um reale Systeme zu bewerten. Außerdem fehlen Möglichkeiten, zwischen den Prozessen unterschiedliche Prioritätsklassen zu bilden, um so den unterschiedlichen Anforderungen von Multimedia-Strömen Rechnung zu tragen.

2.4.5 SPEET

SPEET ist ein Werkzeug zur Leistungsbewertung von Systemen, die in SDL spezifiziert sind. Die SDL-Spezifikation wird durch Code-Generierung in C-Code transformiert und kann dann ausgeführt werden, vgl. Abbildung 2-3.

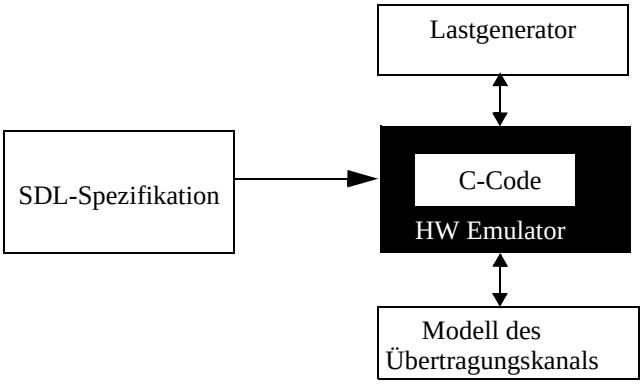


Abbildung 2-3 **SPEET-Komponenten mod. nach [17]**

Die verwendete Hardware wird in diesem Ansatz durch Emulatoren ersetzt. SPEET ermöglicht eine Lastbeschreibung durch Verkehrslastgeneratoren, die Verkehrsströme von Anwendungen wie digitalisierte Sprache, WWW, FTP, SMTP, Telnet oder MPEG-codierte Videoströme ermöglicht. Die einzelnen Ströme können zu beliebigen Lastmixturen gemischt werden und das zu emulierende System belasten [106]. Weiterhin enthält SPEET Kanalmodelle, mit deren Hilfe die Verbindung verschiedener SDL-Systeme in die Untersuchung mit einbezogen werden kann. Die gewünschten Leistungsgrößen können als *Probes* in die SDL-Spezifikation eingebracht werden. Dazu wird ein neues *Probesymbol* eingeführt, das als Annotation innerhalb des SDL-Kommentarsymbols erscheint. Weiterhin ist es möglich, quantitative Anforderungen an das System in Form von erweiterten MSCs auszudrücken. Die Verletzung solcher Anforderungen kann durch die Simulation bzw. die Emulation aufgedeckt werden.

Die wesentlichen Stärken des SPEET-Ansatzes liegen in der Existenz sehr detaillierter Lastmodelle vor allem aus dem Bereich Mobilkommunikation. Darüberhinaus existieren sehr genaue Repräsentationen existierender Hardware-Strukturen, z.B. Mikro-Controler der Firmen Intel, Motorola und Siemens. SPEET ermöglicht daher sehr detaillierte Analysen des neuen Systems, erfordert aber sehr aufwendige Hardware-Emulatoren und Kanalmodelle. Die Lastgeneratoren ermöglichen die Erstellung sehr realistischer Verkehrsströme. Die vorgestellte Methode erfordert zudem die vollständige Ausspezifizierung des gesamten Systems, und kann daher erst relativ spät in den Entwurfsprozeß integriert werden.

2.4.6 SDL-HIT-Integration

Der Ansatz von Heck [53] stellt ein Konzept zur Anbindung von formalen Spezifikationen an Leistungsmodelle vor. Die Spezifikation basiert auf kommunizierenden, erweiterten, endlichen Automaten, während die Leistungsmodelle auf Warte-Bedienstationen beruhen. Damit verfolgt dieser Ansatz am konsequentesten den Weg der Aufteilung in Last und Maschine. Der allgemeine Ansatz, der auf Arbeiten von Heck, Hogrefe und Müller-Clostermann beruht [54] wird im Kontext von SDL und HIT [14] konkretisiert.

Ausgangspunkt ist eine SDL-Spezifikation, in die Angaben über zeitverbrauchende *Service Calls* ergänzt werden, vgl. Abbildung 2-4. Dies geschieht als Ergänzung in den Task-Konstrukten in SDL, wobei Zeitverbräuche durch Signalaustausch und durch Datenmanipulation allgemeiner Art berücksichtigt werden.

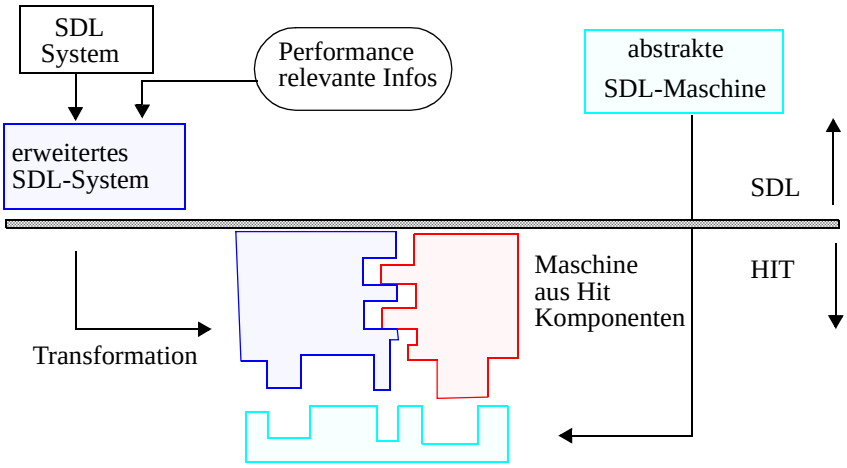


Abbildung 2-4 **SDL/HIT-Modell mod. nach [53]**

Daraus ergibt sich eine erweiterte SDL-Spezifikation, die als Lastkomponente in das HIT-Modell übernommen wird. Dazu ist es notwendig, für jedes SDL-Konstrukt eine Repräsentation in HI-Slang, der Beschreibungssprache für HIT-Modelle, anzugeben.

Parallel dazu werden Maschinen, die die *Service Calls* “abarbeiten” aus zeitverbrauchenden HIT-Komponenten gebildet. Ebenfalls als HIT-Komponente wird eine abstrakte SDL-Maschine formuliert, zu deren Aufgaben u.a. die Auflösung impliziter Variablen oder die Zuweisung systemweit eindeutiger Prozeßidentifikatoren (PIDs) gehört. Aus diesen drei Teilen wird ein vollständiges HIT-Modell erzeugt und durch simulative Modellausführung die gewünschten Leistungsmaße gewonnen.

Der gewählte Ansatz ermöglicht modellgestützte Leistungsbewertung neu zu entwerfender Systeme, wobei die formale Spezifikation als wesentlicher Bestandteil der Lastbeschreibung verwendet wird. Die Erstellung eines vollständigen quantitativ bewertbaren Modells erfordert jedoch zusätzlich zur Identifikation von Stellen im System, an denen *Service Calls* eingefügt werden müssen, detaillierte Kenntnisse der HIT-Welt. Außerdem müssen die verwendeten Maschinen als HIT-Komponenten modelliert werden. Daher erscheint die Integrierbarkeit der Methode in den Entwurfsprozeß zumindest fragwürdig.

2.5 Zusammenfassung Kapitel 2

SDL als eine der etabliertesten Spezifikationssprachen für verteilte reaktive Systeme wurde mit ihren grundlegenden Konzepten vorgestellt. Darüberhinaus sind die fehlenden Möglichkeiten zur Beschreibung nichtfunktionalen Verhaltens herausgestellt worden. Die damit verbundenen Probleme bei der Berücksichtigung von Performance-Aspekten im Entwurfsprozeß dieser Systeme ist verdeutlicht worden. Die Vorstellung und Würdigung existierender Ansätze, die Spezifikationssprache SDL mit Methoden zur Leistungsanalyse zu kombinieren, hat dieses Kapitel abgerundet.

Im nun folgenden Kapitel wird ein Ansatz vorgestellt, der gemeinsame Wurzeln mit der HIT-Integration aufweist. Auch hier sind ergänzende Informationen zur Lastbeschreibung in die SDL-Spezifikation einzufügen und Maschinen zu beschreiben, die die durch das SDL-System und die Umgebung auftretenden Lasten bewältigen müssen.

Kapitel 3

QSDL und QUEST

3.1 Konzepte

Queueing SDL (QSDL) ist eine Sprache, die Ergänzungen zur SDL-Beschreibung enthält, in der fehlende Informationen zu Maschinen, Lasten und deren Bindung enthalten sind. Die Konzepte der Sprache beruhen auf ersten Arbeiten von Heck, Hogrefe und Müller-Clostermann [52],[54], die an der Universität Gesamthochschule Essen im Kontext von SDL weiterentwickelt wurden [35],[36].

QSDL ist im Rahmen dieser Arbeit und der Arbeit von Marc Diefenbruch [38] entwickelt worden und unterstützt die meisten Konstrukte von SDL-88 und einige der Erweiterungen, die in SDL-92 definiert worden sind. Nicht unterstützt werden zur Zeit *Services*, *Packages*, *Channel Partitioning*, *viewed Variables*, *Imported Values*, *Makros*, *Remote Procedures* sowie die *objektorientierten Erweiterungen*. Ein QSDL-System beinhaltet die vollständige SDL-Spezifikation, die das funktionale Verhalten bestimmt, siehe Abbildung 3-1. Eine vollständige Übersicht über die Sprache QSDL findet man in [36].

QUEST ist ein Werkzeugsystem, das an der Universität Gesamthochschule Essen im Rahmen der Arbeiten [38], [61], [93] und [110] entwickelt wurde. Mit QUEST können QSDL-Systeme analysiert werden. Dazu wird aus einer QSDL-Spezifikation durch Code-Generierung einen Evaluator erzeugt, der drei verschiedene Analysen erlaubt.

QUEST ermöglicht eine Validation unter Berücksichtigung zeitlicher und räumlicher Beschränkungen durch Zustandsraumexploration, sowie eine Verifikation des zeitbehafteten Systems durch Model-Checking. Das Haupteinsatzgebiet von QUEST ist die Durchführung von simulativen Leistungsbewertungen. Details zu QUEST sowie zu den Analysetechniken zur Validation und Verifikation findet man in [37] und [38].

Der QUEST-Ansatz weist Gemeinsamkeiten mit dem in SPECS verfolgten Ansatz auf, ist aber allgemeiner. Im Gegensatz zu SDL-Net oder der SDL/HIT-Integration wird nicht versucht, die erweiterte SDL-Spezifikation auf ein Modell abzubilden, das mit Methoden und Werkzeugen zur Leistungsanalyse bewertet werden kann. Statt dessen wird die vollständige, erweiterte Spezifikation in ihrem zu erwartenden realen Verhalten simuliert. Durch Sensoren werden zur Ausfüh-

rungszeit des Simulationsmodells die gewünschten Leistungsmaße ermittelt und können zur Laufzeit visualisiert werden. Durch mehrfache Simulationsläufe können mit den bekannten Verfahren auch stationäre Leistungsmaße für eine Schar unterschiedlicher Parameter ermittelt werden [59], [60].

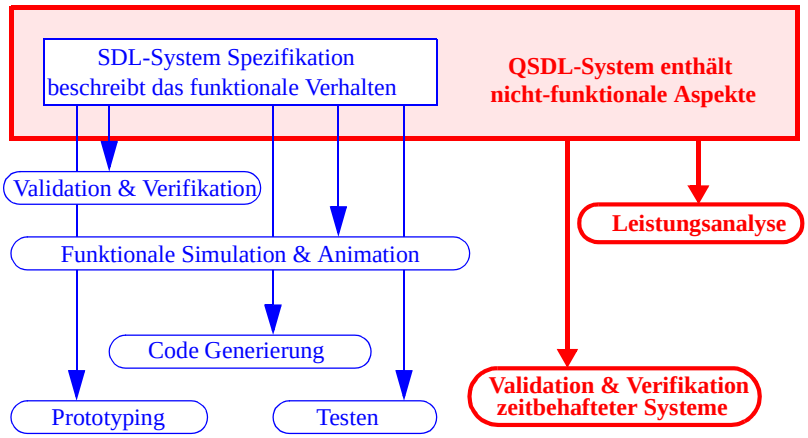


Abbildung 3-1 **Die Rolle von (Q)SDL-Systemen im Entwurfsprozeß**

Das grundlegende Konzept besteht darin, eine SDL-Spezifikation durch ein Maschinenmodell zu ergänzen, mit dessen Hilfe der Systemdesigner den für die Implementierung notwendigen Raum- und Zeitbedarf beschreiben kann. Weiterhin werden Sprachelemente eingeführt, mit deren Hilfe die zu bewältigende Arbeitslast durch geeignete Verkehrsmodelle repräsentiert werden können. Darüberhinaus stellt QSDL einen Bindungsmechanismus zur Verfügung, mit dessen Hilfe die Verteilung von Lasten auf die vorhandenen Maschinen beschrieben werden kann, siehe Abbildung 3-2.

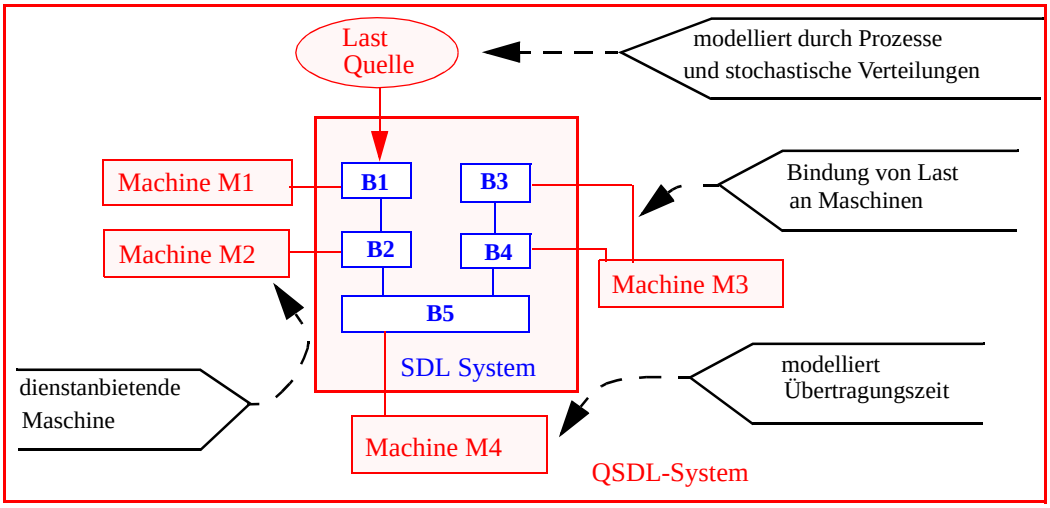


Abbildung 3-2 **QSDL-System**

Den Kern eines QSDL-Systems bildet die SDL-Spezifikation. Diesem System werden Modelle von Verkehrsquellen angelagert, die die Signale, die das SDL-System aus der Umgebung emp-

fangen kann, gemäß des für die Anwendung typischen Musters in das System schicken. Daneben werden die Ressourcen (Maschinen) definiert, die für das zu entwickelnde System bereitgestellt werden sollen. Diese modellieren die Zeitverbräuche die durch Warten auf Zuteilung, Verarbeitung und Signalübertragung entstehen.

Der Bindungsmechanismus ermöglicht eine flexible Verteilung von dienstanfordernden Prozessen auf die im System vorhandenen Maschinen. So können parallele Aktivitäten dargestellt werden. In Abbildung 3-2 haben die Instanzen B1 und B2 exklusiven Zugriff auf die ihnen zugeordneten Maschinen M1 bzw. M2 und können so unabhängig voneinander ihre Aufgaben erfüllen. Im Gegensatz dazu müssen sich die Instanzen B3 und B4 die Maschine M3 teilen und konkurrieren um diese Maschine. Physikalische Signalübertragungszeiten können ebenfalls modelliert werden, siehe Maschine M4. Im nun folgenden Abschnitt wird das QSDL zugrundeliegende Maschinenmodell erläutert.

3.2 QSDL-Maschinenmodell

Das Konzept der QSDL-Maschinen ist aus der Welt der Warteschlangensysteme entnommen, die ein sehr populäres und mächtiges Paradigma zur modellgestützten Leistungsbewertung darstellen, vgl. [49], [74], [115]. Warteschlangensysteme werden zur Beschreibung und Analyse von Phänomenen genutzt, die bei gleichzeitiger Benutzung von exklusiven Betriebsmitteln durch mehrere Nutzer entstehen. Je nach Struktur und Typ eines Warteschlangenmodells existieren analytische, numerische oder simulative Lösungstechniken zur Berechnung von Leistungsmaßen. Für einfache Warteschlangenstationen hat sich die Kendall-Notation zur Beschreibung etabliert. Eine solche Station wird durch A/B/m/K/Z beschrieben, wobei

- A und B den Ankunfts- bzw. den Bedienprozeß charakterisiert,
- m die Anzahl der identischen Bediener bezeichnet,
- K die Gesamtkapazität und
- Z die Scheduling-Disziplin beschreibt, d.h. nach welchem Schema wartende Kunden aus dem Warteraum entnommen werden.

Typische Scheduling-Strategien sind *First In First Out* (FIFO), *Processor Sharing* (PS) als Approximation von Zeitscheibenverfahren mit infinitesimal kleiner Zeitscheibe oder *Prioritätsverfahren*.

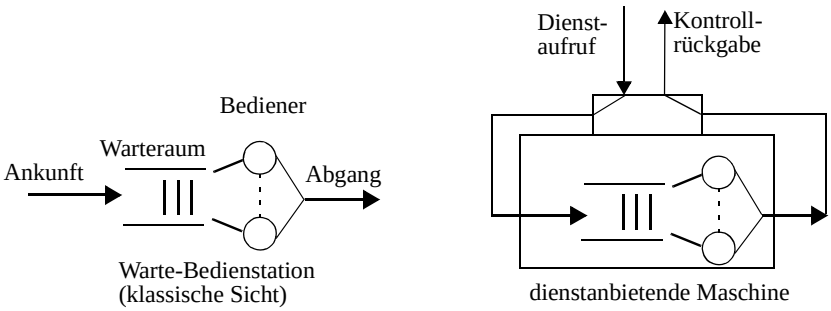


Abbildung 3-3 **Warte-Bedienstation und dienstanbietende Maschine**

Ein einfaches Warte- Bediensystem ist in Abbildung 3-3 links dargestellt. QSDL betrachtet solche Bedienwartesysteme als dienstanbietende Maschine, vgl. Abbildung 3-3 rechts.

Die angebotenen Dienste können von Kunden aus der Umgebung der Maschine über eine Schnittstelle angefordert werden. Der dienstaufrufende Kunde wird solange blockiert, bis der von ihm gewünschte Dienst vollständig erbracht ist. Die Gesamtzeit der Blockierung hängt vom Umfang des gewünschten Dienstes, der Geschwindigkeit der ausführenden Maschine und der Wartezeit des Kunden im Warteraum der Maschine ab. Diese Sichtweise entspricht modernen Konzepten des Systementwurfes und ist bereits länger in der Welt der Leistungsanalyse etabliert [14], [52], [53]. Das vorliegende Maschinenmodell dient lediglich dazu, Zeitverbräuche durch Aktionen und Signalübermittlung, sowie entstehende Wartezeiten durch konkurrierende Nutzung der Maschinen nachzubilden.

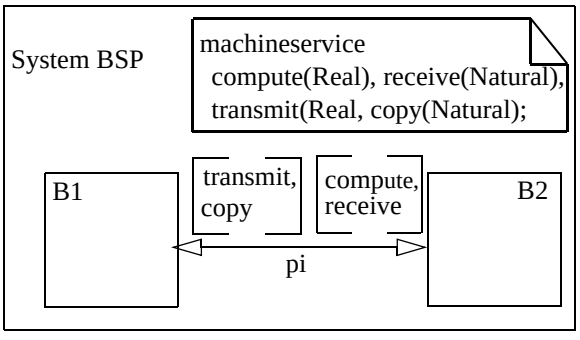
QSDL-Maschinen können nicht wie zum Beispiel in HIT hierarchisch aufgebaut sein und haben keinerlei Funktionalitäten, die über die Modellierung von Zeitverbräuchen bei der Dienstabwicklung hinausgehen.

In den nun folgenden Abschnitten werden alle in QSDL definierten Erweiterungen syntaktisch und semantisch vorgestellt und an Beispielen illustriert. Bei der Einführung der QSDL-Konstrukte wurde besonderes Augenmerk darauf gelegt, diese syntaktisch und strukturell an SDL anzulehnen.

3.3 Ergänzungen auf Systemebene

Auf Systemebene sind in QSDL nur Struktur- und Kommunikationsinformationen festlegbar. Analog zu SDL-Signalen können QSDL-Maschinendienste festgelegt werden. Diese Dienste werden von den QSDL-Maschinen angeboten, siehe Abschnitt 3.2, und können von den QSDL-Prozessen per request angefordert werden, vgl. Abschnitt 3.5. Die Definition wird durch das Schlüsselwort machineservice eingeleitet, dem eine Aufzählung aller Maschinendienste folgt, die systemweit bekannt sein sollen. Als zweites Element auf Systemebene können *pipes* definiert werden. Sie sind das Analogon zu SDL-Kanälen und können Dienstaufrufe zwischen Blöcken transportieren. Das ist immer dann nötig, wenn die dienstausführende Maschine und der aufrufende Prozeß in unterschiedlichen Blöcken spezifiziert sind.

Beispiel 3 -1 **Ergänzungen auf Systemebene (QSDL/GR[a] und QSDL/PR)**




```
System BSP;
  /*## machineservice
     compute(Real),receive(Natural),
     transmit(Real), copy(Natural); ##*/
  block B1 referenced;
  block B2 referenced,
  /*## pipe pi
    from B1 to B2 with compute,receive;
    from B2 to B1 with transmit,copy;
  endpipe; ##*/
endsystem;
```

a. Das Werkzeug QUEST arbeitet ausschließlich mit textueller Sprache, die ursprünglich als echte Spracherweiterung definiert worden ist. Um die Kompatibilität mit Standard-SDL-Werkzeugen zu gewährleisten, werden die QSDL-Ergänzungen ab Version 1.3 in SDL-Kommentare, gefolgt von ## gekapselt.

In Beispiel 3 -1 sind systemweit die Maschinendienste *compute*, *receive*, *transmit* und *copy* definiert. Zwischen den beiden SDL-Blöcken ist eine pipe namens *pi* definiert, die *compute* und *receive* von B1 an B2 übertragen kann und *transmit* und *copy* in die andere Richtung. Daher könnte zum Beispiel im Block B2 eine Maschine definiert werden, die die Dienste *compute* und *receive* anbietet, vgl. Abschnitt 3.4. Diese Dienste könnten von Prozessen in B1 angefordert werden.

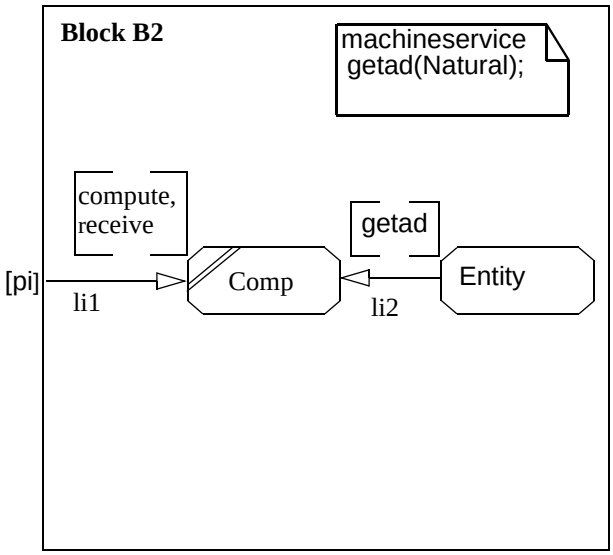
3.4 Ergänzungen auf Blockebene

Auf der Blockebene können wie auf der Systemebene weitere Maschinendienste deklariert werden. Diese sind dann nur innerhalb des Blocks bekannt. Sollten hier Dienste mit Namen definiert werden, die bereits auf Systemebene verwendet wurden, so überschreibt die Blockdefinition die Systemdefinition.

Die Maschinen werden ebenfalls auf Blockebene definiert. Dies geschieht durch das Schlüsselwort machine. Zur vollständigen Spezifikation einer Maschine gehört die Angabe eines Bezeichners, die Anzahl der Bedieneinheiten, eine Angabe der Bedienstrategie sowie eine Menge von Maschinendiensten und dienstspezifischen Geschwindigkeiten.

Den Transport von Dienstaufrufen übernehmen auf Blockebene sogenannte *Links*. *Links* sind uni-direktionale Verbindungen, die in einer Maschine enden. Der Anfangspunkt eines link ist entweder ein Prozeß oder die Blockgrenze, die eine Schnittstelle zur Umgebung bildet. An der Blockgrenze kann der link durch bind to an eine pipe angeschlossen werden.

Beispiel 3 -2 **Ergänzungen auf Blockebene (QSDL/GR und QSDL/PR)**




```
block b2;
    process Entity referenced;
    /*## machineservice getad(Natural); ##*/
    /*## machine Comp;
          server 1;
          discipline fcfs;
          offers compute: 1.0, receive:2,
          getad:4;
          endmachine; ##*/
    /*## link li1 from ENV to Comp
          with compute, receive; ##*/
    /*## link li2 from Entity to Comp
          with getad; ##*/
    /*##  bind pi to li1; ##*/
endblock;
```

Das obige Beispiel zeigt die QSDL-Ergänzungen auf Blockebene exemplarisch anhand des Blocks B2 aus dem vorhergehenden Beispiel.

Zusätzlich zu den bereits auf Systemebene definierten Maschinendiensten wird hier noch ein weiterer Dienst mit Namen *getad* definiert. Dieser Dienst kann neben den beiden systemweit bekannten Diensten *compute* und *receive* von der Maschine *Comp* erbracht werden. Die Maschine besitzt einen Bediener, sie entnimmt die Kunden aus der Warteschlange gemäß der

Reihenfolge ihres Eintreffens (FCFS) und bietet die drei Dienste *compute*, *receive* und *getad* an. Die Dienste werden mit den unterschiedlichen Geschwindigkeiten *1.0*, 2^b und *4* angeboten. Man beachte, daß die Angaben der Maschinengeschwindigkeiten den gleichen Typ haben müssen, wie die Parameter der Maschinendienste.

Der Prozeß *Entity* kann den Maschinendienst *getad* über den link *li2* an die Maschine *Comp* absetzen, während der link *li1* das Environment des Blockes mit der Maschine *Comp* verbindet. Über die pipe *pi* können so Dienstaufrufe aus anderen Teilen des Systems an die Maschine *Comp* gelangen.

3.5 Ergänzungen auf Prozeßebene

Nachdem auf System- und Blockebene alle Maschinendienste, Maschinen und Transportwege definiert wurden, muß nun in den SDL-Prozessen festgelegt werden, welche Dienstaufrufe mit den einzelnen SDL-Aktionen verbunden sein sollen und welchen Umfang sowie gegebenfalls welche Priorität den Dienstaufrufen zugeordnet werden soll.

Dazu dient das QSDL-Konstrukt request, das überall dort in einen SDL-Prozeß eingefügt werden kann, wo auch das SDL-Task-Konstrukt stehen kann. Mit der Angabe von request wird der Name des gewünschten Maschinendienstes angegeben, der Umfang des Bedienwunsches, der dem Typ des Parameters in der machineservice-Deklaration entsprechen muß und optional eine Priorität des gewünschten Dienstes. Maschinen, die keine Prioritäten unterstützen, ignorieren die Prioritätsangabe.

Mit dem Aufruf des request können Adressinformationen verbunden werden, wenn die Zielmaschine nicht eindeutig identifizierbar ist. Das kann entweder durch Verwendung von via und Angabe eines sog. *Request-Pfades* oder durch Angabe des Maschinenbezeichners nach dem Schlüsselwort from realisiert werden. Sollte die Zielmaschine nicht eindeutig sein, wird zufällig eine der vorhandenen Maschinen ausgewählt, die den entsprechenden Dienst anbietet und zu der ein Pfad vom aufrufenden Prozeß aus existiert.

Damit ist ein sehr flexibles und mächtiges Instrument geschaffen worden, mit dessen Hilfe der Systementwickler an jeder Stelle, die ihm relevant erscheint, zeitverbrauchende Dienstaufrufe einfügen kann. Der aufrufende SDL-Prozeß wird solange blockiert, bis der Dienstwunsch von der Maschine bearbeitet wurde. In der Zwischenzeit eingehende Nachrichten werden in der Eingangsschlange des Prozesses gesichert. Die Dauer der Blockierung hängt vom Umfang des gewünschten Dienstes, der Maschinengeschwindigkeit, der Scheduling-Strategie der Maschine sowie der Belastung der Maschine durch andere Prozesse ab.

Alle anderen (Q)-SDL-Aktionen wie input, task oder decision werden als zeitlos angenommen.

Neben der Einführung von zeitverbrauchenden *Dienstaufrufen* ist in QSDL eine weitere Ergänzung vorgenommen worden, die das zeitliche Verhalten von SDL-Systemen beeinflußt. Sollen Systeme zu bestimmten Zeitpunkten aktiviert werden, um Aktionen durchzuführen, die harte Realzeitanforderungen haben, so wird dies in SDL durch Timer realisiert. Wenn der Timer abläuft, wird ein entsprechendes Signal in die Eingangswarteschlange des betreffenden Prozesses gestellt. Der Prozeß kann aber erst reagieren, wenn alle vor dem Timer-Signal wartenden Signale konsumiert und die mit dem Signal verbundenen Aktivitäten durchgeführt sind. Damit kann jedoch keine verläßliche Aussage gemacht werden, zu welchem Zeitpunkt der Prozeß tatsächlich reagieren wird. In QSDL können daher sog. zeitbehaftete Zustände definiert werden, aus denen

b. Konstante des Typs Natural

heraus SDL-Prozesse durch spontane Transitionen unabhängig von vorhandenen Signalen aktiviert werden können. In diesen zeitbehafteten Zuständen können Zeitpunkte festgelegt werden, zu denen spontane Transitionen schalten dürfen. Das geschieht durch Angabe eines Ausdrucks vom Typ duration hinter dem Schlüsselwort awake in der QSDL-Zustandsdefinition. Betritt ein Prozeß einen solchen Zustand zum Zeitpunkt t_0 und ist eine Zeitspanne von Δt definiert, so kann der Prozeß zum Zeitpunkt $t_1 = t_0 + \Delta t$ die nächste spontane Transition ausführen. Sind mehrere spontane Transitionen definiert, wird eine zufällig ausgewählt. Transitionen, die durch Signale ausgelöst werden, können jederzeit ausgeführt werden. Verläßt der Prozeß durch diese Transition den zeitbehafteten Zustand, so wird der awake-Zeitpunkt gelöscht. Bleibt der Prozeß in diesem Zustand, bleibt der bisherige Zeitpunkt gültig.

Abbildung 3-4 illustriert die Verwendung dieses neuen Konstrukts. Gegeben seien die Prozesse *Steuermodul*, *Kessel* und *Wasserbad*. Der Prozeß *Steuermodul* fordert zyklisch die aktuellen Temperaturen der beiden Komponenten an und falls der Prozeß *Kessel* nicht innerhalb von einer Zeiteinheit (ZE) reagiert, wird er durch das Signal *notaus* ausgeschaltet.

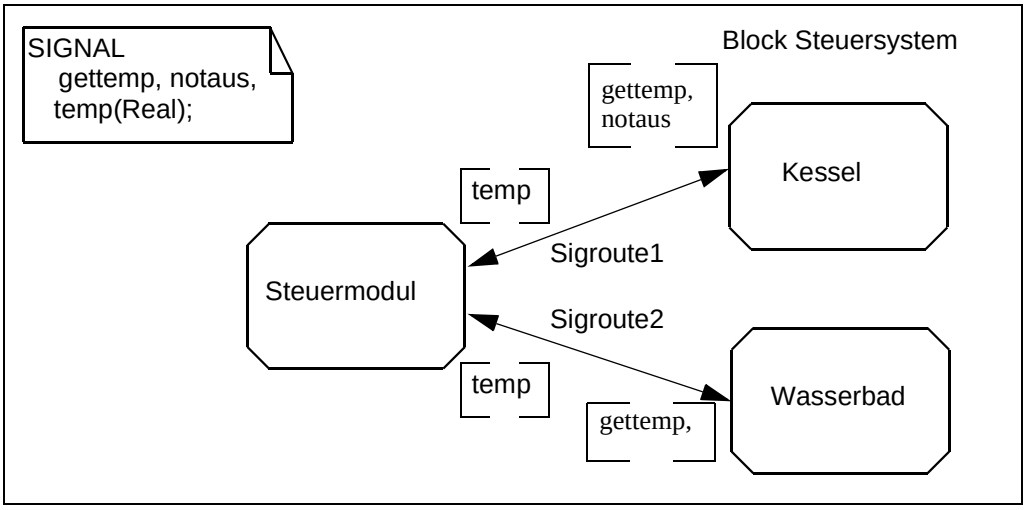


Abbildung 3-4 **Prozeßsteuerung mit awake (Blockebene)**

Der Prozeß *Steuermodul* fordert die aktuelle Temperatur des Prozesses *Kessel* per Signal *gettemp* an, wechselt in den Zustand *waittemp* und pausiert dann für eine Zeiteinheit (*awake 1.0*), siehe Abbildung 3-5. Falls er die Temperatur des Kessels vor Ablauf dieses Zeitpunktes erhält, wechselt er in den Zustand *ready*. Empfängt er nur die Temperatur des Wasserbades, so verbleibt er in dem Zustand. Der Zeitpunkt, zu dem die spontane Transition wieder aktiviert wird, bleibt unverändert.

Empfängt der Prozeß *Steuermodul* die aktuelle Temperatur des Kessels nicht rechtzeitig, kann die spontane Transition feuern und das Signal *notaus* wird gesendet. Rechtzeitig heißt, daß es höchstens eine ZE nach Absenden des Signals *gettemp* eintreffen muß. So läßt sich sicherstellen, daß das System **exakt** zum spezifizierten Zeitpunkt reagiert. Die Verwendung von SDL-Timern würde dazu führen, daß zu dem Zeitpunkt ein Timer-Signal in der Eingabewarteschlange erscheint, der Prozeß also **frühestens** zu diesem Zeitpunkt reagieren kann.

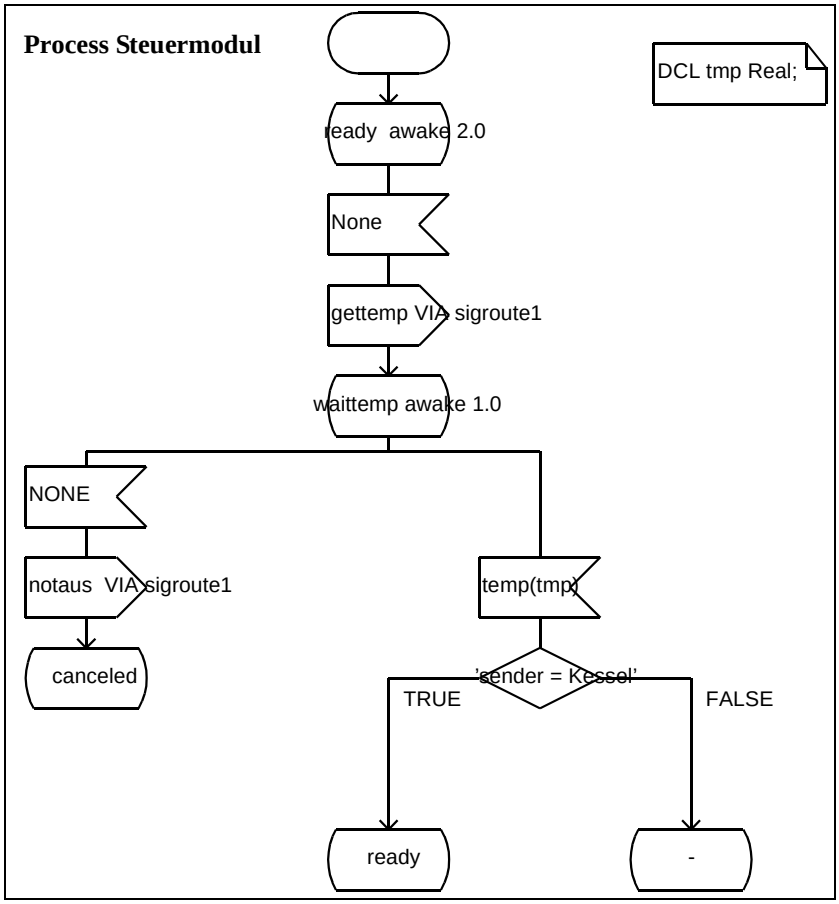


Abbildung 3-5 **Prozeßsteuerung mit awake (Prozeß Steuermodul)**

Eine weitere Ergänzung auf Prozeßebene, die das zeitliche Verhalten von QSDL-Systemen gegenüber SDL-Systemen ändert, betrifft das Output-Konstrukt. Zusätzlich zu den in SDL möglichen Angaben zu Signalparametern und Adress- bzw. Routing-Informationen kann jedem Output ein Wert für eine verzögerte Auslieferung des gesendeten Signals mitgegeben werden. Dadurch ist es möglich, reine Zeitverzögerungen, wie sie zum Beispiel durch physikalische Signallaufzeiten entstehen, auf einfache Weise zu modellieren.

Die Speicherkapazität moderner breitbandiger Übertragungsmedien kann so ebenfalls einfach nachgebildet werden. Das ist durch Modellierung mit Hilfe von Maschinendiensten und Dienstaufrufen durch die Prozesse nur mit zusätzlichem Aufwand möglich, vgl. Kapitel 6.

Die verzögerte Auslieferung von Signalen ist in SDL ebenfalls möglich, wenn man verzögernde Kanäle zwischen SDL-Blöcken verwendet. Im Gegensatz zu dieser nichtdeterministischen Kanalverzögerung, die die Sendereihenfolge von Signalen eines Prozesses an einen anderen erhält, erlaubt es das QSDL-Konstrukt, beliebige Ausdrücke vom Typ duration als Verzögerung anzugeben. Somit ist bei Verwendung von stochastischen Verzögerungswerten ein Überholen von Signalen in QSDL-Systemen möglich.

Als Ergänzung zu den bereits in SDL vordefinierten Datentypen und unter Verwendung von generischen Mechanismen der abstrakten Datentypen sind in QSDL spezielle Datentypen zur Beschreibung von Bedienwünschen, Ankunftsabständen und zur Auflösung von nichtdetermini-

stischen Entscheidungen eingeführt worden. Das folgende Beispiel zeigt die Definition des Datentyps Uniform inklusive der für alle stochastischen Datentypen implementierten Operatoren init und sample.

Beispiel 3 -3 **Datentyp Uniform (Definition und Anwendung in QSDL)**

```
newtype Uniform
    operators
        init: Uniform, Real, Real -> Uniform;
        sample: Uniform -> Real;
endnewtype Uniform;

DCL distance Uniform, arrival Real;
TASK distance := init(distance, 0.1,0.5);
TASK arrival := sample(distance);
```

In einem QSDL-Prozeß können Variablen vom Typ Uniform analog zu anderen Datentypen definiert werden. Durch Anwendung des init-Operators werden diese Variablen initialisiert. Dabei sind im Falle der Uniform-Verteilung die obere und untere Intervallgrenze anzugeben. Die wiederholte Anwendung des Operators sample liefert reelle gleichverteilte Zahlen aus dem Intervall [untere Grenze, obere Grenze]. Die folgende Tabelle gibt eine Übersicht über alle zusätzlich bereitgestellten Datentypen und ihrer Parameter.

Name	**Parameter**	**Bemerkungen**
Deterministic	Rückgabewert	konstanter Wert als Rückgabe
Empirical	Anzahl der Rückgabewerte (Integer), Name (Charstring)	Werte werden aus einer Datei gelesen
Erlang	Mittelwert (Real), Standardabweichung (Real)	Werte gemäß Erlang-Verteilung
Negexp	Mittelwert (Real)	exponential verteilte Zahlen mit Rate 1/Mittelwert
Normal	Mittelwert (Real), Standardabweichung (Real)	normalverteilte Zahlen
Uniform	untere Grenze, obere Grenze	gleichverteilte Zahlen
Bernoulli	Wahrscheinlichkeit für True	liefert wahr oder falsch
Poisson	Mittelwert (Integer)	poisson verteilte Zahlen
RandInt	untere Grenze, obere Grenze	ganzzahlig gleichvert. Zahlen

Tabelle 3 -1 **QSDL-Datentypen**

3.6 QSDL-Sensoren

Neben den Änderungen, die das zeitliche Verhalten von QSDL-Systemen im Vergleich zu SDL-Systemen beschreiben, sind in der Sprache QSDL Sensoren definiert, die es ermöglichen, das Systemverhalten unter Zeit- bzw. Performance-Aspekten zu beobachten.

Neben der Beobachtungsfunktion kann das Verhalten eines QSDL-Systems von aktuellen Werten der Sensoren abhängig gemacht werden. Damit wird den Systementwicklern die Möglichkeit gegeben, das System während des Simulationsvorganges in Abhängigkeit von der aktuellen

System-Performance zu steuern. So kann zum Beispiel die Verteilung von Lasten (*Requests*) auf eine Anzahl existierender Maschinen dadurch vorgenommen werden, daß die Warteschlangenlängen aller in Frage kommender Maschinen ermittelt werden. Die Maschine, deren Warteschlange die kürzeste ist, wird als Ziel des Auftrages ausgewählt.

Sensoren werden entweder vom Laufzeitsystem automatisch aktualisiert, sofern es sich um Sensoren der in QSDL vordefinierten Typen handelt, oder sie müssen explizit innerhalb der Spezifikation aktualisiert werden. Zu den vordefinierten Typen gehören die system- bzw. blockweiten Sensoren, die Maschinensensoren, sowie die prozeßspezifischen Sensoren, vgl. Tabelle 3 -2 bis Tabelle 3 -4.

Typ	Funktion
FreqSignal	Ermittelt die relativen Häufigkeiten von systemweit *übertragenen* Signalen.
FreqRequest	Ermittelt die relativen Häufigkeiten von systemweit *ausgeführten Requests.*
FreqProcessCreate	Berechnet die relativen Häufigkeiten von systemweiten *Prozeßerzeugungen.*
FreqProcessStop	Berechnet die relativen Häufigkeiten von systemweiten *Prozeßterminierungen*.
CountSignal	Zählt die Anzahl der systemweit *übertragenen* Signale.
CountRequest	Zählt die Anzahl der systemweit *angeforderten* Requests.
CountProcessCreate	Zählt die Anzahl der systemweit *erzeugten* Prozeßinstanzen.
CountProcessStop	Zählt die Anzahl der systemweit *terminierten* Prozeßinstanzen.
LifeTimeProcess	Ermittelt die Lebensdauern der Prozeßinstanzen.

Tabelle 3 -2 **System-/ Block-Sensoren in QSDL**

Die System- bzw. Blocksensoren bieten eine Übersicht über das Gesamtsystem bzw. über Blöcke, die als Subsysteme aufgefaßt werden können. Diese Sensoren können in zwei Kategorien aufgeteilt werden. Reine Zähler ermöglichen es, *Signale*, *Requests*, *Prozeßerzeugungen* und *Prozeßterminierungen* zu quantifizieren. Bei der Definition dieser Sensoren können optionale Parameterlisten festgelegt werden. Damit kann erreicht werden, daß nur spezifische Elemente aus dem Wertebereich der Listenelemente erfaßt werden. So kann z.B. einem Sensor vom Typ *CountSignal* eine Liste mit Signaltypen zugeordnet werden. Dann zählt dieser Sensor nur Signale, die in der Liste aufgeführt sind. Fehlt die Angabe einer solchen Liste, werden alle im (Sub-) System vorhandenen Signale gezählt.

Darüberhinaus existieren sog. Frequency-Sensoren, die die Erstellung von Häufigkeitsverteilungen ermöglichen. Diese Sensoren sind immer ohne Parameter zu spezifizieren, da sie relative Werte über alle Vorkommen erheben sollen. Mit ihrer Hilfe ist es möglich, in (Teil-) Systemen zu beobachten, welche Signale und Requests besonders häufig verwendet werden oder welche Prozeßinstanzen besonders häufig instanziiert werden. Auf deren effiziente Implementierung sollte verstärkt Wert gelegt werden, da sich ineffiziente Mechanismen hier besonders nachteilig auswirken.

Eine Sonderrolle kommt dem *LifeTimeProcess*-Sensor zu, der ebenfalls mit Parametern versehen werden kann. Dieser Sensor kann die Lebenszeit von Prozessen ermitteln. Damit sind z.B. Verbindungsdauern zu bestimmen, deren Abwicklung durch separate Prozesse abgewickelt wird. Das ist z.B. im *Transmission Control Protocol* (TCP) der Fall, wo für jede TCP-Verbindung eine eigene Prozeßinstanz erzeugt wird.

Zur Beobachtung des Performance-Verhaltens der QSDL-Maschinen sind ebenfalls eine Reihe von Sensoren definiert.

Typ	Funktion
FreqRequest	Ermittelt die relativen Häufigkeiten der *ausgeführten Requests* an einer Maschine.
QueueLength	Ermittelt die relativen Häufigkeiten der wartenden Requests im Warteraum der Maschine.
QueueContent	Liefert die Aufteilung der im Warteraum *wartenden* Requests.
Utilization	Ermittelt die Maschinenauslastung.
CountRequest	Zählt die Anzahl der von der Maschine *angeforderten* Requests.
WaitTimeRequest	Ermittelt die Wartezeit der Requests an der Maschine.
ServiceTimeRequest	Ermittelt die Bedienzeit der Requests an der Maschine.
Throughput	Ermittelt den Durchsatz von Requests an der Maschine.

Tabelle 3 -3 **Maschinenspezifische Sensoren**

Auch hier sind Sensoren mit und ohne Parametern definiert. Zu den parameterlosen Sensoren gehören die ersten vier Sensoren in Tabelle 3 -3. Sie ermöglichen eine Übersicht über die Maschinenauslastung, die Länge und den Inhalt des Warteraumes sowie über die Verteilung der Häufigkeiten der angeforderten Maschinendienste.

Durchsatz, Warte- und Bedienzeit sowie absolute Dienstanforderungen hingegen können diensttypspezifisch ermittelt werden. Hierzu ist den Sensoren bei der Definition eine Liste mit Diensttypen zuzuordnen. Wird dies unterlassen, so werden die Werte für alle angebotenen Dienstypen bestimmt. Die Maschinensensoren können dazu verwendet werden, solche Komponenten im System zu ermitteln, deren Kapazitäten nicht ausreichen, um die von den Anwendungen geforderten Dienstgüten zu erfüllen.

Die größte Anzahl unterschiedlicher Sensoren existiert für die QSDL-Prozesse, da sie die handelnden Einheiten darstellen und somit eine große Anzahl von Beobachtungsmöglichkeiten bieten. Für Prozesse sind sechs parameterlose Sensortypen definiert, die die Verteilung von empfangenen, gesendeten, gesicherten, verlorenen und konsumierten Signale ermittelt. Darüberhinaus sind zwei parameterlose Sensortypen definiert, die die Eingangswarteschlange von Prozessen bezüglich Inhalt und Füllungsgrad beobachten. Daneben existiert je ein Sensortyp für die erreichten Zustände des Prozesses sowie einer für seine initiierten Maschinendienste.

Für Prozesse sind eine Menge von Zählsensoren definiert, denen optional Signaltypen zugeordnet werden können. Damit können signalspezifische Erhebungen über Sende- und Empfangsereignisse, Signalverluste, -sicherungen und -konsumierungen erstellt werden. Eine Ermittlung der Signalübertragungszeiten ist ebenfalls signalspezifisch mit Hilfe des *TransmissionTimeSignal*-Sensors möglich.

Zur diensttypspezifischen Ermittlung der Anzahl ausgeführter Request-Anforderungen dient der *CountRequest*-Sensor. Der *ResponseTimeRequest*-Sensor kann dazu verwendet werden, die Zeit zu ermitteln, die eine Prozeßinstanz auf die vollständige Bearbeitung von Maschinendienstaufrufen wartet.

Eine Besonderheit stellen die Prozeß-Sensoren dar, die statistische Daten über die Prozeßzustände und die Transitionen erheben. Sie benötigen **immer** genau einen Parameter, der einen Prozeßzustand repräsentiert. Damit können Informationen über erreichte Zustände, die Anzahl und

Verteilung der schaltenden Transitionen sowie Wartezeiten auf Signaleingaben zustandsspezifisch erhoben werden.

In Tabelle 3 -4 sind die prozeßspezifischen Sensoren dargestell, die vom Laufzeitsystem automatisch aktualisiert werden. Eine Übersicht über alle definierten Sensortypen, deren Syntax sowie der definierten Operatoren findet man in [38] und [91].

Typ	Funktion
FreqSignalReceive	Ermittelt die relativen Häufigkeiten von *empfangenen* Signalen.
FreqSignalSend	Ermittelt die relativen Häufigkeiten von *gesendeten* Signalen.
FreqSignalDrop	Ermittelt die relativen Häufigkeiten von *explizit konsumierten* Signalen.
FreqSignalConsume	Ermittelt die relativen Häufigkeiten von *implizit konsumierten* Signalen.
FreqSignalSave	Ermittelt die relativen Häufigkeiten von *gesicherten* Signalen.
FreqSignalLoose	Ermittelt die relativen Häufigkeiten von *verlorenen* Signalen (wg. Pufferüberlauf).
FreqRequest	Ermittelt die relativen Häufigkeiten der von einem Prozeß *ausgeführten Requests.*
FreqState	Ermittelt die relativen Häufigkeiten von *erreichten* Zuständen.
QueueLength	Ermittelt die relativen Häufigkeiten von wartenden Signalen.
QueueContent	Liefert die Aufteilung der im Puffer *wartenden* Signale.
CountSignalReceive	Ermittelt die Anzahl der vom Prozeß *empfangenen* Signale.
CountSignalSend	Ermittelt die Anzahl der vom Prozeß *gesendeten* Signale.
CountSignalDrop	Ermittelt die Anzahl der *implizit konsumierten* Signale.
CountSignalConsume	Ermittelt die Anzahl der *explizit konsumierten* Signale.
CountSignalSave	Ermittelt die Anzahl der *gesicherten* Signale.
CountSignalLoose	Ermittelt die Anzahl der wegen Pufferüberläufen *verlorenen* Signale.
TransmissionTimeSignal	Ermittelt die Zeit zwischen Absenden eines Signals und dessen Ankunft im Puffer des Empfängers.
CountRequest	Ermittelt die Anzahl der Dienstanforderungen des Prozesses.
ResponseTimeRequest	Ermittelt die Verweilzeiten der Dienstanforderungen eines Prozesses.
CountState	Zählt, wie oft ein Prozeß einen bestimmten Zustand betritt.
FreqTransition	Ermittelt die relativen Häufigkeiten der Transitionsausführungen eines prozesses.
CountTransition	Ermittelt die Anzahl der Schaltvorgänge in einem Zustand des Prozesses.
WaitTimeState	Ermittelt die Zeit, die ein Prozeß auf Signaleingabe in einem Zustand wartet.

Tabelle 3 -4 **Prozeßspezifische Sensoren**

Für QSDL-Prozesse sind spezielle Sensoren definiert worden, die dazu verwendet werden können, benutzerdefinierte Leistungsmaße zu erheben. Diese sog. Basissensoren stellen Schablonen dar, die es ermöglichen, die Semantik der Sensoren und damit die Aktualisierungsereignisse und Werte zu definieren. Dazu existieren für diese Sensoren genauso wie für die Standardsensoren Operatoren, mit deren Hilfe ihre Werte abgefragt und verändert werden können.

Die folgenden drei Basissensoren sind in QSDL definiert:

- BaseCounter
- BaseTally
- BaseFrequency

Für *BaseCounter*-Sensoren sind die Abfrage-Operatoren *Sum*, *SumInt*, *SumTime* und *SumTimeInt* definiert. Mit Hilfe des *Sum-Operators* kann die Summe aller Proben des Sensors abgefragt werden. Für den *SumInt-Operator* gilt dies analog für die Summer aller Proben im letzten Zeitintervall. Die beiden anderen Operatoren dienen zur Abfrage der Anzahl von Proben insgesamt bzw. der Anzahl von Proben im letzten Zeitintervall. Aus der Angabe dieser Werte lassen sich zum Beispiel der Mittelwert des Sensors *S1* durch *Sum(S1)/SumTime(S1)* ermitteln. Analog ergibt sich der Mittelwert des Sensors bezogen auf das letzte Zeitintervall durch *SumInt(S1) / SumTimeInt(S1)*.

Für alle Basissensoren sind ein *Update*- und ein *Clear-Operator* definiert. Der *Update-Operator* dient zur Aktualisierung von Sensoren, während mit Hilfe des *Clear-Operators* ein Sensor auf seine Anfangswerte zurückgesetzt werden kann. Das ist zum Beispiel immer dann nötig, wenn die transiente Einschwingphase beendet ist und das System in seinem stationären Verhalten beobachtet werden soll.

Beispiel 3 -4 **Quellprozeß mit Counter-Sensor**

```
process q1;
   procedure minimum referenced;
   procedure sended  referenced;
   dcl    arrival negexp,
               nextsig real,
               mueoff  real,
               tooff   negexp,
               nextoff real,
               mueon   real,
               toon    negexp,
               bulk    randint,
               bulknumber  integer,
               sended     integer,
               ergebnis real,
               nexton  real;
      /*## SENSOR S1 BaseCounter; ##*/
```

```
state on /*##  awake ergebnis ##*/ ;
   input None;
   decision  ergebnis = nextsig;
        (true)   :  task  nextoff := nextoff - nextsig,
                                nextsig := sample(arrival);
                        task bulknumber := sample(bulk);
                        call sended(bulknumber);
                        /*## UPDATE(S1,bulknumber); ##*/
                        task ergebnis := call minimum(nextsig,nextoff);
                        nextstate -;
        (false)  :  task  nexton := sample(toon);
                        nextstate off;
   enddecision;
   endprocess;
```

```
   start;
   task mueon   := (1.0 / (burst - 1.0)),
          mueoff  := 1.0,
          arrival := init(arrival,lambda),
          tooff   := init(tooff,scale * mueoff),
          toon    := init(toon,scale * mueon),
          bulk    := init(bulk,minarr,maxarr),
          nexton  := sample(toon);
   nextstate off;
```

```
  state off /*## awake nexton ##*/ ;
     input None;
     task nextsig := sample(arrival),
            nextoff := sample(tooff);
     task ergebnis := call minimum(nextsig,nextoff);
     nextstate on;
```

BaseCounter-Sensoren können zum Beispiel dazu verwendet werden, den Durchsatz auf Paketbasis oder das Verhalten von Video- bzw. Audio-Quellen auf Frame-Basis zu ermitteln, vgl. Beispiel 3 -4 und Abbildung 3-6.

Die Spezifikation beschreibt eine On-Off-Quelle. Diese Quelle besitzt die beiden Zustände *on* und *off*, zwischen denen sie ständig wechselt. Im Zustand *off* ist die Quelle inaktiv, während sie im Zustand *on* Frames versendet. Falls die Wechsel zwischen den Zuständen durch Poisson-Prozesse beschrieben sind und auch die Anzahl von Sendeereignisse pro Zeiteinheit Poisson-verteilt sind, so stellt der von der Quelle erzeugte Verkehr einen *Interrupted Poisson Process* (IPP) dar. Diese Quelle kann dazu verwendet werden, das Lastmuster einer sprechenden Person zu approximieren, vgl. Abschnitt 5.2.1. Für diesen Quellprozeß ist der Sensor *S1* definiert. Wann immer die Quelle *Frames* versendet, wird dieser Sensor mit der Anzahl der versendeten Frames aktualisiert. Diese Update-Operation muß in die Spezifikation integriert werden, vgl. Beispiel 3 -4, oben rechts.

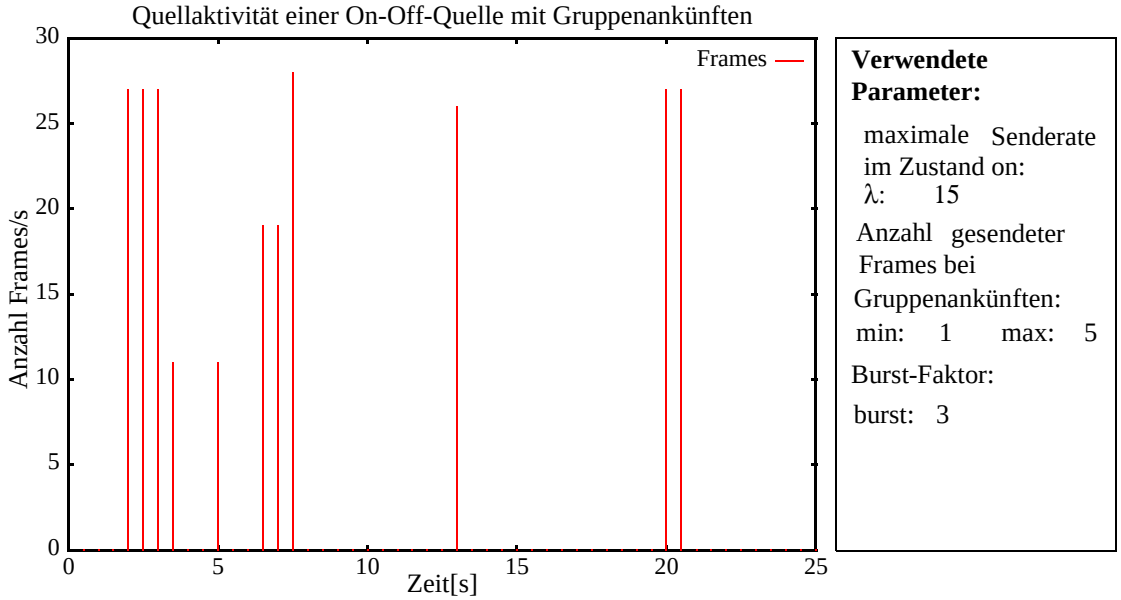


Abbildung 3-6 **Lastverhalten On-Off-Quelle beobachtet mit Counter-Sensor**

Die während einer Simulation ermittelten Werte (hier die Anzahl der versendeten Frames) können graphisch dargestellt werden. In Abbildung 3-6 sind das Verhalten und die verwendeten Parameter dieser Lastquelle dargestellt. In diesem Fall sind die Werte für die Anzahl der versendeten Frames/s über 25 Sekunden ermittelt worden. Dabei sind die Phasen unterschiedlicher Aktivitäten sehr gut zu unterscheiden. Für *BaseFrequency*-Sensoren ist ebenfalls ein *Sum-Operator* definiert, der die Summe aller Proben enthält. Darüberhinaus existieren die beiden Operatoren *AbsFreq* und *RelFreq*, mit deren Hilfe die absoluten, bzw. relativen Häufigkeiten spezieller Wertausprägungen ermittelt werden können. Da *Frequency*-Sensoren ganzzahlige Verteilungen z.B. der Warteschlangenlänge einer QSDL-Maschine ermitteln, könnte eine solche Wertausprägung die Warteschlangenlänge Null betreffen. Damit kann dann ermittelt werden, mit welcher Wahrscheinlichkeit für die Bearbeitung von *Requests* keine Wartezeiten entstehen. Mit Hilfe der Operatoren *LastValue* und *CurrentValue* kann festgestellt werden, welche Ausprägung zuletzt aktualisiert wurde (*LastValue*), bzw. ob der zuletzt aktualisierte Wert mit der letzten ausgeführten Transition verändert wurde (*CurrentValue*[c] = *LastValue*).

c. Liegt die Aktualisierung weiter zurück, ist *CurrentValue* undefiniert

Die größte Anzahl an Operatoren ist für die *BaseTally*-Sensoren definiert. Diese dienen dazu, reellwertige Proben aufzunehmen. Neben denen für die Erfassung der Probenwerte sind eine Reihe von Operatoren definiert, mit deren Hilfe statistische Größen abgefragt werden können.

Für *BaseTally*-Sensoren kann die Anzahl der gesammelten Proben insgesamt und die Anzahl der gesammelten Proben im letzten Intervall mit Hilfe des *Num*- bzw. des *NumInt*-Operators abgefragt werden. Zusätzlich lassen sich der Mittelwert über alle Proben (*Average*), sowie der minimale Probenwert (*Minimum*) und der maximale Probenwert (*Maximum*) abfragen. Weiterhin bietet jeder *BaseTally*-Sensor Operatoren zur Abfrage der Summe aller Proben (*Sum*), der Summe aller Proben im Intervall (*SumInt*), des Mittelwertes im Intervall (*AverageInt*) sowie der Summe über die quadrierten Probenwerte (*SumSquare*).

Diese Angaben können dazu verwendet werden, während der Simulation die Varianz der Stichprobe zu ermitteln. Dabei kann gegenüber der direkten Auswertung der Formel aus (GL.1) durch Anwendung der Formel aus (GL.2) Speicherplatz gespart werden, da nicht alle Stichprobenwerte gespeichert werden müssen.

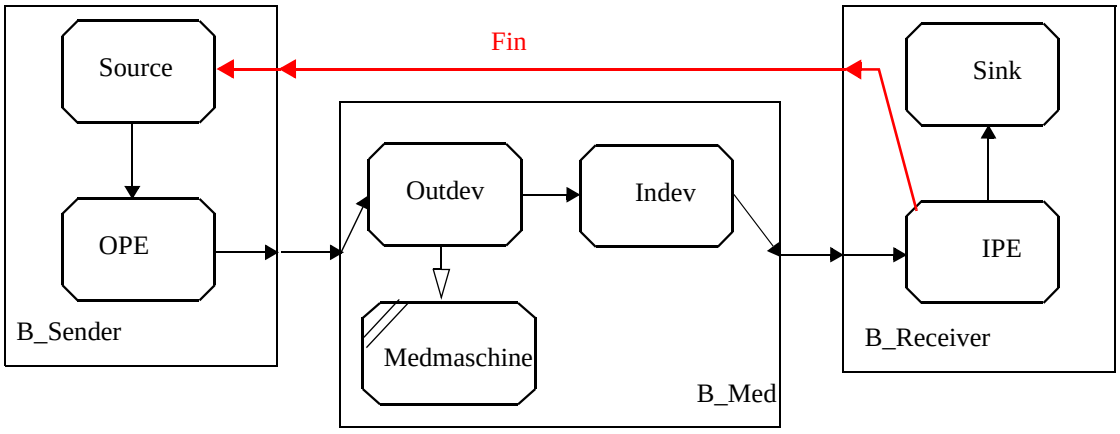
$$s_n^{\,2} = \frac{1}{n-1} \cdot \sum_{i=1}^{n} (x_i - \bar{x}_n)^2 \qquad \text{(Gl. 1)}$$

$$s_n^{\,2} = \frac{1}{n-1} \cdot \left(\sum_{i=1}^{n} x_i^{\,2} - (n \cdot \bar{x}_n^{\,2}) \right) \qquad \text{(Gl. 2)}$$

Die benötigten Größen werden bei der Update-Operation eines Base-Tally-Sensors permanent aktualisiert und können mit *Sum, Num* und *SumSquare* abgefragt und weiterverarbeitet werden.

Somit stehen sie dann zur Berechnung von Konfidenzintervallen entweder mit Hilfe der Tschebyscheff'schen Ungleichung oder auf Basis des zentralen Grenzwertsatzes zur Verfügung. So kann man Performance-Simulationen mit Hilfe von speziellen Kontrollprozessen steuern und ist nicht nur auf das Abbruchkriterium Simulationszeit beschränkt. Die Berechnung von Konfidenzintervallen kann aber auch am Ende einer Simulation vorgenommen werden, um das Vertrauen in die erzielten Ergebnisse zu gewährleisten. Zusätzlich zu den Operatoren zur Abfrage dieser statistischen Werte können auf den Wert und den Zeitpunkt der letzten Probe mit Hilfe der Operatoren *LastSample* und *LastTime* zugegriffen werden.

Die Verwendung von Sensoren zur Steuerung eines QUEST-Simulators wird an einem kleinen Beispiel demonstriert. Das SDL-System besteht aus einem Block *Sender*, einem Block *Empfänger* und einem Block *Kanal*, der die beiden Einheiten verbindet, siehe Beispiel 3 -5.



Beispiel 3 -5 **Übertragungssystem zur Steuerung von QUEST-Simulatoren**

Der Block *B_Sender* enthält eine Sendeinstanz (*OPE*) und einen Lastprozeß (*Source*), der die Belastung des Gesamtsystems ermöglicht. Der Block *B_Med* enthält die Prozesse *Indev* und *Outdev*, die das funktionale und quantitative Verhalten zusammen mit der QSDL-Maschine *Medmaschine* beschreiben. Der Block *B_Receiver* enthält die beiden Prozesse *IPE* und *Sink*. Während der Prozeß *IPE* die Datenpakete vom Netz entgegennimmt, sie auf Fehler prüft und die statistischen Daten erhebt, dient der Prozeß *Sink* lediglich dazu erfolgreich übertragene Pakete aus dem System zu entfernen.

Zusätzlich zur Systembeschreibung wird eine Verbindung vom Prozeß *IPE* des Blocks *B_Receiver* zum Lastprozeß *Source* spezifiziert. Diese Verbindung dient lediglich dazu, die Quelle per *Fin*-Signal zu informieren, daß sie sich beenden kann, wenn die Bestimmung der Durchlaufzeit von Paketen die vorgegebene Genauigkeit erreicht hat. Ist die Quelle nicht mehr existent, gerät das SDL-System gezielt in einen Deadlock, woraufhin QUEST die Simulation beendet.

Im beschriebenen System soll die mittlere Zeit ermittelt werden, die benötigt wird, um Datenpakete vom Sender zum Empfänger zu übertragen. Dabei soll das System solange simuliert werden, bis das 90% Konfidenzintervall eine Größe hat, die 3% des Mittelwertes entspricht.

Der Prozeß *IPE* berechnet aus den Angaben des Paket-Headers die Durchlaufzeit. Die aus jedem Paket ermittelte Laufzeit wird als Argument für die Aktualisierung des Sensors *Durchlaufzeit* verwendet, siehe Abbildung 3-7 oben rechts. Aus den für den Sensortyp *basetally* mitgeführten Werten für *Num, Average* und *SumSquare* berechnet die Instanz die aktuelle Größe des Konfidenzintervalls unter Verwendung des zentralen Grenzwertsatzes.

Die Breite des 1-α -Konfidenzintervalls ergibt sich auf Basis des Zentralen Grenzwertsatzes zu:

$$\delta = \varepsilon_\alpha \cdot \frac{\sigma}{\sqrt{n}} \qquad \text{(Gl. 3)}$$

Dabei bezeichnet ε_α den Wert von ϕ^{-1}, der invertierten N(0,1)-Verteilungsfunktion für das Argument 1-α. Mit n wird der Stichprobenumfang bezeichnet, während σ die Streuung angibt. Dieser Wert kann als Wurzel der Varianz gemäß (GL.2) aus den Angaben *Num, Average* und *SumSquare* berechnet werden.

```
process IPE;
/*## SENSOR
Durchlaufzeit basetally; ##*/
procedure wurzel referenced;

 dcl
   eingang pakettype,
   varianz Real,
   delta   Real,
   phi     Real,
   sqrn    Real,
    h1     Real,
    h2     Real,
    h3     Real,
   laufzeit Duration;
 start;
 nextstate ready;
```

```
state ready;
  input Paket(eingang);
  task laufzeit := NOW - eingang!sendzeitpkt;
  /*## UPDATE(Durchlaufzeit,laufzeit); ##*/
  output Paket(eingang) via toHDLC_Entity;
  decision  Num(Durchlaufzeit) > 1;
    (false) : nextstate -;
    (true)  : task h1 := 1.0 / (Num(Durchlaufzeit) - 1),
                   h2 := Num(Durchlaufzeit) *
                         Average(Durchlaufzeit) *
                         Average(Durchlaufzeit),
                   h3 := SumSquare(Durchlaufzeit) - h2,
                   varianz := h1 * h3;
              task phi := call wurzel(varianz),
                   sqrn := call wurzel(Num(Durchlaufzeit));
              task delta := epsilon * (phi / sqrn);
              decision delta < (tresh *
                                (Average(Durchlaufzeit)));
                (true)  :  output messsig(delta,(thresh *
                           (Average(Durchlaufzeit))));
                           output Fin;
                           nextstate - ;
                (false) : nextstate - ;
              enddecision;
        enddecision;
endstate;
endprocess IPE;
```

Abbildung 3-7 **Prozeß IPE (Ausschnitt)**

Liegt der Wert für δ unterhalb der spezifizierten Genauigkeit (*delta < (thresh * (Average(Durchlaufzeit))*), sendet die Instanz das Signal *Fin* an den Prozeß *Source*. Diese führt dann die SDL-Stop-Aktion aus, woraufhin das Gesamtsystem in einen Deadlock gerät. Das führt zur Beendigung des Simulators, da keine Prozeßinstanz mehr aktiviert werden kann, siehe Abbildung 3-8. Da die Größenordnung des Mittelwertes im vorhinein in der Regel nicht bekannt ist, wird für die gewünschte Größe des Konfidenzintervalls kein absoluter Wert angegeben. Statt dessen wird die Größe in Abhängigkeit des Mittelwerts definiert.

Die Werte für ε_α werden ebenso als *externe Synonyme* deklariert, wie der Wert für *thresh*, der das Verhältnis von Mittelwert und Intervallbreite angibt. Somit erhält der Systementwickler ein flexibles Modell, mit dessen Hilfe er gezielt Ergebniswerte auswählen kann, die mit einer von ihm festgelegten Genauigkeit ermittelt werden. An dieser Stelle sei darauf hingewiesen, daß für die Fehlerschranke Θ gilt: $\theta = \frac{k \cdot s_n}{\sqrt{n}}$, d.h. sie ist umgekehrt proportional zu der Wurzel aus der Anzahl der Beobachtungen. Somit ist für die Halbierung der Fehlerschranke die Erhöhung der Beobachtungen um den Faktor vier notwendig. In der Praxis werden in der Regel keine allzuhohen Genauigkeitsanforderungen gestellt, um die Simulationszeiten nicht unerträglich hoch werden zu lassen.

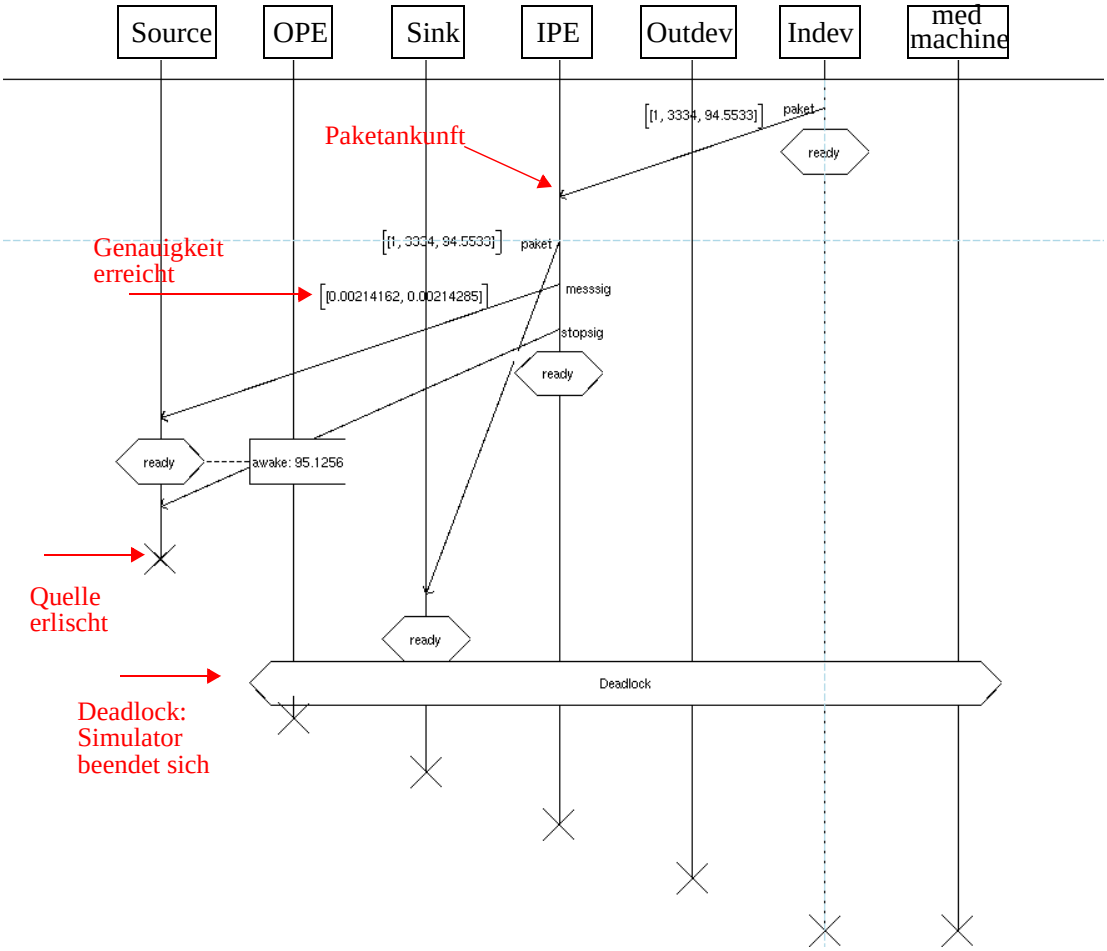


Abbildung 3-8 **MSC Simulationsende**

3.7 QSDL-Mutex

Zur weiteren Synchronisation von QSDL-Prozessen können sogenannte mutex[d] innerhalb eines QSDL-Systems definiert werden. Für einen mutex sind die beiden Operationen *acquire* und *release* definiert. Mit *acquire* wird ein mutex angefordert und mit *release* wieder frei gegeben. Führt ein Prozeß die Operation *acquire* auf einem mutex aus, so ist dieser mutex für weitere *acquire*-Operationen gesperrt. Wollen parallel weitere Prozesse diesen mutex anfordern, werden sie in einer FIFO-Warteschlange gespeichert. Ein wartender Prozeß kann sein *acquire* dann ausführen, wenn sein direkter Vorgänger den mutex mit *release* wieder freigegeben hat.

Somit läßt sich auf einfache Weise der exklusive Zugriff auf bestimmte Bereiche, z.B. Datenstrukturen, realisieren. Die Verwendung des mutex-Konstrukts bei der Modellierung unterschiedlicher Implementierungsstrategien von SDL-Systemen wird in Abschnitt 4.2.3 demonstriert.

d. Der Begriff ist aus dem Bereich der Betriebssysteme entliehen, wo er als Kurzform für *mutual exclusion* verwendet wird.

3.8 Verhältnis QSDL zu SDL

QSDL ist eine echte Erweiterung der Sprache SDL, die die Präzisierung des zeitlichen Verhaltens eines SDL-Systems beschreibt. Während eine SDL-Spezifikation das funktionale Verhalten beschreibt, wird durch ein QSDL das dynamische Verhalten unter zeitlichen Restriktionen beschrieben. Durch die Ergänzungen innerhalb des QSDL-Systems werden die Ereignisse zusätzlich zur Reihenfolge, die durch das SDL-System festgelegt ist, durch Ereigniszeitpunkte gekennzeichnet. Somit ist die QSDL-Beschreibung zeitlich präziser als die SDL-Beschreibung.

Durch die QSDL-Ergänzungen kann es aber nicht nur zu Präzisierungen kommen, sondern auch zur Einschränkung des Zustandsraums. Dieses Phänomen, das auch beim Übergang von zeitlosen zu zeitbehafteten Petri-Netzen bekannt ist, soll an einem kleinen Beispiel erläutert werden.

Beispiel 3 -6 **SDL versus QSDL**

```
Process Proc_Sender;
Start;
Nextstate ready;
Input None;
/*## Request compute(1.0); from M1 ##*/
Output sig_a;
Nextstate ready;
Endprocess Proc_Sender;
```

```
Process Proc_Receiver;
Start;
Nextstate ready;
Input sig_a;
Task 'Aktionen zur Verarbeitung von
      sig_a';
/*## Request compute(0.5); from M1##*/
Nextstate ready;
Endprocess Proc_Receiver;
```

Beispiel 3 -6 beschreibt das Verhalten einer einfachen Sende- und Empfangsinstanz. Durch die SDL-Spezifikation wird für den Prozeß *Proc_Sender* festgelegt, daß er durch eine Spontantransition getriggert, das Signal *sig_a* versendet. Die Spezifikation legt für den Prozeß *Proc_Receiver* fest, daß dieser im Zustand *ready* das Signal *sig_a* empfangen kann, die mit dem Empfang verbundenen Aktionen ausführt und schließlich wieder in den Zustand *ready* wechselt, in dem er wieder empfangsbereit ist.

Die QSDL-Spezifikation präzisiert dieses durch das SDL-System festgelegte Verhalten. Durch die Angabe *Request compute(1.0)* wird festgelegt, daß für die Ausführung der Transition der Maschinendienst *compute* mit einem Bedienwunsch von 1.0 Zeiteinheiten aufgerufen wird. Ebenso wird für die Ausführung der Transition des Prozesses *Proc_Receiver* der Maschinendienst *compute* mit einem Bedienwunsch von 0.5 Zeiteinheiten aufgerufen.

Durch diese Präzisierung werden die beiden Prozesse über die QSDL-Maschine *M1* synchronisiert. Dadurch wird eine Überlastung des Empfangsprozesses vermieden, die laut SDL-Spezifikation noch möglich ist. Unterstellt man gemäß SDL-Semantik die nebenläufige Ausführung beider Prozesse, so existiert eine Zustandsfolge, die zu einer unendlich großen Anzahl von Signalen in der Eingangswarteschlange des Prozesses *Proc_Receiver* führt.

Diese Zustandsfolge, die erreicht wird, wenn nur die immer aktivierte Spontantransition des Prozesses *Proc_Sender* ausgeführt wird, ist im QSDL-System nicht mehr möglich. Durch die Kopplung der beiden Prozesse über die Maschine *M1* werden nach dem zweiten Schalten der Spontantransition beide Prozesse immer abwechselnd ihre Transitionen ausführen und somit die Anzahl der Signale in der Eingangswarteschlange den Wert Zwei nicht übersteigen.

Durch die Einführung des awake-Konstrukts ist es möglich, in einem QSDL-System die Reaktionen zu bestimmten Zeitpunkten zu garantieren, während in einem SDL-System diese Reaktionen frühestens zu den spezifizierten Zeitpunkten möglich sind.

Durch den Ansatz, das zeitliche Verhalten zu präzisieren, bzw. den Zustandsraum durch zeitliche Restriktionen zu verkleinern, gibt es in einem QSDL-System keinen Ablauf, der nicht auch durch das SDL-System festgelegt ist. Somit kann man sicher sein, kein grundsätzlich anderes[e] Verhalten spezifiziert zu haben.

3.1 Zusammenfassung Kapitel 3

In den zurückliegenden Abschnitten ist die gemeinsam mit Marc Diefenbruch entwickelte Sprache QSDL mit ihren Konzepten und Sprachelementen vorgestellt worden. Darüberhinaus ist der Einsatz wesentlicher Elemente der Sprache wie awake oder der definierten Sensoren an Beispielen demonstriert worden. Mit der Definition der Sprache QSDL und der Implementierung des Werkzeugs QUEST[f] sind die **Voraussetzungen** zur entwurfsbegleitenden Performance-Analyse von SDL-Systemen gelegt. Der Erfolg des Ansatzes hängt jedoch stark mit der Entwicklung einer Methodik zur Anwendung der Sprachelemente und der Analysemöglichkeiten des Werkzeugs zusammen. Eine solche Methodik wird im folgenden für den Entwurfs- und Implementierungsprozeß von Kommunikationsprotokollen entwickelt und vorgestellt.

e. An dieser Stelle sei darauf hingewiesen, daß man bei Verwendung des Output-Delay-Konstruktes in Verbindung mit zufälligen Verzögerungswerten eine Signalüberholung in einem QSDL-System erreichen kann. Dies ist aber in einem SDL-System nicht möglich.

f. Das Werkzeug QUEST in durch W. Textor, J. Rühl, C. Rodemeyer und Axel Hirche unter der Leitung von Marc Diefenbruch im Rahmen der Arbeiten [38], [61],[93] und [110] implementiert und im Rahmen dieser Arbeit getestet, validiert und ergänzt worden.

Kapitel 4

Die QUEST-Methodik

4.1 Grundlagen

Die Effizienz und somit die Leistungsfähigkeit eines verteilten Systems hängt nicht nur vom Design, sondern auch von der Implementierung ab. Dabei erfordern effiziente bzw. performante Protokollimplementierungen eine Umsetzung der logischen Struktur in konkrete Hard-/Software-Abläufe. Diese Umsetzung wird im sog. Implementierungs-Entwurfsprozeß vorgenommen. Dabei erweist sich die nachträgliche Einbettung spezifischer Implementierungsstrategien als nicht so effizient, wie deren konsequente Berücksichtigung von Beginn des Entwurfes an [76]. Zu den wesentlichen Aufgaben des Implementierungsentwurfes zählt die Ableitung eines Implementierungsmodells, das die Architektur der zu realisierenden Implementierung vorgibt.

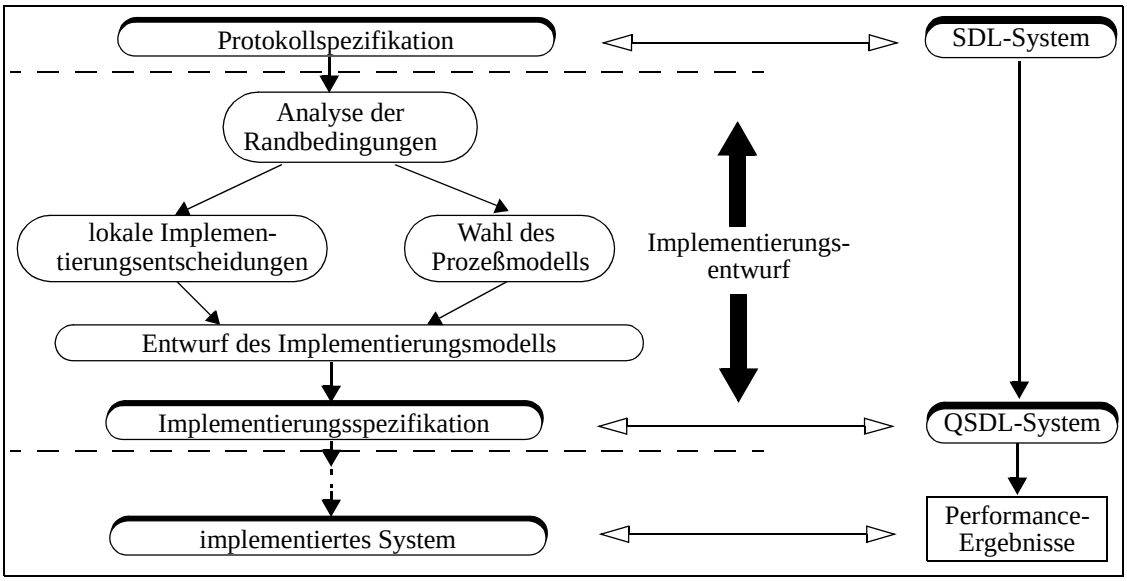


Abbildung 4-1 **Implementierungsentwurf und die QUEST-Methode**

Der Implementierungsentwurf wird durch die Implementierungsspezifikation dokumentiert. Bei Anwendung formaler Methoden stellt die Implementierungsspezifikation in der Regel eine Verfeinerung der Protokollspezifikation dar, in der alle Informationen ergänzt sind, die zu einer vollständigen Implementierung notwendig sind. Insbesondere sind hier Zeitparameter, Präzisierungen wie Beseitigung spontaner Transitionen und anderer Nichtdeterminismen sowie Angaben über die Implementierungsarchitektur enthalten.

Im Falle des SDL-basierten Entwurfs lautet somit die Aufgabe, aus der SDL-Spezifikation ein Implementierungsmodell zu entwickeln. Dabei ist die konkrete Entwicklung eines solchen Modells von einer Reihe von Nebenbedingungen abhängig. Einige der Bedingungen werden direkt durch die SDL-Spezifikation vorgegeben. Andere Abhängigkeiten ergeben sich durch die verwendete Implementierungsumgebung aber auch durch lokale Entscheidungen und das gewählte Prozeßmodell.

Der Implementierungs-Entwurfsprozeß, der die Entwicklung performanter Systeme als Ziel hat, kann durch modellgestützte Performance-Analysen begleitet und somit optimiert werden. Dabei kann die QUEST-Methode entwurfsbegleitend eingesetzt werden, da sie die Spezifikation der oben genannten zusätzlichen Angaben erlaubt, siehe Abbildung 4-1. Das QSDL-Modell stellt dabei einen Repräsentanten der Implementierungsspezifikation dar, der die Auswirkungen der getroffenen Entscheidungen auf die Performance des implementierten Systems beschreibt. Somit sollten, „gute“ Parametrisierung vorausgesetzt, die Performance-Ergebnisse des QSDL-Modells mit den Werten übereinstimmen, die anhand des implementierten Systems gemessen werden können.

Im folgenden wird nun exemplarisch dargestellt, wie einzelne Entscheidungen im Implementierungs-Entwurfsprozeß in entsprechende QSDL-Konstrukte umgesetzt werden und wie die Auswirkungen auf die Performance untersucht werden können. Darüberhinaus werden allgemeine Grundsätze für die Anwendung der QUEST-Methode und Herangehensweisen bei der Analyse spezieller Probleme entwickelt. Zu diesen speziellen Problemen zählen Analysen von Ursache-Wirkungsbeziehungen, die Findung von Performance-Engpässen oder die transiente Analyse von Stressituationen bei kurzfristiger Überlast oder nach dem Auftreten von Fehlern.

4.2 Implementierungsentwurf für Prozeßmodelle und Schnittstellen

Die Kodierung einzelner Protokollinstanzen zählt zu den wesentlichen Aufgaben bei der Implementierung von Protokollen. Sie ist stark durch die Spezifikation der Instanzen als erweiterte endliche Automaten vorgegeben. Allerdings existieren unterschiedliche Möglichkeiten, Zustandsautomaten zu implementieren [76].

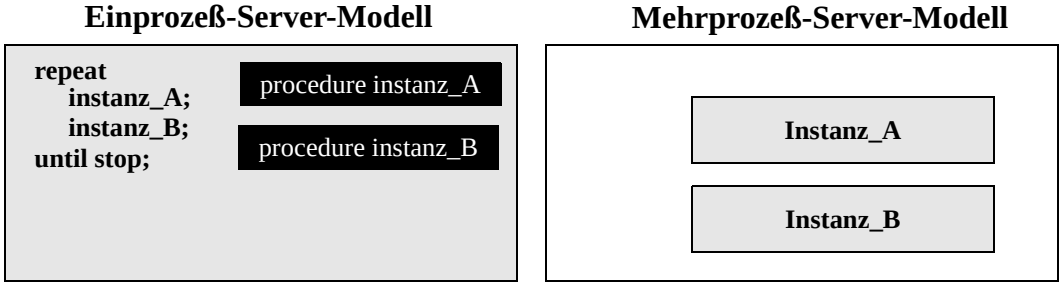


Abbildung 4-2 **Server-Modell für die Implementierung von Instanzen, mod. nach [76]**

So ist ein Prozeß auf Betriebssystemebene je Instanz genauso denkbar, wie mehrere Prozesse je Instanz, aber auch die Zusammenfassung mehrerer Instanzen zu einem Prozeß ist eine Möglichkeit der Abbildung zwischen der logischen Prozeßstruktur der Spezifikation und der Prozeßstruktur, die das Betriebssystem der Zielumgebung vorgibt. Für die Protokollimplementierung haben sich das *Server-Modell* und das *Activity-Thread-Modell* als die wesentlichen Grundmodelle herausgebildet [76].

Im Server-Modell wird jede Instanz bzw. auch mehrere Instanzen durch einen zyklisch ablaufenden sequentiellen Prozeß implementiert, der ein Ereignis aus einer zugehörigen Warteschlange entnimmt, den Ereignistyp bestimmt und dann zu der entsprechenden Aktion (Transition) verzweigt. Nach der Ausführung der Aktionen kehrt der Prozeß an die zentrale Wartestelle zurück und entnimmt, falls vorhanden, das folgende Ereignis aus der Warteschlange. Werden innerhalb der Aktionen Ausgabeereignisse generiert, so werden diese in die Warteschlange des Zielprozesses geschrieben.

Innerhalb des Server-Modells haben sich zwei Varianten entwickelt, siehe Abbildung 4-2. Im sog. *Einprozeß-Server-Modell* werden alle zu implementierenden Instanzen (SDL-Prozesse) als Prozeduren innerhalb eines Betriebssystemprozesses realisiert, wobei jede Prozedur dem Prinzip des Server-Modells folgt. Die Abarbeitungsreihenfolge wird dabei durch die Implementierungsumgebung festgelegt. In der Regel werden die Prozeduren nach dem Round-Robin-Prinzip nacheinander aufgerufen.

Bei der Mehrprozeß-Lösung wird jede Protokollinstanz auf einen Betriebssystemprozeß abgebildet. Die Abarbeitungsreihenfolge wird hierbei durch den Betriebssystem-Scheduler festgelegt. Eng verknüpft mit der Wahl des Prozeßmodells ist die Realisierung der Schnittstellen zwischen einzelnen Instanzen. Während Schnittstellen zwischen Betriebssystemprozessen auf die Verwendung von IPC[a]-Mechanismen angewiesen sind, besteht bei der Realisierung von prozeßinternen Schnittstellen ein größerer Freiheitsgrad. So können hier *puffernde Schnittstellen* über spezielle Datenstrukturen verwendet werden, aber auch *aktionsorientierte Schnittstellen*, die auf Prozeduraufrufen basieren. Implementierungen, die nach dem Server-Modell arbeiten, verwenden in der Praxis meist Schnittstellen mit puffernder Übergabe.

Activity-Thread-Implementierungen werden in der Regel mit aktionsorientierten Schnittstellen versehen. Weit verbreitet sind Mischformen, bei denen die Vorteile puffernder und aktionsorientierter Schnittstellen kombiniert werden. Bei diesen Mischformen werden direkte Prozeduraufrufe immer nur in einer Richtung, *Upcall* oder *Downcall*, verwendet, während in der jeweils anderen Richtung die puffernde Schnittstelle verwendet wird. Man spricht daher auch von *Upcall-Schnittstellen*, bzw. von *Downcall-Schnittstellen* [76].

Diese Schnittstellengestaltung, aber auch die Wahl des Server-Modells haben einen Einfluß auf die Performance und somit auf die Gestaltung des QSDL-Modells. Welche Aspekte bei der Erstellung des QSDL-Modells in Abhängigkeit von der gewählten Schnittstelle und des gewählten Prozeßmodells zu beachten sind, wird im folgenden anhand des *InRes*-Protokolls erläutert. Das *InRes*-Protokoll ist ein verbindungsorientiertes Protokoll, das zwischen zwei Protokollinstanzen *Initiator* und *Responder* operiert. Der Austausch der Daten geschieht über ein unzuverlässiges Medium, das die zur Übertragung anliegenden Daten verlieren bzw. verfälschen kann. Details zu diesem Protokoll und den ausgetauschten Dienstprimitiven entnehme man [62]. Die Struktur der SDL-Spezifikation des System ist in Abbildung 4-3 dargestellt.

a. IPC = Inter Process Communication

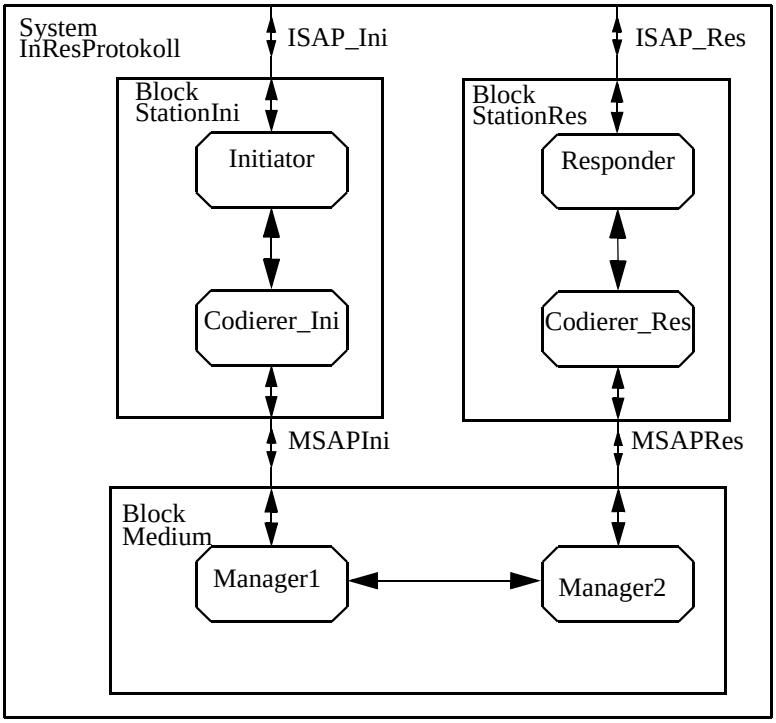


Abbildung 4-3 **System InRes-Protokoll Übersicht**

4.2.1 Mehrprozeß-Server-Implementierung mit separaten Betriebssystemprozessen

Bei einer Implementierung nach dem Mehrprozeß-Server-Modell wird jeder SDL-Prozeß als separater Betriebssystemprozeß aufgefaßt. Daher kann jeder dieser Prozesse unabhängig von anderen Prozessen seine Aufgaben erfüllen. Synchronisationspunkte bzw. Abhängigkeiten ergeben sich lediglich dadurch, daß die Prozesse Signale austauschen und so möglicherweise ein Prozeß auf einen anderen warten muß, bis dieser die benötigten Informationen bereitstellt. Für jeden der Prozesse **kann** ein separater Prozessor bereitgestellt werden, der den Prozessen exklusiv zur Verfügung steht, siehe Abbildung 4-4. Alternativ könnte auch ein Prozessor für die beiden SDL-Prozesse *Initiator* und *Codierer_Ini* zur Verfügung gestellt werden. Beide Optionen stehen bei dieser Art der Implementierung offen.

Die Prozessoren stellen die Maschinendienste zur Verfügung, die von den Prozessen benutzt werden, um ihre Aktionen auszuführen. Dazu zählen z.B. die Analyse der Schnittstellenereignisse und der übergebenen Daten, das Kodieren von PDUs, die Übergabe von Daten an andere Prozesse, et cetera. Die Geschwindigkeit, mit der diese Dienste erbracht werden, kann z.B. in Instruktionen pro Sekunde angegeben werden. So erbringt eine 20 MIPS-Maschine 20 Millionen solcher Instruktionen pro Sekunde. Um die Angaben für eine Performance-Analyse zu vervollständigen, muß nun lediglich die Anzahl der Instruktionen bekannt sein, die für die oben genannten Aufgaben benötigt werden. Obwohl die MIPS-Rate als einfaches Maß für die Leistungsfähigkeit weit verbreitet ist, sei an dieser Stelle auf die Problematik verwiesen, die sich ergibt, wenn man die MIPS-Rate als Vergleichsmaß heranzieht. Da diese vom Befehlssatz abhängt, werden Vergleiche zwischen Rechnern mit unterschiedlichen Befehlssätzen erschwert. Weiterhin variieren die MIPS-Raten zwischen verschiedenen Programmen, die auf einem Rechner ausgeführt werden, und können sich umgekehrt proportional zur Leistung verhalten [58].

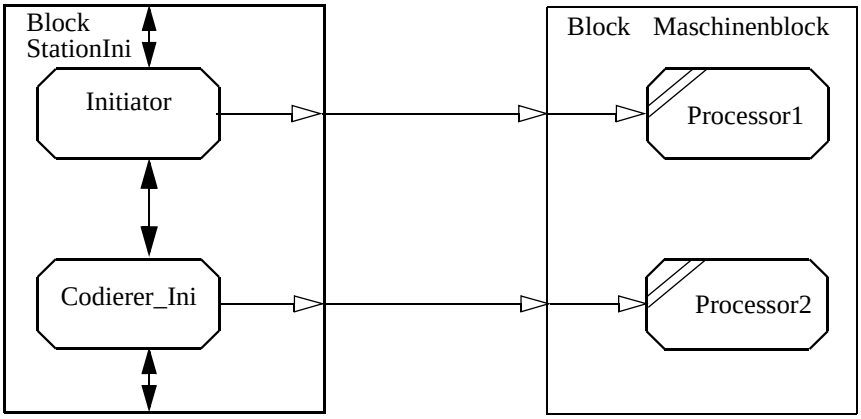


Abbildung 4-4 **QSDL-Modell der Implementierungsarchitektur**

Um die Angaben für die Aufwände einzelner Aktionen (Zuweisungen, Vergleiche, Schleifen) zu erhalten, können Techniken zur Lastvoraussage für neue Anwendungen verwendet werden, die auf Graphmodellen basieren [42]. Das Ziel dieses Ansatzes ist es, aus einem Programmablaufdiagramm mit Anweisungen, Verzweigungen und Verschmelzungen einen Graphen zu erzeugen, dessen Knoten die Verzweigungs- bzw. Verschmelzungspunkte bilden. Die Kanten enthalten die Angaben über die Belegung einzelner Ressourcen, die benötigt werden, um die Aktionen zwischen den Knoten auszuführen. Die Verzweigungsknoten sind mit Wahrscheinlichkeiten beschriftet, mit denen die möglichen Pfade durchlaufen werden. Diese Wahrscheinlichkeiten können von externen Bedingungen, der Anwendungsstruktur oder von Dateninhalten abhängen.

Dieser Ansatz kann direkt in den SDL-basierten Entwurf übernommen werden, da die Automaten, die die Instanzen beschreiben, ebenfalls als Graphen aufgefaßt werden können. Die Knoten stellen die Automatenzustände dar und die Kanten beschreiben die SDL-Aktionen, die innerhalb der Transitionen ausgeführt werden. Die Wahrscheinlichkeiten an Verzweigungspunkten brauchen bei Nutzung des QUEST-Ansatzes nicht berechnet zu werden, da die Auswahl der Transitionen von den anliegenden Signalen und den Prozeßzuständen abhängt. Somit müssen lediglich die Ressourceverbräuche der mit einer Transition verbundenen Aktionen bestimmt werden. Das folgende Beispiel erläutert dieses Vorgehen.
Die SDL-Spezifikation, siehe Abbildung 4-5 (1), legt fest, daß die Protokollinstanz *Initiator* im Zustand *connected* bei Empfang des Signals *DatReq* die Variable *counter* initialisiert, einen internen Timer zur Zeitüberwachung setzt und die mit dem SDL-Input verbundenen Daten zusammen mit einer Sequenznummer übergibt. Nach Abschluß aller Aktionen wechselt die Instanz in den Zustand *waitdatres*.

In die QSDL-Spezifikation müssen nun die Auswirkungen der Implementierung nach dem Mehrprozeß-Server-Modell für diesen Teil der SDL-Spezifikation eingetragen werden. Die QSDL-Maschine *Processor1* stellt den notwendigen Maschinendienst *compute* zur Verfügung. Die Geschwindigkeit für diesen Dienst ist in Instruktionen pro Sekunde angegeben. Daher gilt es nun, die Anzahl der Instruktionen zu bestimmen, die für die einzelnen Aktionen durchgeführt werden müssen. Die erste SDL-Aktion, die von der Instanz durchgeführt wird, ist die Input-Aktion. Aus den Angaben der Implementierung, siehe Abbildung 4-5, (2), kann man ablesen, daß zur Ausführung dieser SDL-Aktion zunächst die Prozedur *get_event* aufgerufen werden muß und deren Rückgabewert in die Variable *myevent* geschrieben wird.

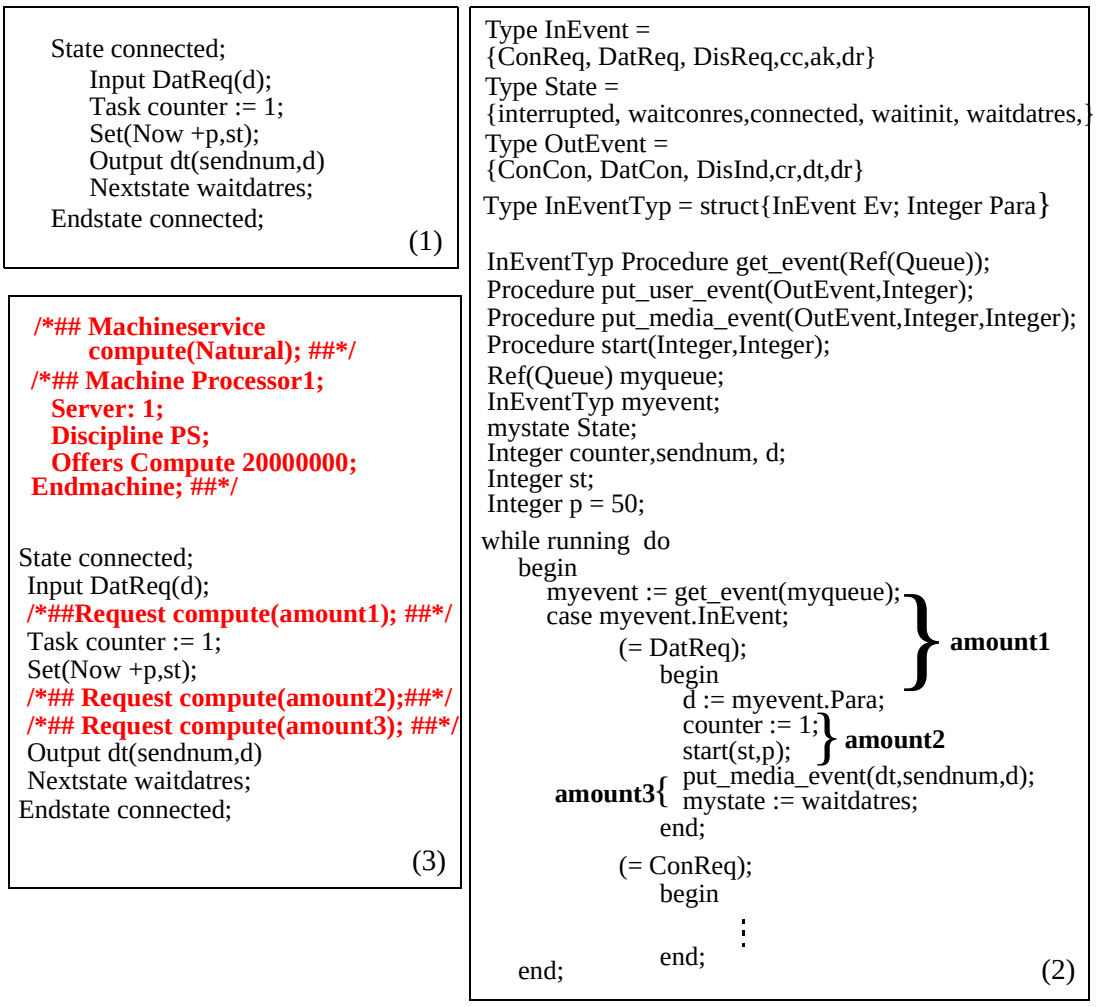


Abbildung 4-5 **Ausschnitt Instanz Initiator (Spezifikation SDL/PR (1), Implementierung Pseudo-Code (2), und Implementierungsspezifikation QSDL/PR (3))**

Danach ist der Vergleich des Rückgabewertes mit einer Konstanten (*DatReq*) nötig. Als letzte Aktion werden die mit dem Ereignis verbundenen Daten in die Variable *d* geschrieben. Die Auswirkungen dieser Prozeduraufrufe und Anweisungen finden sich als Wert in dem Ausdruck *amount1* wieder, der als Bedienwunsch dem ersten QSDL-Request mitgegeben wird. Bei der Abschätzung des Aufwandes für den Aufruf von *get_event* sind die Anweisungen innerhalb des Prozedurrumpfes zu berücksichtigen. Weiterhin sind der meist konstante Aufwand für den Aufruf einer internen Prozedur und der Aufwand für die Übernahme der Daten, die mit dem Aufruf verbunden sind, zu beachten. Dieser wiederum hängt stark davon ab, wie die Schnittstelle zwischen dem übergebenden Prozeß und der Instanz *Initiator* realisiert ist. In der Regel wird diese Übergabe über einen Systemaufruf (z.B. `read` in UNIX) realisiert. Da dieser Aufruf dazu führt, daß das Betriebssystem aktiviert wird, um die Daten zu kopieren, muß noch der Overhead für den damit verbundenen Kontext-Switch berücksichtigt werden. Alternativ kann dieser nicht nur bei Systemaufrufen stattfindende Kontext-Switch in einer reduzierten Bediengeschwindigkeit aller PS-Maschinen berücksichtigt werden, die mehrere Prozesse bedienen. Arbeitet eine CPU mit einer Zeitscheibe von 100 Millisekunden je Prozeß und werden für einen Kontext-Switch

beim Prozeßwechsel 5 Millisekunden benötigt, so reduziert sich die effektive Geschwindigkeit auf 95.24% der nominalen Geschwindigkeit. Die Berücksichtigung von Kontextwechseln durch explizite Maschinendienstanforderungen setzt voraus, daß man die Anzahl der Wechsel, die mit einer Transition verbunden sind, kennt. Diese Zahl schwankt aber mit der Anzahl der bereiten Betriebssystemprozesse und ist daher nicht voraussagbar.

Konkrete Werte für die benötigten Parameter lassen sich z.B. durch Instrumentierung prototypischer Implementierungen durch Messungen des CPU-Zeitverbrauchs bestimmen oder aus Erfahrungen mit früheren Implementierungen ableiten.

Analog dazu wird der Zeitverbrauch der Operationen zur Initialisierung der Variablen *counter* und die Initialisierung des Timers in dem Wert *amount2* zusammengefaßt. Der Aufwand für die Ausgabe der Daten und der Zustandswechsel wird im *amount3* beschrieben. Der Zustandswechsel wird entweder durch eine einfache Zuweisungsoperation oder durch Umsetzung eines Pointers realisiert. Letzteres wird dann angewendet, wenn die Zustände und die definierten Transaktionen in einer Tabelle codiert sind. Der Aufwand für die Übergabe der Daten an den empfangenden Prozeß hängt wiederum sehr stark vom verwendeten IPC-Mechanismus ab. Hier gelten dieselben Überlegungen, die schon bei der Übernahme der Daten durch die SDL-Input-Aktion angestellt wurden.

Können diese detaillierten Bedienwünsche für die durchzuführenden Aktivitäten nicht ermittelt werden oder erscheint diese Feingranularität der Betrachtung als nicht notwendig, so kann alternativ ein allgemeiner Maschinendienst, z.B. *proceede_transition* definiert werden. Für diesen Fall muß dann aber für jede in den SDL-Prozessen definierte Transition der Bedienwunsch ermittelt werden. Im oben vorgestellten Beispiel wäre der Bedienwunsch als Summe von *amount1* bis *amount3* anzugeben. Auch hier ist die Vermessung einer prototypischen Implementierung dieser Anweisungssequenz nötig, um an die benötigten Daten zu gelangen. Bei dieser aggregierten bzw. grobkörnigen Art der Modellierung entfällt in der Regel auch der Vorteil, die benötigten Werte aus früheren Projekten zu übernehmen, da die Kombinationen der Einzelaktionen in der Regel systemspezifisch sind, während die Einzelaktionen doch mit einer größeren Wahrscheinlichkeit in neuen Projekten wieder auftauchen. Aus modelltechnischen Gründen könnten aber die Bedienwünsche zunächst aus den Einzelaktionen abgeleitet werden und dann zu einem Request zusammengefaßt werden. Allerdings ist dabei zu berücksichtigen, daß bei Maschinen, die nicht als PS-Maschinen definiert sind, ein zusätzlicher Anteil an Wartezeiten im Modell entsteht, da jeder Request wieder um die Maschine konkurrieren muß.

4.2.2 Mehrprozeß-Server-Implementierung mit Threads

In einer Mehrprozeß-Server-Implementierung mit Threads wird eine Anzahl von SDL-Prozessen der formalen Spezifikation in Threads umgesetzt. Diese Umsetzung beinhaltet dieselben Freiheitsgrade wie eine Implementierung als separater Betriebssystemprozeß, sofern eine entsprechende Thread-Unterstützung durch ein Laufzeitsystem gegeben ist. Alle Threads, die innerhalb eines Betriebssystemprozesses zusammengefaßt werden, besitzen einen gemeinsamen Adreßraum, gemeinsame globale Variablen, gemeinsame Dateien, Signale und Timer. Jeder Thread besitzt aber einen eigenen Programmzähler, eigene Registermengen, Zustände etc. Threads werden daher auch als Leichtgewichtsprozesse bezeichnet, da sie sich wie eigenständige Prozesse verhalten können.

Daher ergeben sich für die Struktur des Performance-Modells die aus Abschnitt 4.2.1 bekannten Freiheitsgrade, d.h. für jeden SDL-Prozeß **kann** eine separate QSDL-Maschine definiert werden. Die Aufwände für die lokalen Zuweisungen durch SDL-Aktionen, die innerhalb der Transitionen durchgeführt werden, können direkt übernommen werden. Allerdings reduziert sich durch die

Verwendung von Threads der Aufwand, der durch Kontextwechsel anfällt. Während ein Kontextwechsel in einem klassischen Betriebssystem im Bereich von Millisekunden liegt, werden für einen Thread-Wechsel lediglich wenige Mikrosekunden benötigt.

Daher kann bei der Bestimmung der Bediengeschwindigkeit für die Maschine *Processor*, siehe Abbildung 4-6, auf die Skalierung wegen Thread-Wechsel in der Regel verzichtet werden. Bei der Kommunikation zwischen Threads unterschiedlicher Betriebssystemprozesse werden genauso die bereitgestellten IPC-Mechanismen verwendet, wie bei der Kommunikation zwischen Betriebssystemprozessen. Bei der Kommunikation zwischen Threads eines Betriebssystemprozesses wird aus Effizienzgründen der gemeinsame Speicherbereich der Threads verwendet. Hier “kostet” ein Kommunikationsvorgang genausoviel, wie die Zuweisung an eine thread-lokale Variable. Allerdings muß dabei dann der Aufwand berücksichtigt werden, der notwendig ist, den Zugang zu der verwendeten Datenstruktur exklusiv bereitzustellen. Dazu kann bei Verwendung der QUEST-Methode das mutex-Konstrukt verwendet werden.

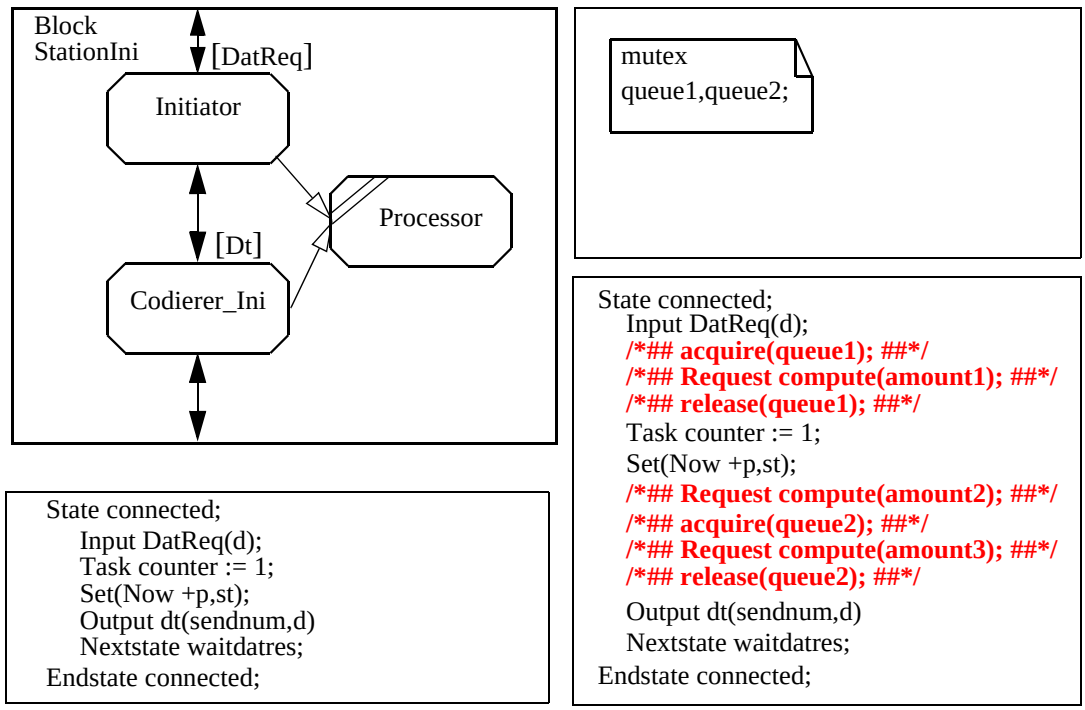


Abbildung 4-6 **QSDL-Modell der InRes-Protokoll Thread-Implementierung (Ausschnitt)**

Als gemeinsamer Speicherbereich für die beiden Threads, die die SDL-Prozesse *Initiator* und *Codierer_Ini* implementieren, kommen die SDL-Eingabewarteschlangen infrage, über die die Kommunikation zwischen den SDL-Prozessen abgewickelt wird. Daher wird für beide Warteschlangen jeweils ein mutex definiert, siehe Abbildung 4-6 oben rechts. Vor jeder Entnahme aus der Schlange (SDL-Input) und vor jeder Hinzufügung (SDL-Output) muß die Zugriffsberechtigung auf die Datenstruktur für den ausführenden Thread (SDL-Prozeß) erteilt werden. Daher werden alle SDL-Input-Aktionen und alle SDL-Output-Aktionen, die auf Warteschlangen operieren, die als gemeinsamer Datenbereich von Threads implementiert sind, durch *acquire*- und *release*- Operationen auf dem zugehörigen mutex geschützt, siehe Abbildung 4-6 unten rechts. Somit wird ein paralleler Zugriff auf diese kritischen Bereiche verhindert und gleichzeitig das

zeitliche Verhalten von parallel ablaufenden Threads, die auf gemeinsamen Speicherbereichen operieren, nachgebildet.

4.2.3 Einprozeß-Server-Implementierung mit Prozeduren

In dieser Implementierungsvariante werden SDL-Prozesse als Prozeduren realisiert. Diese Prozeduren werden in einem Betriebssystemprozeß im *Round-Robin*-Verfahren zyklisch aufgerufen, siehe Abbildung 4-2. Jede der Prozeduren arbeitet nach dem allgemeinen Server-Modell, d.h. sie ermittelt das anliegende Ereignis, verzweigt zu den ereignisabhängigen Aktionen und kehrt dann zur zentralen Entnahmestelle zurück. Auf diese Art und Weise kann jeder SDL-Prozeß pro Zyklus des umgebenden Implementierungsprozesses eine Transition ausführen, sofern ein auslösendes Ereignis (SDL-Signal) anliegt.

Diese Art der Implementierung hat einige Konsequenzen für die Abläufe und somit für die Performance des Systems. Diese Konsequenzen spiegeln sich auch im QSDL-Modell wider. Da die SDL-Prozesse als Prozeduren realisiert werden, die zyklisch sequentiell abgearbeitet werden, **müssen** alle beteiligten Prozesse ihre zeitverbrauchenden Dienstaufrufe an dieselbe QSDL-Maschine richten. Diese Maschine muß als FCFS-Maschine mit genau einem Bediener spezifiziert werden, da andernfalls die exklusive Bedienung der SDL-Prozesse nicht gewährleistet ist. Dieser Lösungsansatz kann aber nur dann verwendet werden, wenn ausschließlich die SDL-Prozesse, die innerhalb eines Betriebssystemprozesses realisiert sind, von dieser Maschine Requests anfordern.

Für den Fall, daß mehrere Einprozeß-Server-Modell-Implementierungen parallel existieren, die im Zeitscheibenverfahren vom Betriebssystem-Scheduler Zugang zum Zentralprozessor erhalten, versagt diese Modellierung, vgl. Abbildung 4-7.

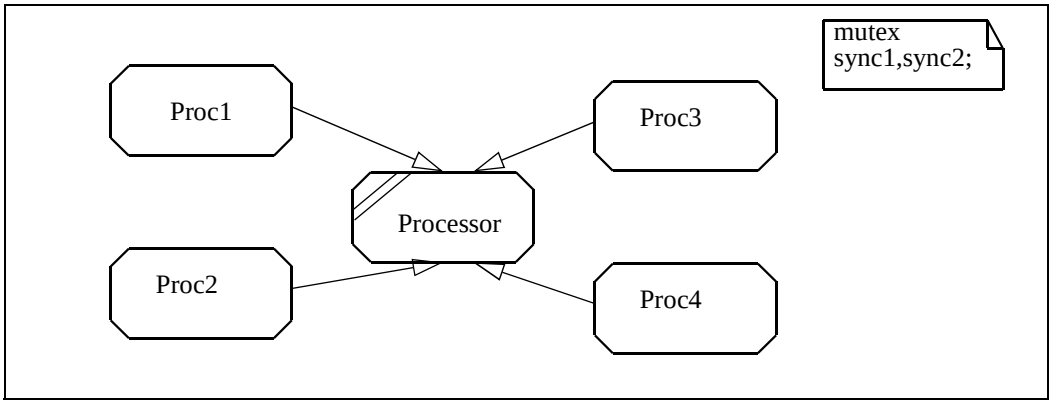


Abbildung 4-7 **Prozeßsynchronisation mit mutex**

Sowohl die SDL-Prozesse *Proc1* und *Proc2* als auch die SDL-Prozesse *Proc3* und *Proc4* sollen als Prozeduren innerhalb eines Betriebssystemprozesses implementiert werden. Die beiden Betriebssystemprozesse laufen quasi-parallel auf einem Prozessor. Ein solches Szenario wird auf der QSDL-Prozeßebene so abgebildet, daß jeder SDL-Prozeß einen Link zur QSDL-Maschine *Processor* enthält, über die die Requests an die Maschine gelangen. Damit der quasi-parallele Ablauf der beiden Betriebssystemprozesse korrekt nachgebildet werden kann, müßte die QSDL-Maschine als Bedienstrategie *Processor Sharing (PS)* erhalten. Das würde aber dazu führen, daß ebenfalls die SDL-Prozesse *Proc1* und *Proc2*, bzw. *Proc3* und *Proc4* quasi-parallel weiter fort-

schreiten würden, was aber durch ihre Art der Implementierung ausgeschlossen ist. Bei Verwendung der Scheduling-Disziplin FCFS wäre zwar die Serialisierung der als Prozeduren implementierten SDL-Prozesse korrekt nachgebildet, aber die quasi-parallele Abarbeitung der beiden Betriebssystemprozesse wäre nicht gewährleistet. An dieser Stelle kann das mutex-Konstrukt verwendet werden, um das Szenario korrekt zu modellieren.

Alle SDL-Prozesse, die als Prozeduren innerhalb eines Betriebsystemprozesses implementiert werden sollen, verwenden einen gemeinsamen mutex. Im obigen Beispiel verwenden die Prozesse *Proc1* und *Proc2* den mutex *sync1*, während die Prozesse *Proc3* und *Proc4* den mutex *sync2* verwenden. Alle Prozesse führen ihre Requests auf der gemeinsamen Maschine *Processor* aus, die als Processor-Sharing-Maschine von allen SDL- Prozessen parallel benutzt werden kann. Vor jeder Request-Aktion wird durch die Operation *acquire* zunächst die Zugangsberechtigung über den zugeordneten mutex geholt, siehe Abbildung 4-8. Versuchen zwei Prozesse, über einen mutex den parallelen Zugang zu erhalten, so wird der zweite Anfragende solange in der Warteschlange des mutex gepuffert, bis der vorherige diesen mit Hilfe der *release*-Operation wieder freigegeben hat.

Auf diese Weise wird dafür gesorgt, daß die SDL-Prozesse, die als Prozeduren innerhalb eines Betriebssystemprozesses realisiert werden, nicht parallel ihre zeitverbrauchenden Requests ausführen können, sondern korrekt serialisiert werden. Für die eigentlichen Betriebssystemprozesse ergeben sich keinerlei Einschränkungen bezüglich Parallelität. Für die Zuordnung von Aktionen in Transitionen zu den Werten, die als Bedienwunsch den Maschinendiensten mitgegeben werden, gelten die in Abschnitt 4.2.1 gemachten Bemerkungen. An dieser Stelle ist allerdings bei Input- bzw. Output-Aktionen darauf zu achten, ob sie als prozeßinterne Datenübergabe innerhalb eines Betriebssystemprozesses oder als Datenübergabe zwischen zwei unterschiedlichen Betriebssystemprozessen realisiert ist.

Proc1

```
State s1;
 Input a;
/*## acquire(sync1); ##*/
/*## Request ms1(amount);##*/
/*## release(sync1); ##*/
Output b;
nextstate s1;
```

Proc3

```
State s1;
 Input a;
/*## acquire(sync2); ##*/
/*## Request ms1(amount);##*/
/*## release(sync2); ##*/
Output b;
nextstate s1;
```

Proc2

```
State s1;
 Input a;
/*## acquire(sync1); ##*/
/*## Request ms1(amount);##*/
/*## release(sync1); ##*/
Output b;
nextstate s1;
```

Proc4

```
State s1;
 Input a;
/*## acquire(sync2); ##*/
/*## Request ms1(amount);##*/
/*## release(sync2); ##*/
Output b;
nextstate s1;
```

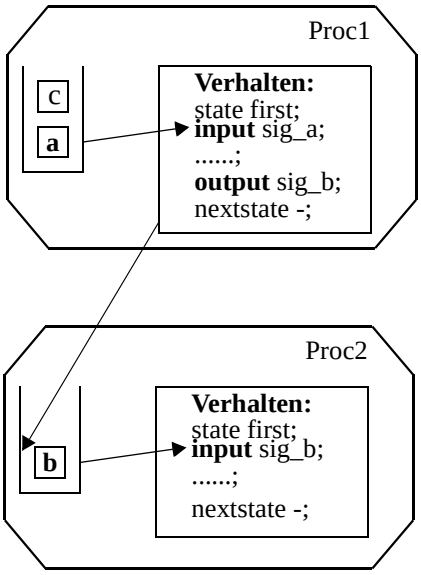
Abbildung 4-8 **Beispieltransitionen der Prozesse *Proc1* bis *Proc4***

4.2.4 Activity-Thread-Modell-Implementierung

Das Grundprinzip einer Implementierung nach dem Activity-Thread-Modell besteht darin, eine Protokollinstanz, also einen SDL-Prozeß als Menge von Prozeduren zu implementieren. Dabei kann je SDL-Prozeß eine Prozedur verwendet werden, aber auch je Transition eine separate Prozedur vorgesehen sein. Die Kommunikation zwischen den SDL-Prozessen wird durch Prozeduraufrufe realisiert, d.h. jedes SDL-Output führt zum Aufruf der entsprechenden Prozedur in der Zielinstanz. Durch diese Art der Implementierung kommt es zu einer Kette von Prozeduraufrufen, den sog. Activity Threads. Angestoßen werden die Aufrufketten durch externe Ereignisse an den Dienstschnittstellen der Instanzen. Durch die Prozeduraufrufe zur Abbildung der Kommunikation wird zu jedem Zeitpunkt genau ein externes Ereignis bearbeitet. Die Kommunikation zwischen den SDL-Prozessen wird synchronisiert, was jedoch dem Semantikmodell von SDL widerspricht. Daher sind Randbedingungen einzuhalten, wenn ein SDL-System nach dem Activity-Thread-Modell implementiert werden soll [56].

Um die SDL-Semantik bezüglich Atomizität von Transitionen und Behandlung von Mehrfachausgaben innerhalb einer SDL-Transition zu erfüllen, kann sowohl das Basic-Activity-Thread-Modell (BAT-Modell) als auch das Extended-Activity-Thread-Modell (EAT-Modell) zur Implementierung von SDL-Systemen angewendet werden [56], [57].

Im BAT-Modell wird das SDL-System auf einen einzelnen Betriebssystem-Thread abgebildet. Dieser Thread wird durch Ereignisse an der externen Schnittstelle aktiviert. Die Schnittstellen der BAT-Implementierung sind synchron, d.h. die parallele Abarbeitung zweier externer Ereignisse ist nicht möglich. Jeder SDL-Prozeß wird auf eine Prozedur abgebildet, wobei bei jedem Aufruf der Prozedur genau eine Transition des SDL-Prozesses zur Ausführung kommt. Darüberhinaus gibt es eine zentrale Datenstruktur, in der interne Variablen, Zustände, Adressen, etc. gespeichert werden.




```
Procedure BAT(external_signal);
  next_signal := external_signal;
  while not_empty(next_signal)
      call_next_procedure(next_signal);
End; /* BAT*/

Procedure proc1(next_signal);
 input_signal := next_signal;
if   (input_signal = sig_a) Then begin
       .....
      next_signal := sig_b; /*Output sig_b*/
fi;
End; /* proc1*/

Procedure proc2(next_signal);
input_signal := next_signal;
if   (input_signal = sig_b) Then begin
    ...........
fi;
End; /*proc2*/
```

Abbildung 4-9 **SDL-Spezifikation und Activity-Thread-Implementierung nach [57]**

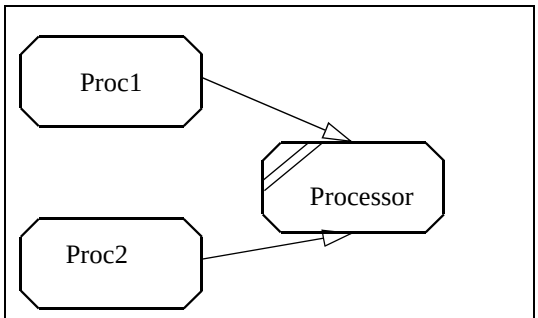
Die Abbildung eines SDL-Systems nach dem BAT-Modell ist allerdings nur dann semantikerhaltend möglich, wenn das System über genau einen Zugangspunkt externe Ereignisse in Form von SDL-Signalen entgegennimmt. Nur unter diesen Voraussetzungen sind die Eingangswarteschlan-

gen derjenigen Prozesse, die interne Signale empfangen, leer, wenn die Prozesse aktiviert werden. Somit ist die Umsetzung des SDL-Outputs in einen Prozeduraufruf auch mit der SDL-Semantik konform, siehe Abbildung 4-9.

Wäre ein zweiter äußerer Dienstzugangspunkt vorhanden, über den der SDL-Prozeß *Proc2* Signale entgegennehmen könnte, so würde das in seiner Warteschlange befindliche Signal durch die Umsetzung des SDL-Outputs als Prozeduraufruf von Signal b überholt. Dies kann jedoch nur durch Priority-Input oder durch die Verwendung des Save-Konstrukts erreicht werden.

In der dargestellten Implementierungsskizze wird das externe Ereignis *a* durch die Prozedur *BAT* entnommen. Diese Prozedur ruft die für dieses Ereignis definierte Prozedur *proc1* auf. Diese führt die definierten Aktionen aus und das SDL-Output des Signals *b* führt dazu, daß durch die umschließende BAT-Prozedur die für das interne Ereignis *sig_b* zuständige Prozedur *proc2* aufgerufen wird. Nach vollständiger Abarbeitung der Prozedur *proc2* kehrt die Kontrolle zur zentralen Entnahmestelle zurück und ein weiteres externes Ereignis, z.B. Eingabe von *sig_c* kann verarbeitet werden.

Die Performance-Spezifikation in QSDL des nach dem BAT-Modell implementierten SDL-Systems hat eine Struktur entsprechend Abbildung 4-10. Es existiert eine QSDL-Maschine für beide SDL-Prozesse, die mit der Bedienstrategie *FCFSPP* (FCFS prior preemptive resume) arbeitet. Diese prioritätsgesteuerte und mit Unterbrechungen arbeitende Maschine wird mit einem Server ausgestattet. Damit wird der Tatsache Rechnung getragen, daß beide SDL-Prozesse innerhalb eines Betriebssystem-Threads laufen. Die Bedienstrategie *FCFSPP* wird benötigt, um die Blockierung des Prozesses *Proc1* zu erreichen, der ja erst dann wieder Signale aus der Warteschlange entnehmen kann, wenn der mit dem aktuellen Ereignis verbundene Activity-Thread abgearbeitet ist.




```
/*## Machineservice
  compute(Real,Natural); ##*/
/*## Machine processor;
Server: 1;
Discipline: FCFSPP;
Offers compute 200000;
Endmachine; ##*/
/*## Link prc1 from Proc1 to
      Processor with compute; ##*/
/*## Link prc2 from Proc2 to
      Processor with compute; ##*/
```

Process Proc1 (Ausschnitt)

```
State first;
Input sig_a;
/*## Request compute(amount_input,0); ##*/
.....
/*## Request compute(amount_other,0); ##*/
Output b;
/*## Request compute(amount_return,0); ##*/
nextstate -;
```

Process Proc2 (Ausschnitt)

```
State first;
Input sig_b;
/*## Request compute(amount_input,1); ##*/
.....
/*## Request compute(amount_other,1); ##*/
nextstate -;
```

Abbildung 4-10 **QSDL-Struktur und Verhalten des BAT-Modells**

Zunächst entnimmt der Prozeß *Proc1* das erste Signal aus der Warteschlange. Die Zeit, die dazu benötigt wird, modelliert der erste Request. Dann werden die internen Aktionen des Prozesse abgearbeitet, was mit Hilfe des zweiten Requests modelliert wird. In dem Wert für *amount_other* ist ebenfalls der minimale Aufwand enthalten, der durch die Umsetzung des SDL-Outputs als Prozeduraufruf anfällt. Das dann ausgeführte SDL-Output führt dazu, daß auch der Prozeß *Proc2* aktiviert wird.

Während *Proc1* durch den letzten Request, der den Aufwand für die Kontrollrückgabe modelliert, daran gehindert wird, bereits das nächste externe Ereignis zu bearbeiten, führt der SDL-Prozeß *Proc2* den ersten Request zur Übernahme der Daten aus. Da dieser Request eine höhere Priorität hat, als der von Prozeß *Proc1* und die Maschine *Processor* mit unterbrechender Bedienstrategie arbeitet, wird *Proc1* solange unterbrochen, bis die von *Proc2* ausgeführten Aktionen vollständig abgearbeitet sind. Erst dann kann der Prozeß *Proc1* seinen Request zu Ende ausführen und dann das Ereignis *Eingabe sig_c* bearbeiten.

Somit entspricht das modellierte Verhalten dem implementierten, und die konkreten Werte für die benötigten Eingabeparameter können durch Instrumentierung von Prototypen oder durch Erfahrungswerte aus anderen Projekten ermittelt werden.

Für Implementierungen, bei denen mehrere Activity-Threads quasi-parallel oder bei Ablauf auf einem Mehrprozessorsystem parallel ablaufen, bzw. bei der Implementierung von SDL-Systemen, bei denen Prozeßmengen gemeinsam implementiert werden, die über mehr als eine Schnittstelle für externe Ereignisse verfügen, kann das BAT-Modell nicht verwendet werden. Statt dessen kann das EAT-Modell eingesetzt werden, bei dem für jedes anliegende externe Signal ein Betriebssystem-Thread erzeugt wird [57]. Damit können viele Activity-Threads parallel ausgeführt werden. Bei der Implementierung ist darauf zu achten, daß Prozeduraufrufe für identische Prozeduren, die gleichzeitig stattfinden über Semaphore zwangsweise serialisiert werden. Das ist jedoch bei der Ausführung von SDL-Systemen standardmäßig der Fall, da immer nur eine Transition je SDL-Prozeß zur Zeit ausgeführt wird. Somit läßt sich für eine Implementierung nach dem EAT-Modell sehr einfach ein QSDL-Modell angeben. Es wird für alle in das EAT-Modell übernommenen Prozesse eine QSDL-Maschine mit Bedienstrategie *PS* bereitgestellt. Diese kann analog zur gewählten Implementierungsmaschine aus einem oder mehreren Prozessoren bestehen. Die “Kosten” für die Ausführung der angestoßenen Transitionen können direkt in den QSDL-Prozeß eingetragen werden. Die Wartezeiten, die sich möglicherweise durch die Synchronisation über die Semaphore in der Implementierung ergeben, werden im QSDL-System automatisch durch die Wartezeiten in den Eingabewarteschlangen der QSDL-Prozesse modelliert. Diese entstehen immer dann, wenn der Prozeß noch einen Request ausführt, der durch einen parallel ablaufenden Activity-Thread entstanden ist.

Damit sind für die wichtigsten Implementierungsansätze von Kommunikationsprotokollen Modellierungsbausteine entwickelt und vorgestellt worden. Im folgenden werden noch einige spezielle Probleme im Kontext von Protokollimplementierung und Performance aufgegriffen.

4.3 Parallelität in Protokollen

Die Leistungsfähigkeit von Kommunikationsprotokollen hängt nicht alleine von der gewählten Implementierungsarchitektur ab. Vielmehr ergeben sich potentielle Leistungsreserven dadurch, daß die in den Protokollabläufen vorhandene Parallelität ausgenutzt wird, um nebenläufig ausführbare Soft- und Hardware-Strukturen abzuleiten.

Dabei werden mehrere Formen der Parallelität vorgestellt, die sich durch die Aufteilung des Gesamtsystems in parallel ausführbare Einheiten unterscheiden [113], [116]. Dabei haben sich die folgenden Parallelitätsmodelle herausgebildet:

- Schichtenbezogene Parallelität
- Verbindungsbezogene Parallelität
- Richtungsbezogene Parallelität
- Nachrichten- (PDU)- bezogene Parallelität
- Funktionsbezogene Parallelität

Bei *schichtenbezogener Parallelität* bilden die Protokollschichten gemäß des ISO/OSI-Modells die parallel ausführbaren Einheiten, während bei *verbindungsbezogener Parallelität* jede Verbindung als parallel ausführbare Einheit angesehen wird. Dabei kann schichtenübergreifend parallelisiert werden. Außerdem ist die letztere Form an die Existenz einer logischen Verbindung geknüpft. Bei *richtungsbezogener Parallelität* wird die Bearbeitung von Sende- und Empfangs-PDUs (protocol data units) durch getrennte Einheiten vorgenommen.

Bei *nachrichtenbezogener Parallelität* existieren eine Reihe von (identischen) Einheiten, die eingehende Nachrichten unabhängig von ihrer Zugehörigkeit zu bestimmten Verbindungen, bearbeiten können. Diese Form der Parallelisierung eignet sich besonders gut für Datagram-Dienste wie zum Beispiel IP, da zwischen je zwei Nachrichten keinerlei Bezug existiert.

Die aufwendigste Form der Parallelisierung, die aber auch den größten Performance-Gewinn verspricht, stellt die *funktionsbezogene Parallelisierung* dar. Hierbei werden für alle benötigten Protokollfunktionen spezialisierte Einheiten bereitgestellt, die diese Funktionalitäten allen Verbindungen und Schichten zur Verfügung stellen. Davon verspricht man sich vor allem, daß unabhängige Aufgaben, z.B. die Funktionen *Prüfsummenberechnung* und *Ermittlung des Nachfolgeknotens* in einer Instanz der Netzwerkschicht, parallel ausgeführt werden können. Außerdem kann mit diesem Ansatz die Ausführung gleicher Funktionen in unterschiedlichen Schichten vermieden werden [116]. Funktionsbezogene Zerlegung von Protokollen ist in der Regel nicht trivial, da sich keine allgemeingültigen Ableitungsregeln angeben lassen. Statt dessen sollen parallele Strukturen bereits in der Entwurfsphase berücksichtigt werden [113].

Für die oben vorgestellten Parallelisierungsansätze werden nun im folgenden QSDL-Bausteine entwickelt, mit deren Hilfe die Auswirkungen auf die Performance der Protokollausführung untersucht werden können.

Alle potentiell nebenläufig ausführbaren Einheiten müssen auf ebensolche QSDL-Einheiten abgebildet werden. Daher ist darauf zu achten, daß diese Einheiten oder Instanzen auf einen (Q)SDL-Prozeß oder eine Menge von (Q)SDL-Prozessen abgebildet werden, da nur diese in einem (Q)SDL-System parallel fortschreiten können. Damit ist ein Grad maximaler Parallelität möglich, der durch die Zuordnung von (Q)SDL-Prozessen zu QSDL-Maschinen entsprechend der Implementierungsentscheidungen eingeschränkt werden kann.

4.3.1 Modelle für schichtenbezogene Parallelität

Implementierung vollständiger Parallelität

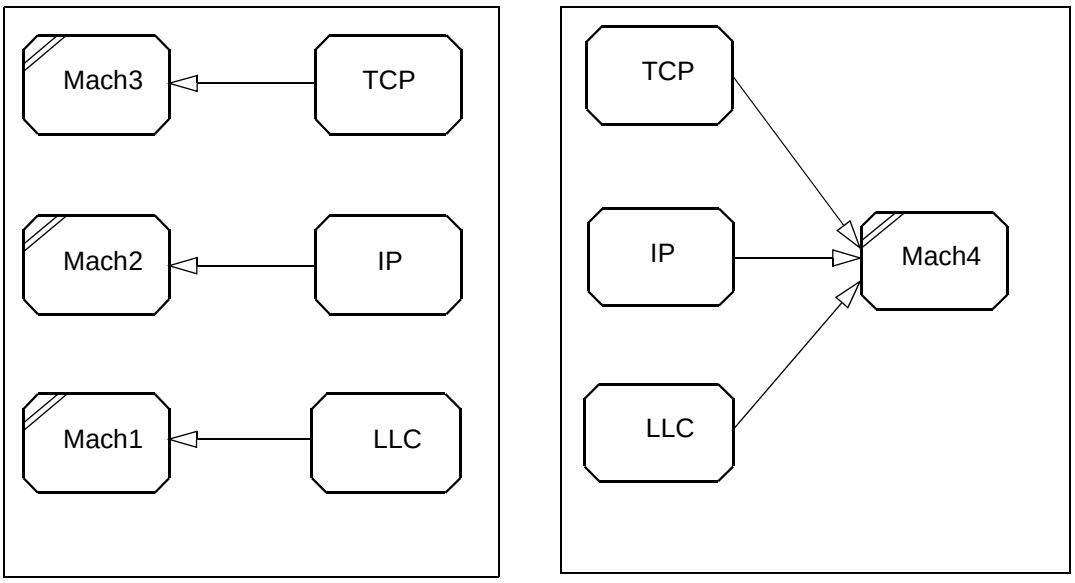
Damit Parallelisierung in vollem Umfang möglich ist, müssen alle potentiell parallelen Aktionen in unterschiedlichen SDL-Prozessen spezifiziert werden. Andernfalls schränkt man bereits durch

die Spezifikation parallele Abläufe ein. Durch die jeweilige Implementierung kann nachfolgend allerdings die Parallelität noch eingeschränkt werden.

Damit Parallelität uneingeschränkt möglich ist, sollten die Instanzen gemäß des Mehrprozeß-Server-Modells entweder mit separaten Betriebssystemprozessen oder mit Threads implementiert werden. Die Prozesse können dann auf dedizierten Prozessoren zum Ablauf gebracht werden, was einer Modellstruktur gemäß Abbildung 4-11a) ergeben würde. Die in der Abbildung dargestellten Maschinen *Mach1* bis *Mach3* können als FCFS-Maschinen, als PS-Maschinen oder als IS-Maschinen spezifiziert werden. Da sie jeweils von einem QSDL-Prozeß exklusiv benutzt werden, ergeben sich bei allen Varianten identische Performance-Ergebnisse. Alternativ wäre eine Struktur gemäß Abbildung 4-11b) möglich. Dabei müßte die spezifizierte Maschine aber für jeden mit ihr verbundenen QSDL-Prozeß einen Bediener haben, oder als IS-Maschine spezifiziert werden.

Implementierung eingeschränkter Parallelität

Für den Fall, daß nicht für jede Protokollinstanz ein Prozessor exklusiv zur Verfügung steht, ergibt sich ein Szenario gemäß Abbildung 4-11b). Für den Fall, daß die implementierten Instanzen als separate Betriebssystemprozesse oder als Threads realisiert sind, ist die Maschine *Mach4* mit der Scheduling-Strategie Processor Sharing (PS) zu spezifizieren.



a) echte Parallelität **b) Quasi- Parallelität**

Abbildung 4-11 **Schichtenbezogene Parallelität**

Implementierung vollständiger Serialität

Falls die Instanzen als Einprozeß-Server Implementierung realisiert sind, muß die Maschine mit der Strategie FCFS spezifiziert werden. Dabei werden diese allerdings zwangsweise an der Maschine serialisiert, was aber durch die Implementierung bedingt ist und sich daher auch im QSDL-Modell wiederfinden muß.

4.3.2 Modelle für verbindungsbezogene Parallelität

Die Ausnutzung verbindungsbezogener Parallelität bietet sich vor allem in Transportschichten oder bei der Implementierung von Server-Prozessen an. Das folgende Beispiel einer Spezifikation des *Transmission Control Protocols* (TCP) erläutert die Vorgehensweise bei der Analyse verbindungsbezogener Parallelität. In Abbildung 4-12 ist ein Ausschnitt aus dem TCP/IP-Protokollsystem dargestellt. Dabei enthält der SDL-Prozeß *TCP* den Automaten entsprechend RFC793 [88]. Der Prozeß wird vom SDL-Prozeß *Port_Generator* immer dann erzeugt, wenn eine neue TCP-Verbindung angefordert wird. Somit wickelt jeder der SDL-Prozesse *TCP* genau eine Verbindung ab. Detaillierte Angaben zu diesem Szenario sind in [60] veröffentlicht worden. Die Anzahl der im System aktiven TCP-Verbindungen kann von Null bis zur oberen Schranke für die maximale Anzahl der gleichzeitig zugelassenen TCP-Verbindungen *(nocon)* variieren.

Für dieses Szenario bieten sich einige Varianten zur Implementierung und somit zur Modellierung an. Falls die Anzahl der maximal zugelassenen Verbindungen kleiner ist als die Anzahl der Prozessoren auf dem Zielsystem, kann die Maschine *Mach* wahlweise mit der Bedienstrategie FCFS oder PS spezifiziert werden. Die Anzahl der Bediener muß dann mit dem entsprechenden Wert angegeben werden. Somit ist gewährleistet, daß jede TCP-Verbindung genau einen Prozessor hat und sich die Verbindungen nicht gegenseitig stören. Die Verwendung einer Maschine mit der Bedienstrategie IS würde ebenfalls dafür sorgen, daß die Verbindungen unbeeinflußt voneinander bedient werden. Allerdings ist dieser Ansatz nicht so naheliegend wie die Verwendung einer Mehrprozessor-Maschine.

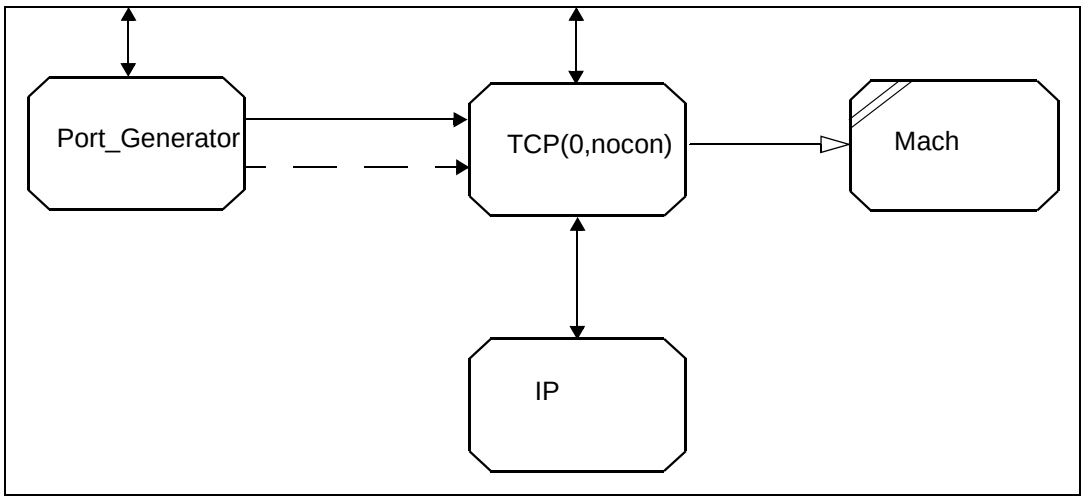


Abbildung 4-12 **Verbindungsbezogene Parallelität**

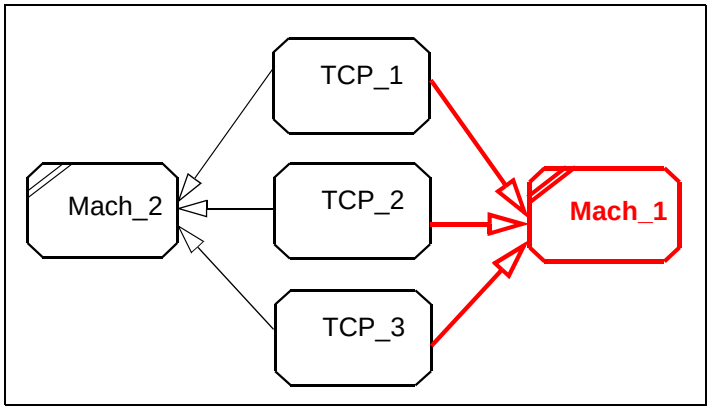
Falls eine feste Zuordnung der Instanzen zu den vorhandenen Prozessoren nicht möglich ist, oder weniger Prozessoren als parallele Verbindungen vorhanden sind, wird die Zuteilung von Prozessoren zur Laufzeit durch den Betriebssystem-Scheduler vorgenommen. Ein solches Szenario wird unter Verwendung der QSDL-Maschine *Mach* mit der Bedienstrategie PS modelliert, womit das parallele Fortschreiten aller aktiven Verbindungen ebenfalls möglich ist. Allerdings wird dies nur unter reduzierter Leistung möglich sein.

Müssen die Instanzen zur Abwicklung der Verbindungen auf gemeinsame, exklusive Bereiche zugreifen, so kann dies entweder mit Hilfe des QSDL-mutex-Konzeptes realisiert werden, vgl.

Abschnitt 4.2.3 oder unter Verwendung einer zweiten QSDL-Maschine, die mit einem Prozessor und der Bedienstrategie FCFS spezifiziert werden muß.

Wird der mutex-Ansatz benutzt, so müssen die zeitverbrauchenden Dienstaufrufe durch die Operationen *acquire* und *release* eingekapselt werden, da diese ja von der exklusiv dem Prozeß zugewiesenen QSDL-Maschine ausgeführt werden.

Bei Benutzung der separaten Maschine wird der den Zugriff modellierende Request vor der Ausführung der Zugriffsoperationen von der FCFS-Maschine bedient, vgl. Abbildung 4-13.




```
/*## Machineservice
      compute(real),
      retrieval(real);   ##*/
/*## Machine Mach1;
      Server 1;
      Discipline: FCFS;
      Offers retrieval: 5.0;
      Endmachine;    ##*/
 /*## Machine Mach2;;
       Server 3;
       Discipline: FCFS;
       Offers compute: 25.0;
       Endmachine;    ##*/
process TCP_1 referenced;
process TCP_2 referenced;
process TCP_3 referenced;
mutex retrieval_sync;

/*##
link L1 from TCP_1 to Mach1
with retrieval; ##*/
/*##
link L2 from TCP_1 to Mach2
with compute; ##*/
/*##
link L3 from TCP_2 to Mach1
with retrieval; ##*/
/*##
link L4 from TCP_2 to Mach2
with compute; ##*/
/*##
link L5 from TCP_3 to Mach1
with retrieval; ##*/
/*##
link L6 from TCP_3 to Mach2
with compute; ##*/
```

```
Process TCP_1;  (Ausschnitt)
State z1;
Input pdu;
/*## Request compute(compute_amount); ##*/
Task 'Aktionen für PDU-Analyse'
/*## acquire(retrieval_sync); ##*/
/*## Request compute(retrieval_amount); ##*/
/*## release(retrieval_sync); ##*/
/*## Request retrieval(retrieval_amount); ##*/
Task 'Aktionen für Retrieval'
Output pdu;
Nextstate z2;
Endprocess;
```

Abbildung 4-13 **Verbindungsbezogene Parallelität mit Synchronisation**

Die in Standardschrift angegebenen QSDL-Konstrukte gelten für beide Modellierungsvarianten, während die in Fettschrift angegebenen Elemente für das Modell mit separater QSDL-Maschine gelten. Analog dazu gelten die in Kursivschrift angegebenen Konstrukte nur für den mutex-Ansatz.

In Abbildung 4-13 oben links ist die Prozeß- und Maschinenstruktur des QSDL-Modells dargestellt. Die drei TCP-Prozesse werden parallel auf die QSDL-Maschine *Mach2* abgebildet. Da diese mit drei Bedienern ausgestattet ist, können die Prozesse echt parallel fortschreiten. Die Spezifikation der TCP-Prozesse, siehe Abbildung 4-13 unten links, legt fest, daß bei Eingang einer PDU Aktionen zur PDU-Analyse und Aktionen zur Informationsbeschaffung ausgeführt werden müssen.

Während für die Aktionen zur PDU-Analyse alle Informationen prozeßlokal sein sollen, benötigt die Instanz für die anderen Aktionen Informationen aus einem Bereich, auf den alle Instanzen gemeinsam aber jeweils exklusiv zugreifen können. Während bei Verwendung des mutex-Ansatzes die benötigte Zeit über die gleiche Maschine modelliert wird, die auch schon den Aufwand für die PDU-Analyse modelliert und die Synchronisation der Prozesse über den mutex *retrieval_sync* ausgeführt wird, übernimmt die FCFS-Maschine *mach1* im zweiten Ansatz sowohl die Synchronisation der Prozesse als auch die Modellierung der benötigten Zeit. In beiden Fällen wird die Synchronisation der Prozesse zeitgetreu nachgebildet, und die Bearbeitungszeit für die gesamte SDL-Transition ändert sich bei gleichen Bediengeschwindigkeiten der Maschinen nicht. Allerdings werden mit den beiden Ansätzen unterschiedliche Maschinenauslastungen für die Maschine *Mach_2* erzielt, so daß die Auswahl des Modellierungsansatzes unter anderem auch von den gewünschten Ergebnisgrößen abhängig ist.

4.3.3 Modelle für richtungsbezogene Parallelität

Bei richtungsbezogener Parallelität sind in den einzelnen Protokollinstanzen parallel agierende Einheiten für das Senden und das Empfangen von PDUs vorzusehen. Diese Art der Parallelität findet man vor allem in den tieferen Schichten des ISO-OSI-Modells, z.B. beim Medienzugang in der Schicht 2, siehe Abschnitt 6.2.1. Daher müssen die Sende- und Empfangsoperationen als separate SDL-Prozesse spezifiziert und analog zum Vorgehen bei verbindungsorientierter Parallelität an separate QSDL-Maschinen gebunden werden. Die Modellierung von Synchronisationsaspekten zwischen Senden und Empfangen von PDUs kann analog über mutex oder mit Hilfe separater QSDL-Maschinen mit Bedienstrategie FCFS geschehen.

4.3.4 Modelle für nachrichtenbezogene Parallelität

Nachrichtenbezogene Parallelität sieht die Bearbeitung gleichzeitig vorhandener PDUs auf verschiedenen Prozessoren des Systems vor. Es werden auf jedem Prozessor Replikationen der implementierten Instanz abgelegt. Dazu ist allerdings ein gemeinsamer Eingangsbereich notwendig, aus dem die Instanzen die eingehenden PDUs entnehmen, vgl. Abbildung 4-14.

Faßt man die implementierten Instanzen als Realisierung von SDL-Prozessen auf, so besteht das Problem, daß SDL-Prozesse keinen gemeinsamen Eingangswarteschlangen für Signale haben. So muß bereits die sendende Instanz entscheiden, welcher der vorhandenen SDL-Prozesse die gesendeten Signale empfängt. Ließe man diese Entscheidung offen, d.h. bekommen alle Output-Aktionen des Prozesses *Schicht_I-Instanz* keinerlei Adressateninformation durch Verwendung des SDL-via bzw. das SDL-to-Konstrukts, so wäre laut SDL-Semantik der Empfänger unbestimmt und kann willkürlich aus der Menge der potentiellen Empfänger ausgewählt werden. Somit läge eine Unterspezifikation vor und eine sinnvolle Performance-Analyse ist nur dann möglich, wenn der Nichtdeterminismus bei der Auswahl der Zielinstanz aufgelöst würde. So könnte zum Beispiel festgelegt werden, daß aufeinanderfolgende PDUs immer abwechselnd auf die beiden vorhandenen Instanzen verteilt werden.

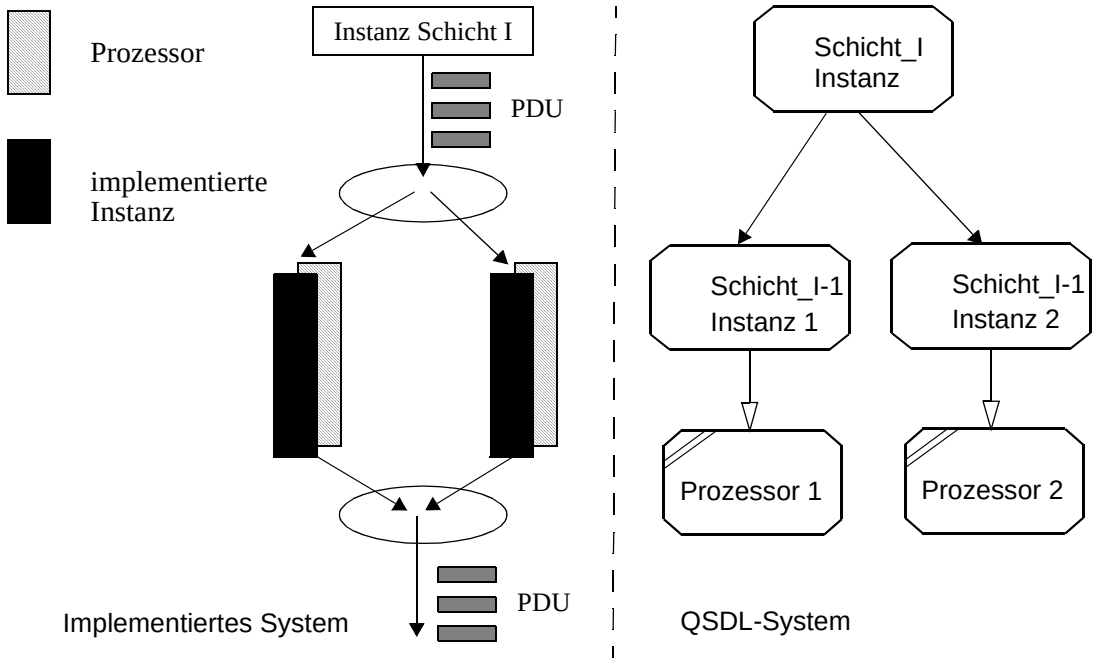


Abbildung 4-14 **Nachrichtenbezogene Parallelität**

Die beiden SDL-Prozesse *Schicht_I-1_instanz1* und *Schicht_I-1_instanz2* werden jeweils separaten QSDL-Maschinen zugeordnet, die die Ausführung der Instanzen auf den beiden Prozessoren modellieren, vgl. Abbildung 4-14 links.

4.3.5 Modelle für funktionsbezogene Parallelität

Die Idee einer funktionsbezogenen Dekomposition von Kommunikationsprotokollen geht auf M. Zitterbart [116] zurück. Dieser Ansatz entstand aus der Erkenntnis heraus, daß das hierarchisch gegliederte ISO/OSI-Kommunikationsmodell den Herausforderungen neuer multimedialer Anwendungen nicht gewachsen ist. Dabei wird der Entwurf völlig neuer Protokoll- und Dienstarchitekturen angeregt und insbesondere “eine Integration von Implementierungsaspekten bereits beim Entwurf solcher Systeme” gefordert[b].

Der Ansatz basiert auf der Zerlegung einer Instanz bzw. eines ganzen Protokollturmes in eine Menge von Protokollfunktionen. Jede Protokollfunktion wird als atomare Einheit mit Ein- und Ausgängen aufgefaßt. Nur solche Protokollfunktionen, zwischen denen keinerlei Abhängigkeiten bestehen, können parallel ausgeführt werden. Die Aufteilung eines Protokollturmes in eine Menge von Protokollfunktionen soll an dieser Stelle nicht durchgeführt werden. Eine solche Zerlegung des *Transmission Control Protocols* (TCP) wird in [116] vorgestellt.

Es wird hier davon ausgegangen, daß eine funktionsbasierte Zerlegung durchgeführt wurde und die identifizierten Protokollfunktionen als SDL-Prozesse spezifiziert worden sind. Somit kann eine Zuordnung von SDL-Prozessen auf QSDL-Maschinen genauso vorgenommen werden, wie dies in den Fällen der schichtenbezogenen, richtungsbezogenen oder verbindungsbezogenen Par-

b. vgl. [116], Seite 72/73

allelisierung durchgeführt wurde. Der Unterschied besteht lediglich darin, daß die SDL-Prozesse kleinere Funktionseinheiten beschreiben. Die funktionalen Abhängigkeiten werden durch das SDL-System bereits beschrieben.

Nachdem Modellbausteine für wichtige Implementierungs- und Parallelisierungsansätze vorgestellt worden sind und damit die Spezifikation zeitlichen Verhaltens eines SDL-Systems demonstriert wurde, wird nun im folgenden der Schwerpunkt auf Analysetechniken unter Zuhilfenahme des Werkzeugs QUEST gelegt. Dabei wird exemplarisch gezeigt, wie das Werkzeug dazu verwendet werden kann, das zeitliche Verhalten eines SDL-Systems zu beobachten, um so das Verständnis für das System unter Berücksichtigung zeitlicher Aspekte zu erhöhen.

4.4 Ursache-Wirkungsanalysen mit QUEST

Standen in den Abschnitten 4.1 und 4.2 Implementierungsstrategien und deren Modellierung im Zentrum der methodischen Überlegungen, gilt es nun, die Analysemethoden des Werkzeugs QUEST dazu einzusetzen, spezifische Performance-Probleme zu entdecken und gegebenenfalls Änderungsvorschläge abzuleiten, die zu einer verbesserten Protokoll-Performance führen.

Die QUEST-Methode ermöglicht durch die Simulation des in QSDL spezifizierten Systems im Zusammenspiel mit der Online-Beobachtung des Modells eine Reihe von Analysen, die mit klassischen Performance-Modellen wie Warteschlangensystemen, zeitbehafteten Petri-Netzen, etc. nicht möglich sind. So können in einem QSDL-System aggregierte Leistungsmaße in ihrer Entwicklung und damit in ihrem transienten Verhalten beobachtet werden. Dadurch können zum Beispiel die Auswirkungen von plötzlich auftretenden Lastspitzen ebenso beobachtet werden, wie die Auswirkungen ungenügender Leistungsfähigkeit von Komponenten oder die Auswirkungen kurzzeitiger Ausfälle von (Teil-) Systemen.

In einem ersten Beispiel werden die Auswirkungen plötzlich auftretender Lastspitzen auf ein Übertragungssystem untersucht, das aus zwei Protokollinstanzen besteht, die über ein unzuverlässiges Medium verbunden sind, vgl. Abbildung 4-15. Die beiden Protokollinstanzen Protinst1 und Protinst2 verwenden jeweils den Sliding-Window-Mechanismus *Selective Repeat* zur Flußkontrolle und Fehlerbehebung. Die Funktion beider Mechanismen kann im Detail bei A. Tanenbaum in [108] nachgelesen werden. Die Protokollfunktionalität ist vollständig in SDL beschrieben worden und wird hier nur kurz zusammengefaßt[c].

Die Nutzer des von den Protokollinstanzen bereitgestellten Dienstes können durch das Dienstprimitiv *UReq* die zuverlässige aber unbestätigte Übertragung von Dateneinheiten zwischen direkt verbundenen Einheiten anfordern. Die Datenreihenfolge wird dabei eingehalten. Korrekt übertragene Dateneinheiten werden durch Verwendung des Dienstprimitives *UInd* dem Empfänger angezeigt. Die Protokollinstanzen erbringen den Dienst unter Verwendung eines unzuverlässigen Dienstes, der Datenpakete verfälschen oder verlieren kann.

Jede Protokollinstanz hat einen Puffer, in dem eingehende Daten bis zu ihrer Verarbeitung gespeichert werden und einen Puffer, in dem ausgehende Pakete solange gespeichert werden, bis der korrekte Empfang des Paketes von der Empfängerseite bestätigt wurde. Falls ein Paket während der Übertragung verloren geht oder die Bestätigung nicht vor Ablauf eines Timers bei der Sendeinstanz eingeht, wird eine erneute Übertragung des Paketes veranlaßt. Korrekt übertragene Pakete werden per *Piggy-Backing* bestätigt, falls Daten für die Rückrichtung vorhanden sind. Sind keine Daten vorhanden, werden separate Bestätigungspakete gesendet.

c. Die vollständige SDL-Spezifikation befindet sich im Anhang zu dieser Arbeit.

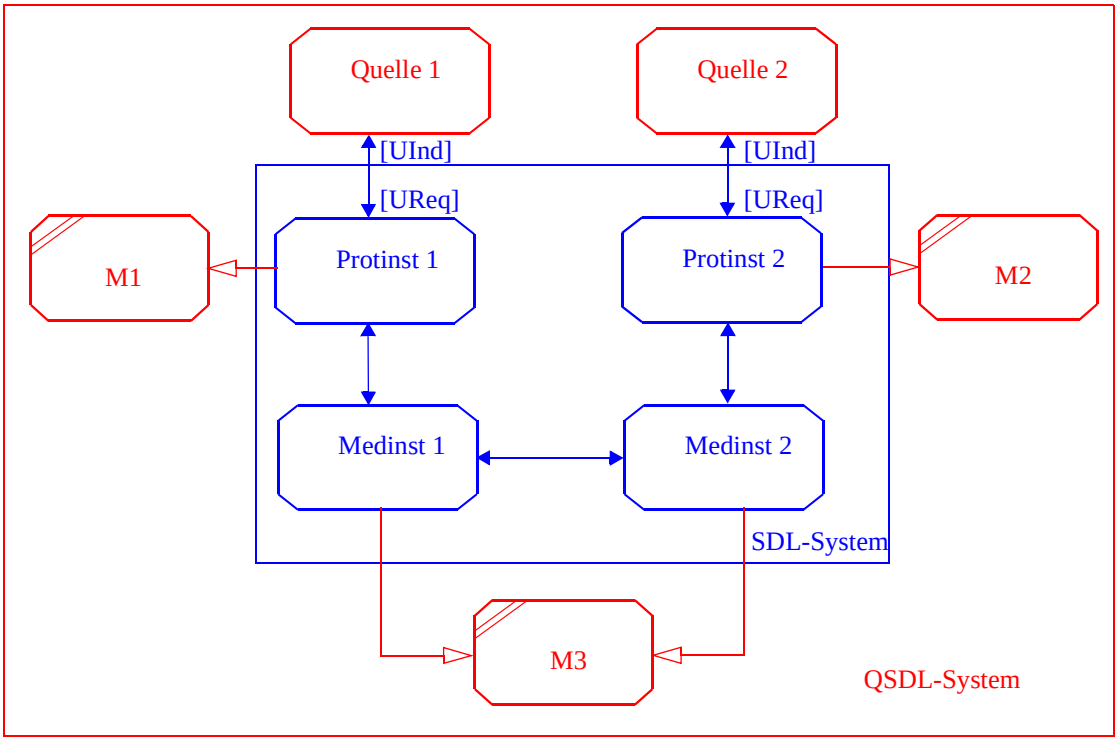


Abbildung 4-15 **Protokollsystem Selective Repeat, nach Tanenbaum [108]**

Die Protokollinstanzen übergeben die empfangenen Datenpakete an die oberen Dienstnehmer in der Reihenfolge, in der sie abgesendet wurden, wenn alle bisher versendeten Datenpakete lükkenlos vorliegen. Jede Instanz hat ein Limit *w* (*window size*) für gesendete, unbestätigte Pakete. Wird diese obere Grenze erreicht, können erst wieder Datenpakete gesendet werden, wenn Bestätigungen eingehen.

Für die Performance-Untersuchungen werden je eine Maschine für die beiden Protokollinstanzen und eine Maschine für die Datenübertragung auf dem Verbindungsmedium definiert. Die Maschinen *M1* und *M2* modellieren die Zeitverbräuche für alle Aktionen, die von den Protokollinstanzen zur Bearbeitung von PDUs durchgeführt werden müssen. Die Definition separater Maschinen für jede Protokollinstanz trägt der Tatsache Rechnung, daß die Instanzen auf separaten Rechnern installiert sind.

Die Maschine *M3* modelliert den Zeitverbrauch, der durch die Übertragung der Pakete auf dem physikalischen Medium benötigt wird. Im vorliegenden Fall wird vereinfachend angenommen, daß es sich bei dem Medium um einen Halbduplexkanal handelt. Zusätzlich zu diesen Ergänzungen werden noch zwei QSDL-Prozesse definiert, die das Benutzerverhalten modellieren. Hierbei handelt es sich um erweiterte SDL-Prozesse, die nach einem genau festgelegten Muster, die Dienstprimitive *Ureq* aufrufen, die an der oberen Dienstschnittstelle zur Verfügung stehen, und so die Belastung der Protokollinstanzen und des Mediums veranlassen.

Im vorliegenden Szenario sind die Quellprozesse als sog. *On-Off-Quellen* spezifiziert worden. Zwischen den beiden Zuständen *On* und *Off* wechselt die Quelle ständig hin und her. Während sie im Zustand *Off* keinerlei Signale versendet, sendet sie im Zustand *On* mit der maximalen Rate.

Folgende Parameter beschreiben die quantitativen Merkmale einer *On-Off-Quelle*:

1. Senderate μ [Pakete/s]
2. Übergangsrate in den Zustand *On* μ_{on} [1/s]
3. Übergangsrate in den Zustand *Off* μ_{off} [1/s]

Das Verhältnis von $\frac{\mu_{on}}{\mu_{off}}$ legt die Aufenthaltswahrscheinlichkeiten für die beiden Zustände fest. Damit wird das Verhältnis von durchschnittlicher zu maximaler Aktivität festgelegt. Dieses Verhältnis wird auch als Burst-Faktor bezeichnet. Mit Hilfe des QSDL-Systems kann nun die Auswirkung von Lastspitzen, d.h. eine hohe Anzahl von Sendeereignissen, auf das Protokollsystem untersucht werden. Mit Hilfe von QSDL-Sensoren kann nicht nur das transiente Verhalten der Quellen zur Laufzeit des Simulators angezeigt werden, sondern auch die Entwicklung der Wartezeiten von SDL-Signalen in den Warteschlangen der SDL-Prozesse visualisiert werden. Darüberhinaus lassen sich die Auslastungen der QSDL-Maschinen visualisieren, so daß man erkennen kann, welche Maschine für die lange Blockierung der SDL-Prozesse verantwortlich ist, siehe Abbildung 4-16.

So erkennt man, daß die Lastspitze der Quelle 2 (*Source 2*) zunächst zu einem Anstieg der Wartezeiten an dem Prozeß *Medinst2* führt, siehe Abbildung 4-16, rechts Oben und Mitte. Die später einsetzende Aktivität des Prozesses *Source1* führt nicht nur zu einem Anstieg der Wartezeiten am Prozeß *Medinst1*, sondern auch zu einer Belastung der Maschine *M3*, die die Übertragung der Datenpakete auf dem Medium modelliert. Diese Belastung setzt allerdings erst mit einer Verzögerung von ca. 3 s ein, siehe Abbildung 4-16 unten links.

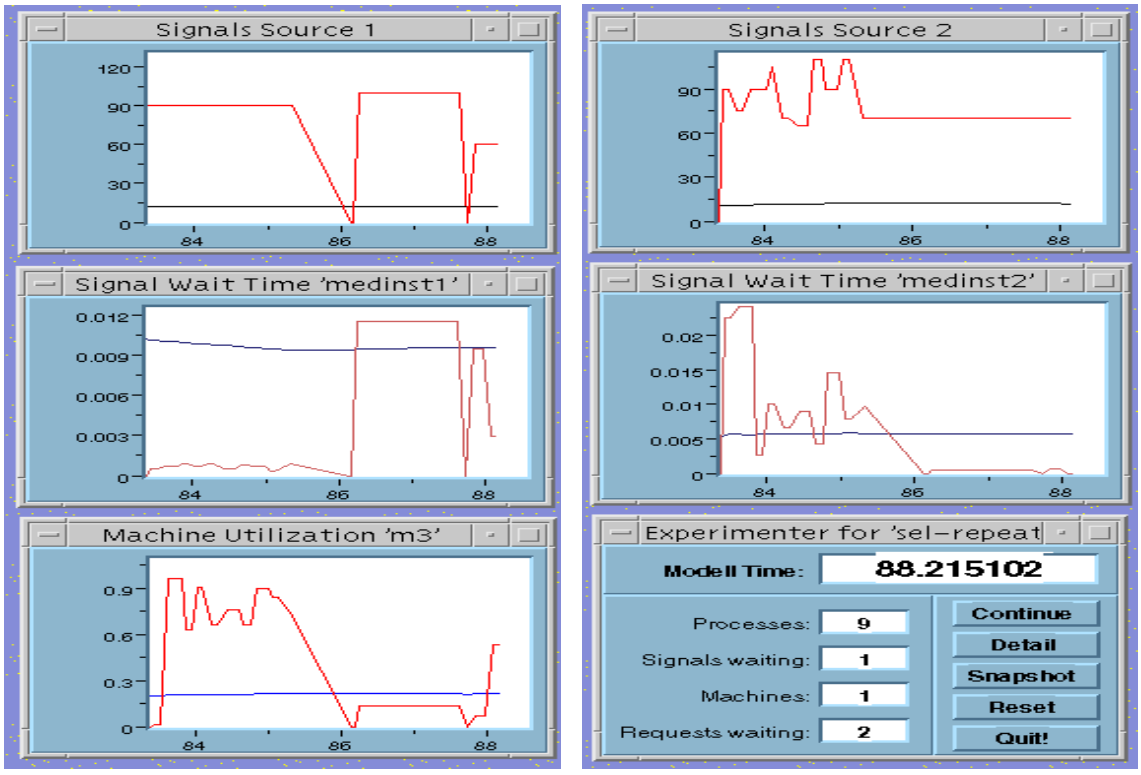


Abbildung 4-16 **Kurzzeitverhalten von Lastquellen und dessen Auswirkungen**

Insbesondere kann mit Hilfe der QUEST-Methode überprüft werden, wieviel Zeit das Protokollsystem benötigt, die Datenpakete einer Lastspitze vollständig zu bearbeiten und an welchen Stellen große Wartezeiten auftreten. Weiterhin kann untersucht werden, wieviele SDL-Signale an den einzelnen Instanzen gespeichert werden müssen. Somit ist eine Abschätzung des benötigten Speicherplatzes möglich. Besonders problematisch ist die Situation, wenn ein Protokollsystem durch eine Lastquelle überlastet wird. Auch in diesem Fall kann untersucht werden, welche Komponenten nicht ausreichend dimensioniert sind und welche Kapazitäten bereitgestellt werden müssen, um die gewünschten Lasten zu bewältigen. Alternativ kann untersucht werden, welche Lasten von einem System mit einer beschränkten Kapazität verkraftet werden können.

4.5 Engpaßanalyse bei zu geringer Performance

Das in Abschnitt 4.4 vorgestellte System kann ebenfalls dazu verwendet werden, eine Engpaßanalyse vorzunehmen. Dazu ist im Rahmen dieser Arbeit untersucht worden, welchen Einfluß die Einstellung der Werte für die Sendewiederholungs-Timer in Abhängigkeit der vorhandenen Bandbreite auf die Protokoll-Performance hat. Dazu sind die Quellen so spezifiziert worden, daß diese mit einer mittleren Senderate von 10 Paketen/s bei einem Burst-Faktor von zwei arbeiten. Die Paketlänge der zu übertragenden Pakete ist mit 4096 Bit konstant gehalten worden.

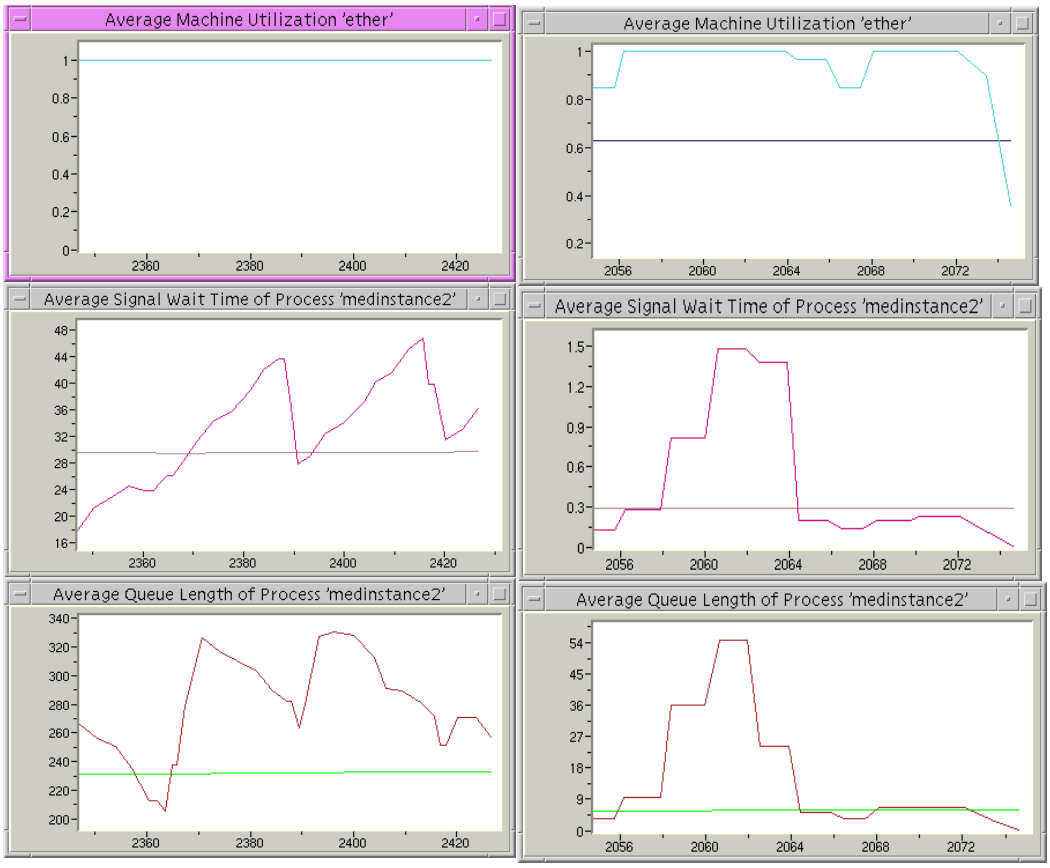


Abbildung 4-17 **Maschinenauslastung, Warteschlangenlänge und Signalwartezeiten bei ausreichender Performance (rechts) und Überlast (links)**

Das Übertragungsmedium ist mit einer Bandbreite von 128 kBit/s ausgestattet worden. Werden in diesem System die Timer-Werte auf 0.3 s festgesetzt, so ergibt sich ein Verhalten entsprechend Abbildung 4-17 links.
Die Übertragungszeit für die Datenpakete zuzüglich der Zeit für die Übertragung der Bestätigung sind größer als die Zeitspanne, die durch den Timer-Wert vorgegeben ist. Daher kommt es ständig zu Sendewiederholungen, obwohl die Datenpakete korrekt beim Empfänger ankommen. Daraus resultiert eine anwachsende Warteschlangenlänge des Prozesses *Medinst2,* die wiederum zu einem Anstieg der Wartezeiten führt. Die Auslastung der QSDL-Maschine liegt ständig bei Eins, d.h. sie ist überlastet. Das Gesamtsystem ist somit nicht stabil.

Durch Veränderung der Timer-Werte kann das Protokoll in einen Zustand überführt werden, in dem die Maschine die angebotene Last bewältigt. Erhöht man den Timer-Wert von 0.3 auf 0.6 s, so ergibt sich ein Performance-Verhalten entsprechend Abbildung 4-17 rechts. Die mittlere Maschinenauslastung pendelt sich auf etwa 0.6 ein, während für die mittlere Wartezeit ein Wert von 0.3 Sekunden zu erwarten ist. Zusammen mit der Bedienzeit eines Datenpaketes ergibt sich eine Gesamtdurchlaufzeit, die deutlich unterhalb von 0.6 s liegt.

Ein ähnliches Verhalten ließe sich auch erzielen, wenn man die Bandbreite des Mediums auf 256 kBit/s erhöhen würde. Dazu wären aber zusätzliche Kapazitäten notwendig, die bei einer Änderung der Timer-Werte nicht bereitgestellt werden müssen.

4.6 Durchsatzanalysen eines Transportprotokolls

Als letztes Beispiel in diesem Kapitel soll nun die Analyse des Transportprotokolls TCP vorgestellt werden. Anhand dieses Beispiels wird der kombinierte Einsatz von transienter und stationärer Analyse mit QUEST demonstriert. Das Untersuchungsobjekt wird im Rahmen dieser Arbeit nur skizziert. Die vollständige Untersuchung des Protokolls und ein Vergleich der erzielten Ergebnisse mit einer klassischen Performance-Simulation ist in [60] veröffentlicht worden.

.

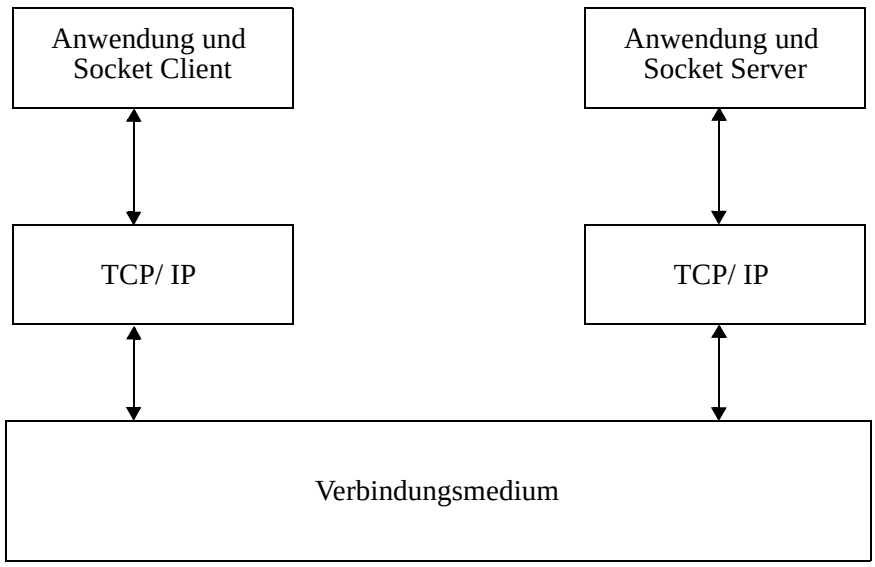


Abbildung 4-18 **Das QSDL-System für TCP**

Das *Transmission Control Protocol* (TCP) realisiert einen verbindungsorientierten zuverlässigen Dienst und stützt sich dabei auf einen unzuverlässigen Datagram-Dienst, wie er von IP angeboten wird. Zu den wesentlichen Funktionen, die von TCP erbracht werden, gehören die Segmentierung von Anwenderdaten, die Vernichtung von Duplikaten sowie Flußkontrolle und Fehlerbehebung durch die Verwendung von Timern, Bestätigungen und Sequenznummern [107].

TCP verwendet zu Beginn einer neuen Verbindung den *Slow-Start-Mechanismus*, um eine Verstopfung von Zwischenknoten zu verhindern. Zunächst ist es der Sendeinstanz erlaubt, ein TCP-Segment zu versenden. Erst nach dem Empfang einer Quittung, die durch den Empfänger erstellt wird, kann der Sender die Anzahl unbestätigter Segmente um eins erhöhen. Dieses Schema erhöht die Anzahl unbestätigter Segmente mit jeder positiven Quittung und führt so zu einer Verdopplung dieses Wertes je *Round Trip Time* (Zeitdauer für die Sendung des Segments und den Rückweg der Quittung).

Wenn erstmalig ein Segment innerhalb einer Verbindung verloren geht, wird dies von der Empfangsinstanz dadurch angezeigt, daß diese sofort das letzte korrekt empfangene Segment erneut bestätigt. Diese Bestätigung wird als *duplicated acknowledgement* bezeichnet. Empfängt der Sender das erste *duplicated acknowledgement* für eine Verbindung, wird der aktuelle Wert für die Sendefenstergröße halbiert und als Schwellwert (*ssthresh*) für diese Verbindung gespeichert. Solange der aktuelle Wert für die Sendefenstergröße unterhalb dieser Schwelle liegt, wird der *Slow-Start-Algorithmus* ausgeführt. Liegt der Wert oberhalb dieser Schwelle wird der sog. *Congestion-Avoidance-Algorithmus* ausgeführt. Dieser erlaubt lediglich die Erhöhung der Größe des Sendefensters um ein Segment je Round-Trip-Time.

Es existieren diverse TCP-Implementierungen, die sich durch die Art und Weise unterscheiden, wie sie auf Paketverluste reagieren. Zu den am weitesten verbreiteten Implementierungen zählen *TCP-Tahoe*, *TCP-Reno* und *Sack-TCP* [41]. *TCP-Tahoe* verwendet den *Slow-Start-Algorithmus*, den *Congestion-Avoidance-Algorithmus* und den sog. *Fast-Retransmit-Algorithmus*. Dieser Algorithmus wiederholt die Sendung eines unbestätigten Segmentes, wenn es das dritte *duplicated acknowledgement* empfängt, ohne auf den Ablauf des *Retransmit-Timers* zu warten. *TCP-Tahoe* startet jedesmal den *Slow-Start-Algorithmus* neu, wenn Segmentverluste erkannt werden.

TCP-Reno arbeitet analog zu *TCP-Tahoe*, modifiziert aber dessen *Retransmit-Algorithmus*. *TCP-Reno* unterscheidet, ob der Segment-Verlust durch Ablauf des *Retransmit-Timers* oder durch den Empfang des dritten *duplicated acknowledgements* erkannt wird. Bei Ablauf des *Retransmit-Timers* beginnt auch *TCP-Reno* mit dem *Slow-Start-Algorithmus*, andernfalls wechselt das Protokoll in die sog. *Fast-Recovery-Phase*. In dieser Phase wird zunächst das älteste ausstehende Segment wiederholt. Jedes neu ankommende *duplicated acknowledgement* führt nun dazu, daß weitere ausstehende Segmente versendet werden. Wenn die erste Bestätigung für bisher unbestätigte Segmente kommt, verläßt diese Protokollvariante die *Fast-Recovery-Phase* und setzt die Übertragung mit dem *Congestion-Avoidance-Algorithmus* fort.

Die Protokollvariante *Sack-TCP* versucht die Nachteile der beiden anderen Protokollvarianten zu vermeiden. Dazu ist es allerdings notwendig, daß der Empfänger von TCP-Segmenten zusätzliche Informationen an den Sender zurückmeldet. So meldet der Empfänger dem Sender zurück, welche Segmente korrekt empfangen worden sind und lediglich zwischengespeichert werden bis fehlende Segmente erneut übertragen worden sind. Wenn der Sender das dritte *duplicated acknowledgement* empfängt, wechselt er analog zum Verhalten von *TCP-Reno* in die *Fast-Recovery-Phase*. In dieser Phase wird eine Variable namens *pipe* verwaltet. Der Wert der Variablen korrespondiert mit der berechneten Anzahl von unbestätigten Paketen in der Verbindung. Der Wert wird um eins erhöht, wenn ein neues Paket oder ein wiederholtes Paket gesendet wird.

Mit dem Empfang eines jeden *duplicated acknowledgements* wird der Wert der Variablen um eins vermindert. Der Sender darf nur dann weitere Pakete senden, wenn der Wert der Pipe-Variablen kleiner ist als der Wert für die Sendefenstergröße. Die Entscheidung, ob neue oder alte Segmente gesendet werden, wird durch eine Datenstruktur namens *scoreboard* geregelt, die vom Sender verwaltet wird. Neue Pakete werden nur dann gesendet, wenn keine Segmente an der Empfängerseite fehlen. Die *Fast-Recovery-Phase* wird verlassen, wenn eine Quittung vom Empfänger kommt, die keinerlei *Sack*-Informationen enthält. Geht ein wiederholtes Paket verloren, so wird dieser Verlust durch Ablauf des *Retransmit-Timers* entdeckt und der Sender startet den *Slow-Start-Algorithmus*.

Die Auswirkungen auf die Performance einer TCP-Verbindung in Abhängigkeit des angenommenen Verlustmodells der drei Protokollvarianten kann mit Hilfe des QUEST-Ansatzes untersucht werden. Dazu ist ein SDL-System entsprechend Abbildung 4-18 spezifiziert worden, das aus einer Anwendungsschicht, einer TCP/IP-Schicht und einer Medienschicht besteht. Während in der Anwendungsschicht das Verhalten von TCP-Nutzern sowie der Zugang über einen Socket modelliert wurde, enthält die TCP/IP-Schicht die Automaten der Protokollinstanzen entsprechend der RFCs. In der Medienschicht ist das Szenario einer Simulationsstudie ([41]) nachgebildet worden, die als Referenzstudie verwendet wurde, um die erzielten Ergebnisse mit anderen Performance-Studien zu vergleichen. Durch die Übereinstimmung der Ergebnisse soll gezeigt werden, daß der QUEST-Ansatz in der Lage ist, dieselben Ergebnisse zu erzielen, die mit separaten simulativen Performance-Modellen erzielt werden. Diese Ergebnisse können mit Hilfe des QUEST-Ansatzes **ohne** die Erstellung separater Performance-Modelle erzielt werden. Darüberhinaus werden Modelle und Methoden genutzt, die auch für den Entwurf von Protokollen verwendet werden. An dieser Stelle wird auf die detaillierte Darstellung des SDL- und auch des QSDL-Systems verzichtet, da die Modellerstellung hier nicht im Fokus des Interesses liegt. Die vollständigen Modellierungsansätze können in [60] nachlesen, die vollständige QSDL-Spezifikation befindet sich im Anhang zu dieser Arbeit.

Durch die Online-Visualisierung der QSDL-Sensoren kann der Durchsatz einer TCP-Verbindung zur Simulationszeit beobachtet werden. In Abbildung 4-19 ist der Einfluß des Verlustes eines einzelnen TCP-Segmentes auf den Durchsatz der zugehörigen Verbindung für die untersuchten TCP-Varianten dargestellt.

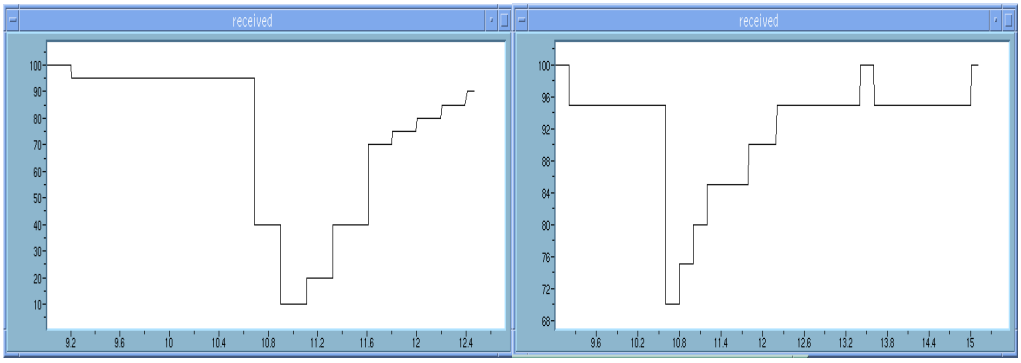


Abbildung 4-19 **Performance-Verhalten bei Verlusten eines einzelnen Paketes (Tahoe links, Reno und Sack rechts)**

TCP-Reno und Sack-TCP verhalten sich im Falle eines Einzelverlustes gleich, siehe Abbildung 4-19 rechts. Man sieht sehr deutlich, wie der Durchsatz bei TCP-Tahoe auf fast Null

absinkt, da diese Variante den Slow-Start-Algorithmus ausführt und wieder mit einer Sendefenstergröße von eins beginnt. Man sieht nicht nur sehr deutlich, daß das Protokoll in drei Schritten den Wert für das Sendefenster jeweils verdoppelt, bevor es dann in den Congestion-Avoidance-Algorithmus wechselt, sondern auch, daß die Round Trip Time etwa zwei Sekunden beträgt und das Protokoll etwa sechzehn Sekunden (8x RTT) benötigt, um den ursprünglichen Durchsatz der Verbindung wieder herzustellen.

Im Gegensatz dazu gelingt es den beiden Varianten TCP-Reno und Sack-TCP, den Durchsatz bei einem Wert von etwa 70 TCP-Segmenten pro Sekunde zu halten. Mit dem Eintritt in die Fast-Recovery-Phase gelingt es, den Durchsatz bereits nach fünfzehn Sekunden wieder auf den ursprünglichen Wert anzuheben.

Noch deutlicher werden die Unterschiede, wenn man den Fall betrachtet, bei dem zwei aufeinanderfolgende TCP-Segmente verloren gehen, siehe Abbildung 4-20. Während TCP-Tahoe wieder mit dem Slow-Start-Algorithmus beginnt und achtundzwanzig Sekunden benötigt, um den ursprünglichen Durchsatz wieder zu erreichen, sieht man sehr deutlich, wie TCP-Reno zunächst den Durchsatz von 100 Segmenten/s auf 60 Segmente reduziert, während der zweite Verlust dazu führt, daß der Retransmit-Timer abläuft und so ebenfalls der Slow-Start-Algorithmus ausgeführt wird. Sack-TCP hingegen zeigt sich sehr robust und kann bereits nach sechzehn Sekunden wieder den ursprünglichen Durchsatz bereitstellen.

Während diese transiente Analyse das Verständnis für die Protokollabläufe erhöht und die Reaktionszeiten der Varianten auf Paketverluste visualisiert, ergibt sich zusammen mit einer stationären Analyse ein vollständiges Performance-Bild der unterschiedlichen Protokollvarianten.

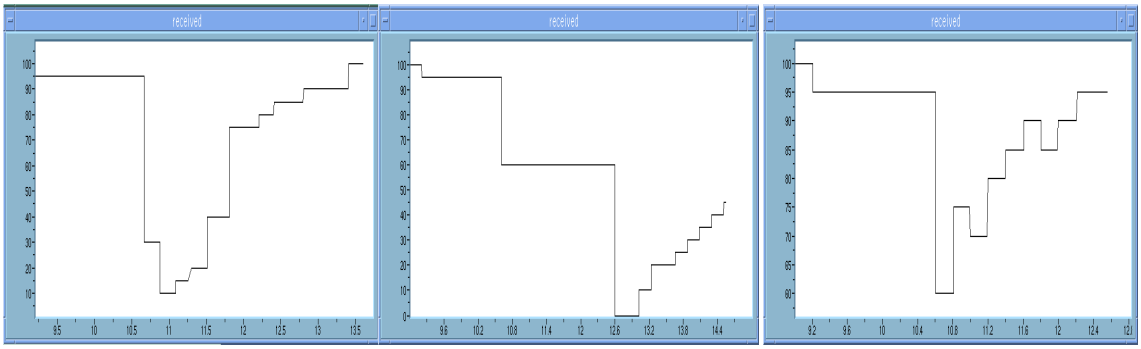


Abbildung 4-20 **Performance-Verhalten bei Verlusten von zwei aufeinanderfolgenden Paketen (Tahoe, Reno, Sack von links nach rechts)**

In einer Experimentserie wurden daher für alle Protokollvarianten der effektive Durchsatz einer TCP-Verbindung in Abhängigkeit von Abstand und Anzahl von Paketverlusten ermittelt. Zusätzlich ist die Übertragungszeit für einen Filetransfer von 15 MByte Dateigröße bei Verwendung von Segmenten der Größe 1024 Byte ermittelt worden. Bei allen Untersuchungen sind die Abstände von Verlustphasen als exponential verteilt angenommen worden. Die mittleren Abstände der Verluste sind von zehn Sekunden bis dreißig Sekunden variiert worden. Darüberhinaus sind je Verlustphase Einzelverluste sowie Mehrfachverluste von bis zu vier Paketen modelliert worden. Die Ergebnisse dieser Experimentserie sind in Abbildung 4-21 bis Abbildung 4-23 dargestellt.

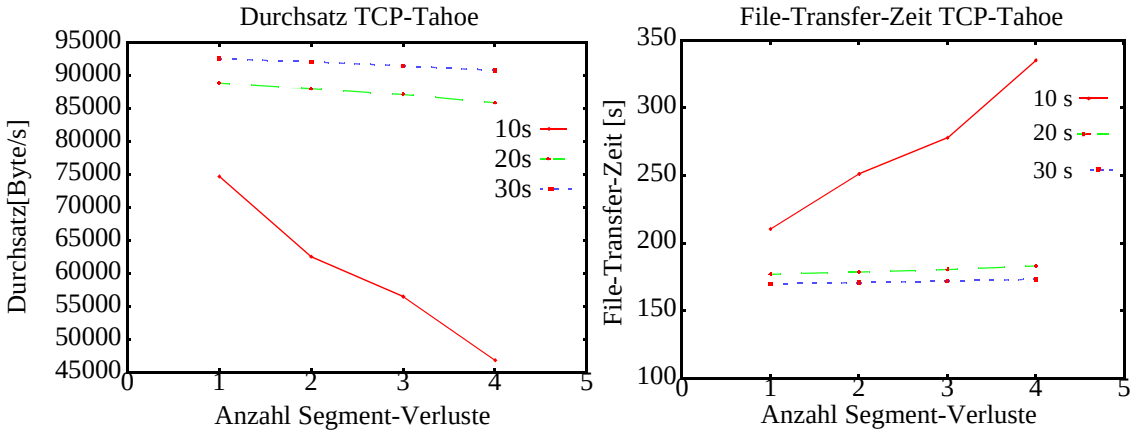


Abbildung 4-21 **Stationäre Leistungsmaße für TCP-Tahoe**

Die Protokollvariante TCP-Tahoe ist sehr robust und auch bei Mehrfachverlusten sinkt der Durchsatz von 90 KBytes/s lediglich auf 74,7 KBytes/s bei Einzelverlusten, bzw. auf 46,8 KBytes/s bei vier Verlusten, siehe Abbildung 4-21 links. Die Übertragungszeit für die 15 MByte große Datei steigt von 169 Sekunden auf 335 Sekunden, vgl. Abbildung 4-21 rechts.

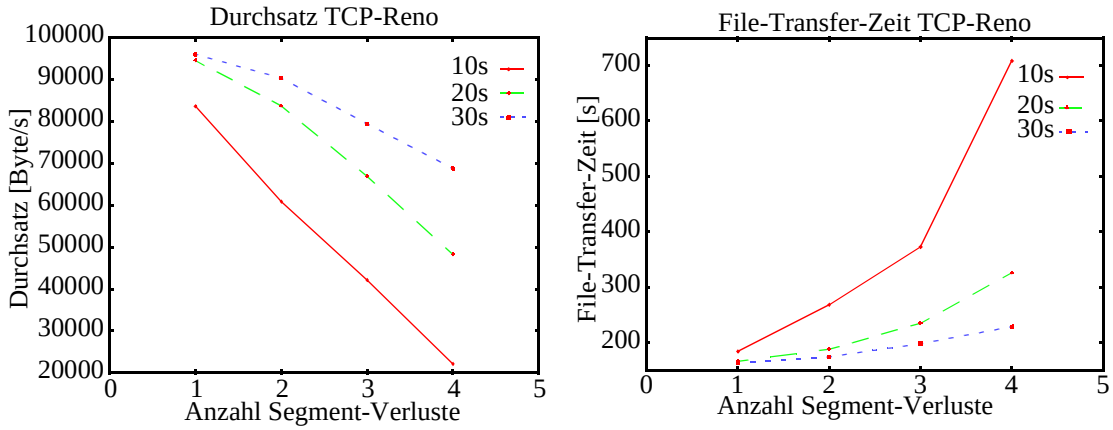


Abbildung 4-22 **Stationäre Leistungsmaße für TCP-Reno**

TCP-Reno ist für den Fall von Einzelverlusten optimiert. Das erkennt man an dem vergleichsweise hohen Durchsatz, der in diesem Falle noch erreicht wird. So sinkt der beobachtbare Durchsatz lediglich auf 83,5 KBytes/s, wenn lediglich ein Segment verloren geht. Dafür sinkt der Durchsatz dramatisch, wenn es zu Mehrfachverlusten kommt. Bei vier aufeinanderfolgenden Verlusten kann ein Durchsatz von lediglich 22,1 KBytes/s aufrecht erhalten werden. So steigt die Übertragungszeit für 15 MByte Daten auf 700 Sekunden und liegt damit mehr als doppelt so hoch wie der Wert für TCP-Tahoe.

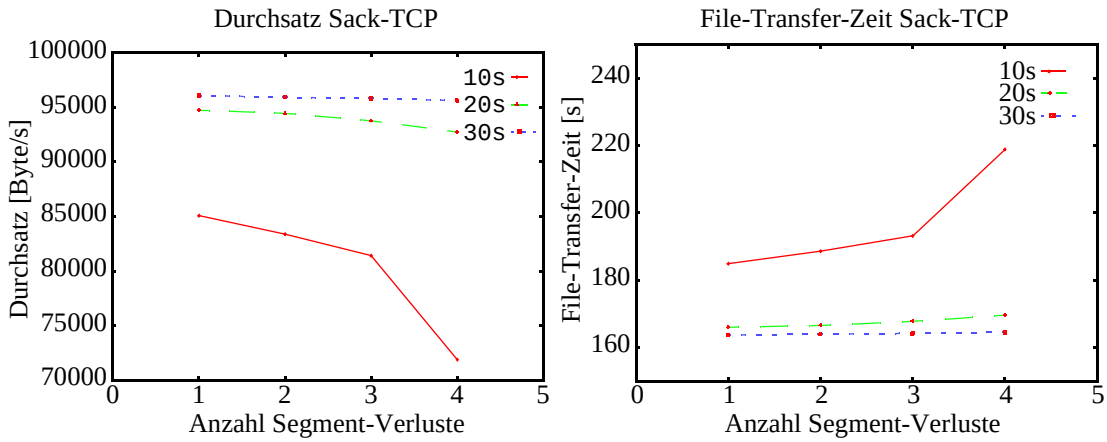


Abbildung 4-23 **Stationäre Leistungsmaße für Sack-TCP**

Das beste Performance-Verhalten über den gesamten Parameterraum weist *Sack-TCP* auf. Der minimale Durchsatz fällt auf 71,8 KBytes/s ab. Dieser Wert kann sogar auf 92 KBytes/s erhöht werden, wenn der Abstand der Verlustphasen mindestens zwanzig Sekunden beträgt. Die maximale Übertragungszeit für die Datei der Größe 15 MByte erreicht einen Wert von 220 Sekunden, während unter der Einzelverlustannahme sogar lediglich 163 Sekunden benötigt werden.

Die Analyse der Transportprotokollvarianten hat gezeigt, wie sehr die beiden Varianten TCP-Reno und TCP-Tahoe auf die Szenarien Einzelpaket- bzw. Mehrfachpaketverlust optimiert worden sind. Außerdem läßt sich quantifizieren, wie groß der Performance-Vorteil der Protokollvariante Sack-TCP gegenüber den beiden Standardvarianten TCP-Reno und TCP-Tahoe ausfällt. Somit kann eine fundierte Aufwand/Nutzenanalyse durchgeführt werden.

Diese Ergebnisse können sowohl durch Online- Modellbeobachtung als auch durch die Auswertung der stationären Leistungsmaße durch die QUEST-Sensoren beobachtet werden. Darüberhinaus ergibt sich durch die kombinierte Betrachtung des stationären und transienten Verhaltens des Protokolls ein umfassendes Bild über das zeitliche Verhalten des SDL-Systems.

Diese Betrachtung kann mit Hilfe eines Modells vorgenommen werden, das aus der funktionalen Beschreibung abgeleitet worden ist. Klassische Performance-Modelle erfordern nicht nur die Erstellung separater Modelle, sondern erfordern entweder getrennte Modelle für die Ermittlung stationärer und transienter Leistungsmaße oder die Anwendung unterschiedlicher Algorithmen auf dieselben Modelle. Der QUEST-Ansatz hingegen ermöglicht die Ermittlung beider Maße durch die Anwendung **einer** Analysetechnik auf **ein** Modell.

4.7 Zusammenfassung Kapitel 4

In diesem Kapitel wurde die QUEST-Methodik für die Integration von modellgestützten Performance-Analysen in den Entwicklungs- und Implementierungsprozeß von Kommunikationsprotokollen dargestellt. Dabei sind nicht nur für die gängigsten Implementierungsansätze *Server-Modell* und *Activity-Thread-Modell* QSDL-Bausteine entwickelt worden, sondern darüberhinaus anhand komplexer Szenarien, Analysetechniken für den Einsatz des Werkzeugs QUEST vorgestellt worden. Die entwickelte Methodik wird nun angewendet und anhand von Beispielen erläutert, um homogene Performance-Modelle für geschichtete Kommunikationsprotokolle zu

entwickeln und zu analysieren. Dabei wird zunächst die Erstellung von QSDL-basierten Lastmodellen im Vordergrund stehen, ehe der Fokus auf die Erstellung von Performance-Modellen für Link-Layer-Protokolle als Diensterbringer für Transportsysteme wechselt.

Kapitel 5

Lastmodellierung mit QSDL

Die Leistung eines Systems S kann mit Hilfe einer abstrakten Funktion *F* ausgedrückt werden, die für ein Tupel (*Last, Maschine)* die zugehörigen Leistungskennzahlen beschreibt. Die SDL-Sicht eines Systems, sei es in Entwicklung oder realisiert, besteht immer aus dem System und einer mit dem System kommunizierenden Umgebung. Fassen wir das SDL-System oder besser das um Maschinen erweiterte QSDL-System als abstrakte Maschine auf, so bietet diese abstrakte Maschine der Umgebung Dienste an. Die Umgebung kann diese Dienste durch Signale anfordern, bzw. das System kann die Ergebnisse von Anforderungen an die Umgebung zurückgeben, vgl. Abbildung 5-1.

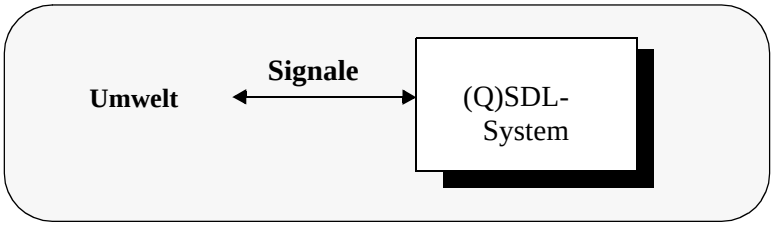


Abbildung 5-1 **Interaktion zwischen System und Umgebung**

Die zwischen Umgebung und System ausgetauschten Signale, deren Häufigkeiten und die mit den Signalen angeforderten Dienste legen zusammen mit den konkreten Ausprägungen der Signalparameter die Belastung des *QSDL-Systems* fest. In der Umgebung des (Q)SDL-Systems können daher Lastprozesse definiert werden, die die vom System angebotenen Dienste durch die definierten Dienstprimitive (Signale) anfordern und dabei die für eine Anwendung typischen Lastmuster erzeugen.

Die vor allem in der analytischen Bewertung angewandte Technik, die Verkehrscharakteristik mehrerer Quellen eines Typs oder die Überlagerung von Verkehrsströmen unterschiedlicher Anwendungen durch eine, den resultierenden Verkehrsstrom beschreibende stochastische Ver-

teilung anzugeben, ist im Rahmen dieser Arbeit nicht weiter verfolgt worden. Vielmehr sind Prozesse spezifiziert worden, die das logische und zeitliche Verhalten eines Anwenders nachbilden, d.h. das Auftreten von Schnittstellenereignissen entspricht dem Auftreten dieser Ereignisse in realen Anwendungen. Die Überlagerung mehrerer Anwender kann dann durch Erzeugung mehrerer solcher Prozesse realisiert werden.

Die spezifizierten Lastprozesse werden gemeinsam mit dem erweiterten SDL-System in ein quantitativ bewertbares Simulationsmodell überführt, durch dessen Ausführung die Leistungskennzahlen ermittelt werden können. Der Vorteil des QSDL-Ansatzes zur Lastmodellierung besteht darin, daß die Beschreibung der Lastprozesse dieselben Sprachmittel und Konzepte benutzt, mit denen auch das System beschrieben ist. Somit kann eine integrierte Beschreibung von Last und abstrakter Maschine vorgenommen werden. Entwickler, die ihre Systeme mit SDL beschreiben, können sehr einfach und intuitiv die gewünschten Lastprozesse mit QSDL beschreiben, bzw. die vorgefertigten Bausteine entsprechend parametrisieren.

In den nun folgenden Abschnitten werden Lastprozesse für die typischen Multimedia-Anwendungen vorgestellt, die als vorgefertigte Bausteine im Werkzeug QUEST verwendet werden[a] können.

5.1 Lastquellen für diskrete Datenströme

Unter diskreten Datenströmen versteht man klassische Computerdatenströme, die keine Anforderungen an Mindestbandbreiten oder Verzögerungsschranken haben. Neben den neuen Herausforderungen spielen auch diese klassischen Datenanwendungen wie Telnet [89] oder FTP [90] weiterhin eine Rolle in Multimedia-Netzen. In vielen Studien, die analytische Modelle verwenden, werden die Ankunftsprozesse sowohl für Netzverbindungen als auch für Paketankünfte innerhalb einer Verbindung als Poisson-Prozeß modelliert. Paxon [85], [86] hat in mehreren Studien herausgefunden, daß diese Annahmen für den Ankunftsprozeß von Netzverbindungen zutreffend sind, während die Annahme für die Paketankünfte innerhalb einer Verbindung die Schwankungen der Verkehrsintensitäten (*Burstiness)* des Verkehrs unterschätzen. Eine mögliche Lösung zur realistischeren Beschreibung der Lastprozesse stellt die Verwendung von empirischen Daten zur Lasterzeugung dar, wie sie in der *tcplib*-Bibliothek bereitgestellt werden [32].

Diese Bibliothek stellt Bausteine zur Verfügung, mit deren Hilfe realistische TCP/IP-Verkehrsströme generiert werden können. Die Daten basieren auf Verkehrsmessungen, die die Anwendungsprogramme FTP, SMTP, NNTP, VMNET, *Telnet* und *Rlogin* an den Universitäten UC Berkeley, *South California*, London und am *Bell Communication Research Center* betreffen. Die genannten Anwendungsprogramme waren für 96% des gemessenen TCP/IP-Verkehrs verantwortlich.

Die Verkehrscharakteristik einer Anwendung wird durch ein zweistufiges Modell beschrieben. Erst wird der Ankunftsstrom von Verbindungen innerhalb einer Anwendung beschrieben, danach das Verhalten einer einzelnen Verbindung. Jede Verbindung ist im Falle einer interaktiven Anwendung wie Telnet durch die Paketgrößenverteilung, die Zwischenankunftsabstandsverteilung von Paketen sowie die Verbindungsdauer charakterisiert. Bei sogenanntem *Bulk*-Datenverkehr legt die Größe der zu übertragenden Datei die Charakteristik einer Verbindung innerhalb der Anwendung fest.

a. Die Bausteine müssen allerdings insoweit verändert werden, daß die Signalnamen und Parameter mit denen übereinstimmen, die zwischen QSDL-System und der Umgebung ausgetauscht werden.

Im Kontext der SDL-basierten Leistungsbewertung müssen Verkehrsquellen als QSDL-Prozesse spezifiziert werden, die dasselbe statistische Verhalten aufweisen, wie es aus Verkehrsmessungen bekannt ist. Dazu werden die Ergebnisse von Langzeitmessungen verwendet, aus denen das gewünschte Verhalten abgelesen werden kann. Das funktionale Modell sowohl von Telnet- als auch von FTP-Anwendern wird durch Modifikation des Modells der *tcplib* in [32] beschrieben.

5.1.1 Telnet

Das Telnet-Protokoll ermöglicht entfernten Anwendern den kommandozeilenorientierten Zugang zu einem Rechner. Es stellt eine Standardanwendung des TCP/IP Protokolls [87], [88], [107] dar und folgt dem Client/Server Paradigma, vgl. Abbildung 5-2.

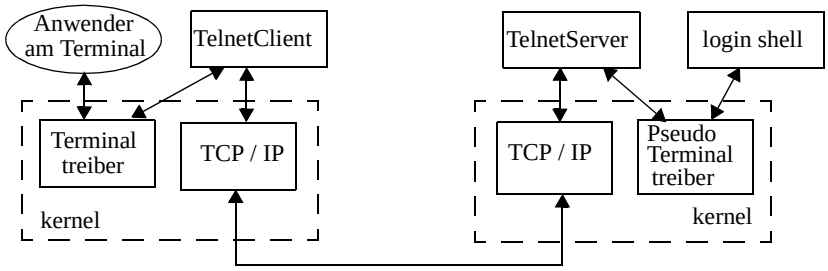


Abbildung 5-2 **Struktur einer Telnet-Anwendung**

Die gesamte Last, die durch Telnet-Anwendungen erzeugt wird, kann durch den Ankunftsprozeß von Telnet-Sitzungen, sowie durch die Lasterzeugung einer Telnet-Sitzung durch den Client und den Server beschrieben werden. Daher sind für Telnet-Anwendungen Verkehrsquellen sowohl für den Server als auch für den Client vorzusehen.

In Abbildung 5-3 ist die Blockstruktur von Telnet-Verbindungsinitiatoren (Clients) dargestellt. Ein *Verbindungsinitiator* erzeugt einen Prozeß, der das zeitliche Verhalten eines Telnet-Anwenders nachbildet. Aus diversen Meßreihen [85] weiß man, daß der Ankunftsstrom von Telnet-Verbindungen sowohl für Minuten- als auch für Stundenintervalle als Poisson-Strom modelliert werden kann. Daher werden im Lastmodell die Prozesse *Telnetsession* mit exponentiell verteilten Zwischenerzeugungsabständen vom *Verbindungsinitiator* erzeugt.

Zur Beschreibung des zeitlichen Verhaltens einer einzelnen Telnet-Sitzung hat sich die Pareto-Verteilung als geeignete Approximation herausgestellt. Die Zwischenankunftabstände der Client-Pakete innerhalb einer Telnet-Sitzung sind Pareto-verteilt mit den Parametern α und k [85].

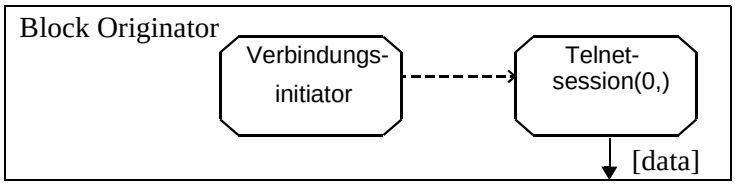


Abbildung 5-3 **Lastquelle für Telnet-Verkehr (Initiator)**

Definition 1 Eine Zufallsvariable X heißt Pareto-verteilt mit Parametern k, α, falls die Verteilungsfunktion gegeben ist durch $F(x) = 1 - \left(\frac{k}{x}\right)^{-\alpha} \quad k, \alpha > 0\,,\ x > k$

Die zugehörige Dichtefunktion $f(x) = \alpha \cdot k^{\alpha} \cdot x^{-\alpha-1}$ ist für einige ausgewählte Parameter α und k in Abbildung 5-4 dargestellt. Mit Hilfe der Inversen-Transformation lassen sich aus U(0,1)-gleichverteilten Zufallszahlen auf einfache Weise Pareto-verteilte Zufallszahlen generieren:

Falls $u \sim U(0,1)$, dann ist $x = \frac{k}{u^{1/\alpha}} = F^{-1}(x)$ pareto-verteilt.

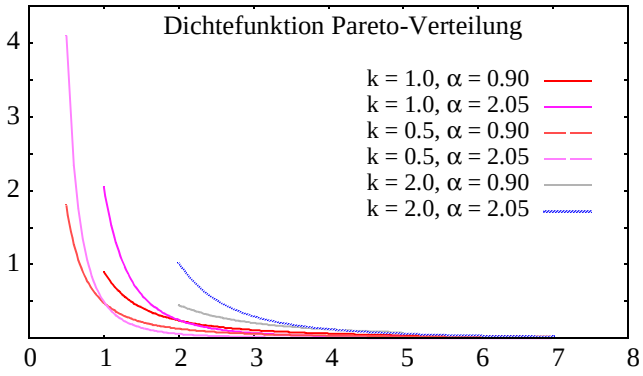


Abbildung 5-4 **Dichtefunktionen der Pareto-Verteilung**

Die Länge einer Telnet-Sitzung ist durch die Anzahl der versendeten Pakete zusammen mit den Zwischenankunftsabständen festgelegt. Zur Beschreibung der Anzahl versandter Telnet-Pakete einer Verbindung ist die $\log_2$-Normalverteilung mit Mittelwert μ und Standardabweichung σ geeignet [85].

Falls die Zufallsvariable $Y = lg_2(X)$ normalverteilt ist, dann ist die Zufallsvariable X $\log_2$-normalverteilt. Zur Erzeugung $\log_2$-normalverteilter Zufallszahlen werden N(0,1)-verteilte Zufallszahlen generiert und diese dann transformiert. Falls $u \sim N(0,1)$, dann ist $x = 2^{\mu + \sigma x}$ $\log_2$-normalverteilt mit Parameter μ und σ [74]. Über die Paketgrößenverteilung finden sich keine Angaben in den Studien. In der *tcplib*-Bibliothek erzeugt der Telnet-Client Pakete mit Nutzdaten der Größe 1 Byte. Diese Annahme muß aber wegen des Nagle-Algorithmus[b] als nicht zutreffend angesehen werden. So wurden zum Beispiel in [86] in 85000 Paketen insgesamt 139000 Byte übertragen. Das im Rahmen dieser Arbeit entwickelte Verkehrsmodell verwendet daher Nutzdatengrößen von 1 bis 5 Byte, wobei eine prozentuale Aufteilung der Größen gemäß Tabelle 5 -1 vorgenommen wurde.

1Byte	2 Byte	3 Byte	4 Byte	5 Byte
55%	35%	5%	3%	2%

Tabelle 5 -1 **Größenverteilung Nutzdaten Telnet-Client**

b. Algorithmus zur Minimierung der Anzahl *tinygrams* (1 Byte Nutzdaten), beschrieben im RFC 896.

Mit den erläuterten Verteilungen lassen sich nun parametrisierte Lastprozesse spezifizieren, deren erzeugte Lastmuster in ihrem statistischen Verhalten keine signifikanten Unterschiede zu den von realen Applikationen erhobenen Meßdaten aufweisen.

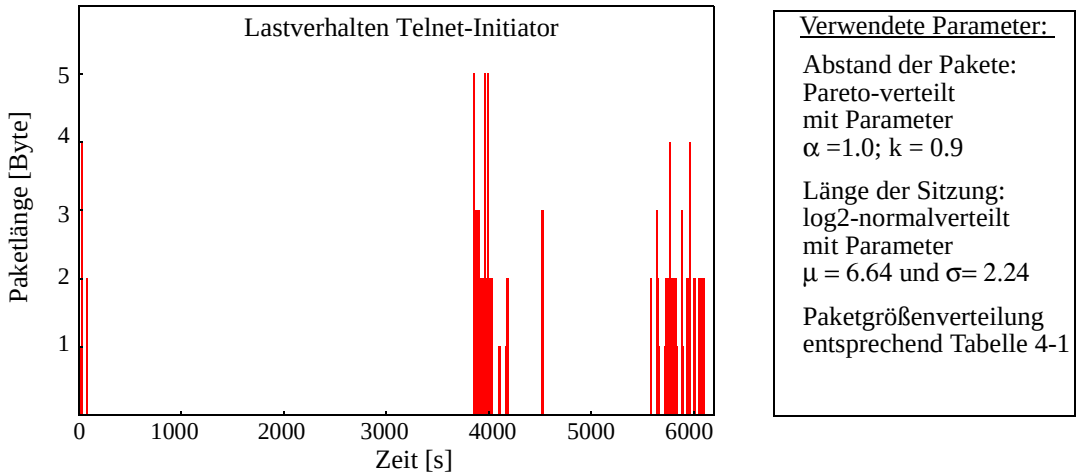


Abbildung 5-5 **Lasterzeugung durch Telnet-Initiator**

Abbildung 5-5 zeigt ein Lastmuster, wie es durch eine Telnet-Sitzung erzeugt werden könnte. Die Parameter entsprechen denen der Studien in [85] und [86]. Die dargestellte Telnet-Sitzung hat eine Dauer von etwa zwei Stunden, in denen sich stark unterschiedliche Aktivitätsphasen zeigen.

Innerhalb einer Phase erhöhter Aktivität stellt sich das Lastverhalten des Telnet-Clients entsprechend Abbildung 5-6 dar.

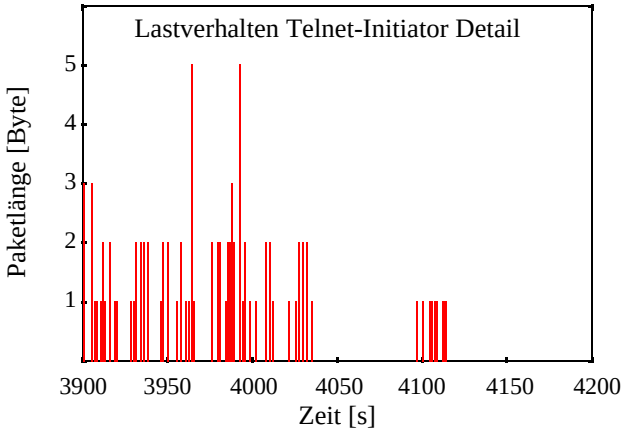


Abbildung 5-6 **Lasterzeugung Telnet-Initiator im Detail**

Eine wesentlich stärkere Belastung des unterliegenden Transportsystems erzeugt der Telnet-Server, da die von ihm zurückgesendeten Pakete größer sind als die vom Client verschickten.

Für jede Telnet-Session wird auf dem Zielrechner ein Prozeß *Sessionresponder* auf Anforderung durch den Client erzeugt, der die vom Initiator kommenden Daten empfängt und darauf reagiert.

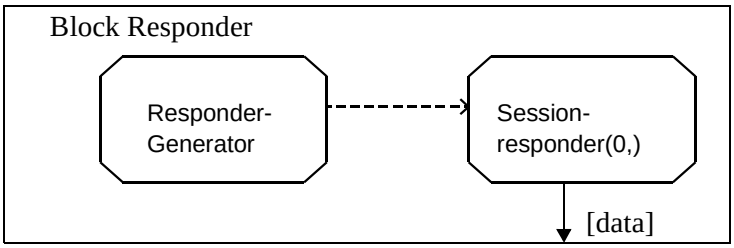


Abbildung 5-7 **Lastquelle für Telnet-Verkehr (Responder)**

Aus Verkehrsmessungen ist bekannt, daß das Verhältnis von Initiator-Daten zu Responder-Daten einer $\log_2$-Normalverteilung mit Mittelwert μ und Standardabweichung σ folgt [85]. Im verwendeten Lastprozeß wird für jedes empfangene Paket vom Telnet-Initiator ein Paket zurückgesendet. Die Paketgröße wird als Vielfaches der Paketgröße des empfangenen Paketes gewählt. Auch hier handelt es sich dabei um Nutzdaten, zu denen natürlich ein konstanter Protokoll-Header hinzugefügt wird. Dieses funktionale Verhalten entspricht dem Vorgehen, wie es in der *tcplib*-Bibliothek angewendet wird.

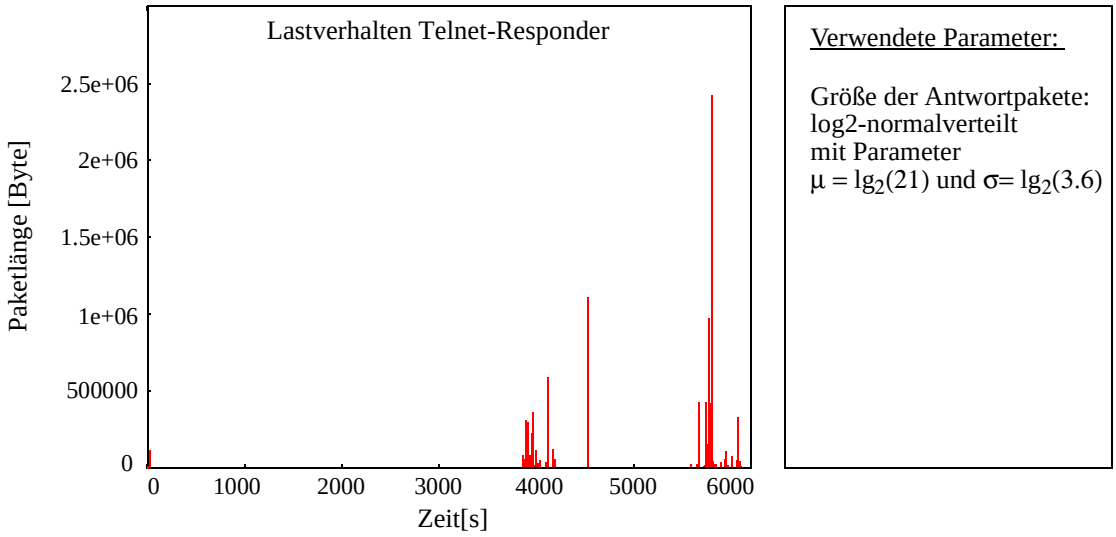


Abbildung 5-8 **Lasterzeugung durch Telnet-Responder**

Analog zum Lastverhalten des Initiators ergibt sich ein mögliches Lastverhalten eines Responders gemäß Abbildung 5-8 für die gesamte Sitzung bzw. gemäß Abbildung 5-9 eine detailliertere Sicht auf ein Intervall mit erhöhter Aktivität.

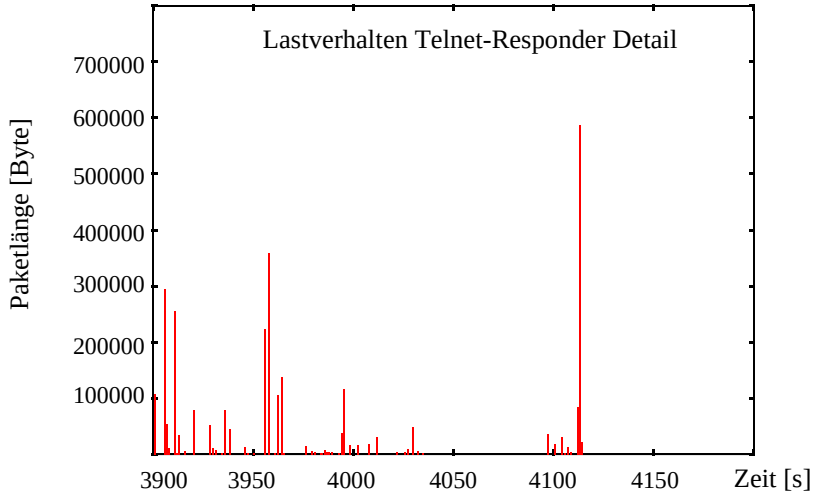


Abbildung 5-9 **Lasterzeugung Telnet-Responder im Detail**

5.1.2 File-Transfer-Protokoll

Bei der Modellierung von Verkehrslasten, die den gemessenen Daten von FTP-Sitzungen entsprechen, sollte die Struktur der Sitzungen berücksichtigt werden. Jede FTP-Sitzung (*ftp* bis *quit*) besteht aus einer Folge von FTP-Datenströmen (*open* bis *close*). Innerhalb der FTP-Datenströme lassen sich Lastspitzen (Datenbursts) (*get, put, ls-command*) ausmachen, siehe Abbildung 5-10.

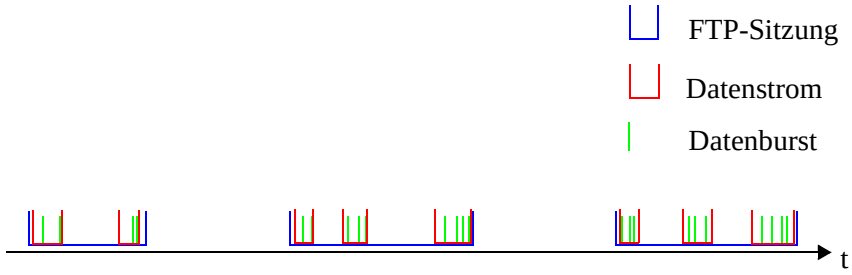


Abbildung 5-10 **Zeitliche Struktur einer FTP-Sitzung**

Zur vollständigen Charakterisierung von Lastquellen, die das Verhalten von FTP-Anwendern nachbilden, müssen daher die Zwischenankunftsabstände und die Dauern von FTP-Sitzungen berücksichtigt werden. Weiterhin müssen pro FTP-Sitzung der Abstand sowie die Länge von FTP-Datenströmen und innerhalb der Datenströme die Ankunftszeitpunkte von Datenbursts und die mit einem Burst auftretende Datenmenge in das Modell integriert werden.

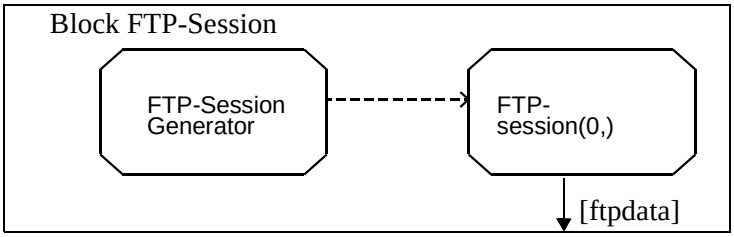


Abbildung 5-11 **Lastquelle FTP-Session**

Die Architektur des Verkehrsmodells für FTP-Anwendungen entspricht der des Modells sowohl von Telnet-Responder als auch von Telnet-Initiator. Es existiert ein Prozeß *FTP-SessionGenerator*, der dynamisch die Prozesse *FTP-Session* erzeugt, vgl. Abbildung 5-11. Die Ankünfte von FTP-Sitzungen bilden einen Poisson-Strom, d.h. die Zwischenankunftsabstände sind exponentialverteilt [86]. Die Länge einer Sitzung wird durch das zu versendende Volumen festgelegt. Die Datenmenge einer FTP-Sitzung ist $\log_2$-normalverteilt mit Mittelwert μ und Standardabweichung σ [85].

Die Pausen zwischen FTP-Datenströmen einer Sitzung folgen einer $\log_2$-Normalverteilung [86]. Die Länge eines Datenstromes wird analog zum Vorgehen bei FTP-Sitzungen durch das Datenvolumen eines Stroms festgelegt. Die Datenmenge eines FTP-Datenstromes ist ebenfalls $\log_2$-normalverteilt mit Mittelwert μ und Standardabweichung σ [85]. Innerhalb von Datenströmen können sog. Datenbursts beobachtet werden, siehe Abbildung 5-10.

Betrachtet man Zeiträume, die kürzer als 10 Minuten sind, so können die Pausenlängen zwischen einzelnen Bursts durch eine Exponentialverteilung beschrieben werden. Daher sind im vorliegenden QSDL-Modell mehrere Aktivitätsniveaus definiert, zwischen denen die FTP-Quelle wechselt. Ein Wechsel zwischen den Niveaus findet immer dann statt, wenn ein Datenstrom länger als 10 Minuten dauert. Die Datenmenge pro Burst folgt einer Pareto-Verteilung. Im folgenden wird nun anhand der Spezifikation des Prozesses *FTPSitzung* erläutert, wie die Ergebnisse der Meßstudien in das QSDL-Modell der Lastquelle einfließen, vgl. Abbildung 5-12.

Der Prozeß *FTP-Sitzung* besitzt den formalen Parameter *volumen*. Der aktuelle Wert des Parameters wird vom Prozeß *SessionGenerator* festgelegt und er definiert damit das zu versendende Datenvolumen in Byte, das innerhalb dieser Sitzung versendet werden soll. Damit ist indirekt die Länge der FTP-Sitzung und damit die Lebenszeit dieses Prozesses festgelegt.

Die Variablen des Prozesses (1) dienen dazu, die Abstände und Längen von Datenströmen, die Pausen zwischen Datenbursts und die mit einem Burst verbundene Datenmenge gemäß der in den Referenzstudien ermittelten Verteilungen festzulegen.

In der Starttransition (2) werden die beiden Normalverteilungen initialisiert, aus denen durch Inversentransformation das Volumen eines Datenstromes sowie die Pausen zwischen den Strömen festgelegt werden. Der Vektor *ankunft* enthält Variablen vom Typ *exponential*, von denen zunächst eine Intensitätsstufe ausgewählt wird und dann gemäß der gewählten Intensitätsstufe die Pause zwischen zwei Datenbursts innerhalb eines Datenstromes festgelegt wird.

```
process FTPSitzung;
  fpar  volumen Integer;
  dcl   gesendet Integer,
        streamlimit Integer,
        rest Integer,
        limit Integer,
        hilf1 Normal,
        hilf2 Normal,
        nextstream Real,
        ankunft Ankunfttyp,
        nextburst real,
        sessionbeginn Time,
        intensitaet Integer,
        aktivitaet Randint,
        burstsize Real,
        expo Real,
        u Real,
        zw Real,
       bursthilf Uniform;
```

(1)

```
start;
  task hilf1 := Init(hilf1,0.0,1.0),
       hilf2 := Init(hilf2,0.0,1.0),
        ankunft(1) := Init(ankunft(1),lambdadata1),
        ankunft(2) := Init(ankunft(2),lambdadata2),
        ankunft(3) := Init(ankunft(3),lambdadata3),
        ankunft(4) := Init(ankunft(4),lambdadata4),
        ankunft(5) := Init(ankunft(5),lambdadata5),
       aktivitaet := Init(aktivitaet,1,5),
      bursthilf  := Init(bursthilf, 0.0,1.0);
  task zw := sample(hilf1);
  task streamlimit :=
       FIX(call potenz(2.0, meandata + vardata * zw));
  task gesendet := 0;
  task rest := volumen;

  task sessionbeginn := now;
  task limit := Call minimum(volumen,streamlimit);
  task intensitaet := sample(aktivitaet),
         nextburst := sample(ankunft(intensitaet));
 nextstate aktiv;
```

(2)

```
state aktiv /*## awake nextburst ##*/;  /* aktiver Datenstrom */
 input none;
 task u := sample(bursthilf),
       expo := 1.0 / burstshapepara,
        zw := call potenz(u,expo);
 task burstsize := burstlocatepara / zw;
      output ftpdata(now, FIX(burstsize));
 task gesendet := gesendet + FIX(burstsize);
 decision gesendet <= limit;
    (true) :  decision now <= sessionbeginn + 600.0;
              (true) : task nextburst := sample(ankunft(intensitaet));
                       nextstate aktiv;
              (false): task intensitaet := sample(aktivitaet),
                           nextburst := sample(ankunft(intensitaet));
                       task sessionbeginn := now;
                       nextstate aktiv;
             enddecision;
    (false):  decision gesendet <= volumen;
              (true) : task zw := sample(hilf2),
                           nextstream := call potenz
                                 (2.0, meanstream + (varstream * zw));
                       task rest := rest - gesendet;
                       nextstate passiv;
              (false): stop;
              enddecision;
  enddecision;
  endstate;
```

(3)

```
state passiv /*## awake nextstream ##*/;
input none;
task zw := sample(hilf1);
task streamlimit :=
      FIX(call potenz
      (2.0, meandata + vardata * zw));
task sessionbeginn := now;
task limit := Call minimum(rest,streamlimit);
task intensitaet := sample(aktivitaet),
       nextburst := sample(ankunft(intensitaet));
nextstate aktiv;
endstate;
```

(4)

(1) **Formale Parameter und Variablen**

(2) **Starttransition**

(3) **Zustand aktiv**

(4) **Zustand passiv**

Abbildung 5-12 **Prozeß FTPSitzung**

Als erstes wird die Datenmenge bestimmt, die mit dem ersten Strom gesendet werden soll. Das Limit für diesen Strom ergibt sich dann als Minimum des Datenvolumens der Sitzung und des Volumens dieses Stromes. Das Minimum wird deswegen bestimmt, weil durch die Verwendung der Heavy-Tail-Verteilungen das aktuelle Datenvolumen eines Stromes innerhalb der Sitzung größer sein kann, als das Datenvolumen der gesamten Sitzung. In diesem Fall wird das Volumen des Stroms abgeschnitten, damit die Charakteristik der Sitzung erhalten bleibt.

Als abschließende Aktion in der Starttransition wird das Aktivitätsniveau und anschließend der Zeitpunkt des nächsten Bursts festgelegt. Der Prozeß wechselt in den Zustand *aktiv* (3).

Beide Zustände des Prozesses sind sogenannte zeitbehaftete Zustände, d.h. es sind mit Hilfe des awake-Konstruktes Zeitpunkte definiert, zwischen denen spontane Transitionen des Prozesses gesperrt sind. Da keine signalkonsumierenden Transitionen spezifiziert sind und die Signalausgaben des Prozesses durch spontane Transitionen getriggert sind, kann so der Zeitpunkt des nächsten *Output* exakt festgelegt werden.

Falls eine spontane Transition aktiviert ist, siehe Abbildung 5-12 (3), wird in dieser die Größe der zu sendenden Datei bestimmt. Durch inverse Transformation werden aus u(0,1)-gleichverteilten Zufallszahlen Pareto-verteilte Zufallszahlen erzeugt. Der konkrete Wert wird als aktueller Parameter dem Signal *ftpdata* mitgegeben. Danach wird festgestellt, ob ein Wechsel des Aktivitätsniveaus stattfinden soll, und der nächste Aufweckzeitpunkt wird gemäß des gültigen Niveaus festgelegt. Der Prozeß bleibt im Zustand *aktiv.*

Sollte die Datenmenge für den aktuellen Strom erreicht sein, wird geprüft, ob auch das Gesamtvolumen der FTP-Sitzung erreicht ist. Falls dies der Fall ist, wird der Prozeß gestoppt, d.h. er existiert nicht weiter im System. Falls das bisher versendete Datenvolumen kleiner ist als das bei der Erzeugung festgelegte Volumen, so wird der Zeitpunkt für den nächsten Datenstrom bestimmt und der Prozeß wechselt in den Zustand *passiv.*

Die Transition im Zustand *passiv* wird ebenfalls durch eine spontane Transition getriggert (4). Wenn diese aktiviert ist, legt der Prozeß das Volumen für den nächsten Strom fest und bestimmt das Aktivitätsniveau und dann entsprechend des Niveaus den Zeitpunkt des nächsten Bursts. Er wechselt in den Zustand *aktiv.* Ein Lastszenario einer FTP-Sitzung mit Parametern, wie sie in den Studien von V. Paxson und S. Floyd [85] und [86] ermittelt wurden, ist in Abbildung 5-13 dargestellt.

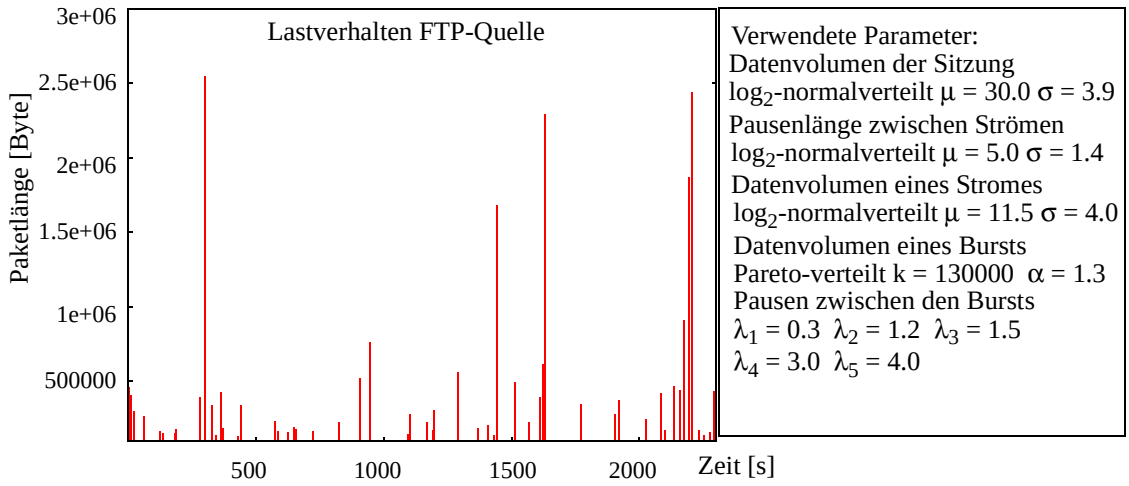


Abbildung 5-13 **Lasterzeugung FTP-User**

5.1.3 World Wide Web

Zu den populärsten Anwendungen im Internet gehört das World Wide Web (WWW). WWW benutzt das Hypertext Transfer Protokoll (HTTP) [47], um eine Verbindung zwischen Web-Client und Web-Server aufzubauen.

HTTP folgt einem einfachen Request/Response Schema, vgl. Abbildung 5-14. Während in den Protokollversionen bis einschließlich HTTP/1.0 für jedes angeforderte HTML Dokument sowie für jedes in diesem Dokument vorkommende Image-File eine HTTP-Verbindung aufgebaut werden mußte [78], sind mit der Version 1.1 dauerhafte Verbindungen eingeführt worden. Über eine geöffnete Verbindung können solange Dokumente per Request angefordert werden, bis entweder Client oder Server die Verbindung explizit schließen, bzw. die Verbindung per Time-out geschlossen wird.

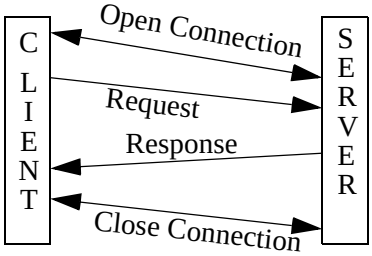


Abbildung 5-14 **HTTP-Request/Response-Schema**

Der Ankunftsprozeß von Client-Requests kann als Poisson-Prozeß modelliert werden. Da mit jedem Dokument in der Regel mehr als eine Datei übertragen werden muß, ergibt sich der gesamte Ankunftsprozeß als Compound Poisson Arrival Process (CPOAP) [78], d.h. mit jedem Ankunftereignis sind mehrere HTTP1.1-Requests verbunden. Die Anzahl der mit einem HTML-Dokument zu übertragenden Image-Dateien ist in der Studie von Li und Latchmann [78] nicht untersucht worden. Da keine weiteren Informationen vorliegen, werden sie als Poisson-verteilt angenommen. In der Studie von Cunha et. al. [30] ist die Dateigrößenverteilung von Web-Dokumenten untersucht worden. Dabei hat sich die Pareto-Verteilung als gute Approximation herausgestellt. Wie schon im Falle von FTP kann auch bei Web-Dokumenten die Größe der Dateien durch die Pareto-Verteilung beschrieben werden. Eine in QSDL spezifizierte Verkehrsquelle für einen Web-Server belastet ein Kommunikationssystem mit den Lasten entsprechend Abbildung 5-15.

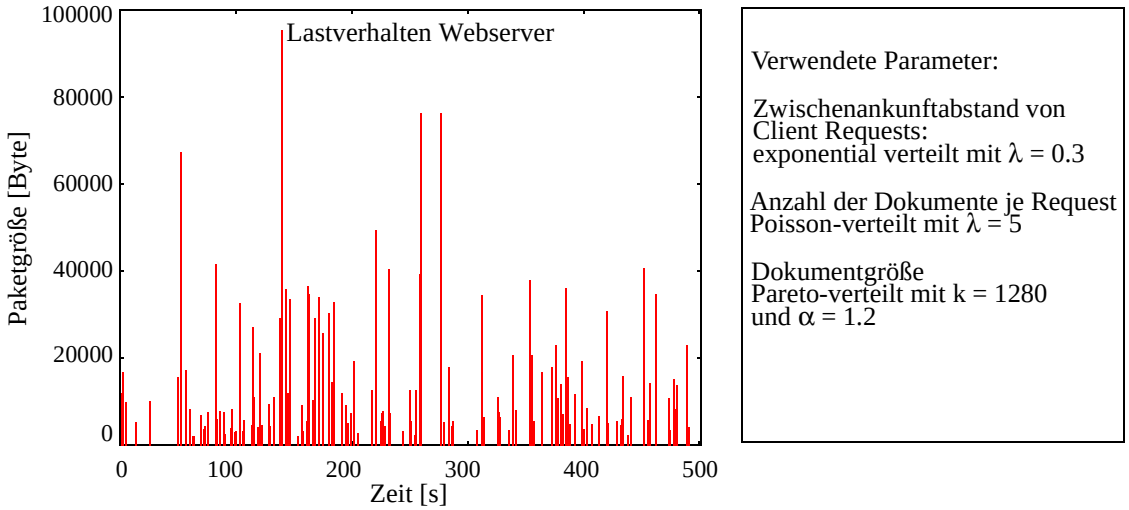


Abbildung 5-15 **Lastprofil einer Verkehrsquelle für Web-Server**

5.2 Lastquellen für kontinuierliche Datenströme

Zu den kontinuierlichen Datenströmen zählen im wesentlichen Töne und Bewegtbilder, aber auch die Werte von Luftdruck-, Temperatur-, Druck- oder Feuchtigkeitssensoren. Die zugrundeliegenden Signale sind zeit- und wertekontinuierlich. Aus diesen analogen Signalen wird durch Abtastung zu festen, regelmäßigen Zeitpunkten der Wert des Signalpegels entnommen und anschließend durch Quantisierung einem diskreten Wert zugeordnet. Werden die Werte anschließend binär codiert, so bezeichnet man das verwendete Verfahren als PCM-Codierung [55]. Die Güte der Codierung und damit die Qualität des digitalisierten Signals wird durch die Anzahl der Quantisierungsstufen und durch die Abtastfrequenz beeinflußt. Je mehr Quantisierungsstufen zur Verfügung stehen, desto genauer kann das ursprüngliche analoge Signal approximiert werden. Kontinuierliche Daten erzeugen ein regelmäßiges Lastmuster, sofern sie unverändert an ein Transportsystem übergeben werden. Die benötigte Bandbreite ergibt sich als Produkt von Abtastrate und Quantisierungstiefe. So wird bei digitalisierter Sprache alle 125 µs ein acht Bit großer Wert ermittelt, was zu einer Datenrate von 64 kBit/s führt. Durch weitere Codierungs- und Kompressionsverfahren lassen sich die benötigten Bandbreiten jedoch reduzieren. Im folgenden werden nun Verkehrsquellen für digitale Sprache, digitalen Ton und Bewegtbildübertragung als QSDL-Prozesse modelliert.

5.2.1 Lastquellen für Sprache

Ausgangspunkt für die Betrachtungen der Verkehrsquellen für Sprache ist das PCM-kodierte Signal. Durch Abtastung, Quantisierung und anschließender Binärcodierung ist aus dem zugrundeliegenden Analogsignal ein streng periodischer Datenstrom, eine Folge von 8-Bit-Werten, entstanden.

Durch Verwendung weiterer Kodierverfahren, die auf diesem Bitstrom aufsetzen, lassen sich weitere Datenreduktionen vornehmen. Das von der ITU standardisierte Verfahren *Differential Pulse Code Modulation* (DPCM) berücksichtigt statistische Abhängigkeiten zwischen benachbarten Abtastwerten und reduziert das Datenvolumen auf 56 kBit/s. Adaptive Erweiterungen des Verfahrens (ADPCM) benötigen nur noch vier Bit pro Abtastwert und reduzieren so das Datenvolumen auf 32 kBit/s ohne Qualitätsverluste gegenüber dem PCM-kodierten Signal.

Weitere Maßnahmen, um die benötigte Übertragungsbandbreite zu reduzieren, sind bereits in den 60er Jahren im Kontext von analogem Telefonverkehr angewendet worden [21]. Durch Sprachaktivitätserkennung werden Aktiv- und Passivphasen in Sprachsequenzen ermittelt. Damit läßt sich digitalisierte Sprache als Abfolge von Pausen- und Sprachphasen darstellen. Um Störeffekte durch Rauschen oder durch kurze Passagen, die leise gesprochen werden, zu minimieren, werden zu kurze aktive Phasen als Störung interpretiert und aus dem Strom entfernt. Zu kurze Pausen werden als Fortsetzung der Aktivität angesehen und aufgefüllt, vgl. Abbildung 5-16.

Einfaches Sprachmodell

Brady [22] analysierte, wie groß die Mindestlängen für aktive Phasen und für Pausen sein müssen, damit der durch den Spracherkenner erzeugte Strom mit dem subjektiven Hörempfinden von Menschen übereinstimmt. Weiterhin untersuchte er, wie lange die Pausen- und die Aktivitätsphasen bei realen Telefongesprächen sind und welcher Verteilung die Phasenlänge folgen.

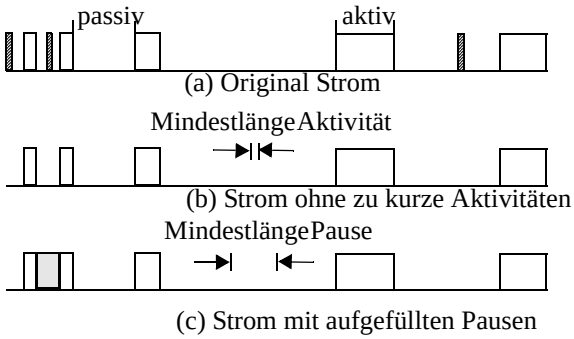


Abbildung 5-16 **Prinzip der Sprachaktivitätserkennung**

Dabei wurde ein erstes, einfaches Modell einer Sprachquelle entwickelt, bei dem jedoch keinerlei Rückkoppelungseffekte berücksichtigt wurden. Eine solche Quelle hat die beiden Zustände *on* und *off*, die der aktiven bzw. passiven Phase entsprechen, siehe Abbildung 5-17 und Abbildung 5-18. Die Zustandswechsel werden durch Poisson-Prozesse getriggert. Allerdings ist zu beachten, daß der Zustand *on* mindestens solange eingenommen sein muß, wie es die Schranke *throwawaytime* vorgibt. Kürzere Aufenthaltsdauern wären eliminiert und dem Zustand *off* zugeordnet worden.

```
process source;
procedure minimum referenced;
dcl   toon     Negexp,
      tooff    Negexp,
      nexton   Real,
      nextoff  Real,
      nextsig  Real,
      ergebnis Real,
      n        Natural,
      packet   framepara;
```

```
start;
task  tooff := init(tooff,meanactivity),
          toon  := init(toon,meanpause),
          n := 1;
lab1: task nexton := sample(toon);
 decision nexton < fillintime;
        (true) : join lab1;
        (false): nextstate off;
enddecision;
```

```
state off  /*## awake nexton ##*/;

input none;
task nextoff := sample(tooff);
decision nextoff < throwawaytime;          /* zu kurze Aktivität? */
      (true) : task nexton := nextoff;
                 nextstate -;
      (false): task nextsig := 1.0 / samplerate;
                task ergebnis := call minimum(nextsig,nextoff);
       nextstate on;
enddecision;
```

Abbildung 5-17 **Prozeß *source;* Variablen, Starttransition und Zustand *off***

```
state on /*## awake ergebnis ##*/;
input none;
decision ergebnis = nextsig;
    (false):  task nexton := sample(toon);
              decision nexton < fillintime;
                    (false): decision n = 1;
                                (false): output frame(packet);
                                         task packet!nrofsamples := 0,
                                              n := 1;
                                 (true):  task n := 1;
                             enddecision;
                              nextstate off;
                       (true) : task ergebnis := call minimum(nexton,nextsig);
                                 nextstate on;
              enddecision;
(true) :  task nextoff := nextoff - nextsig;
          decision n = 1;
               (true): task packet!timestamp := now,
                       packet!nrofsamples := packet!nrofsamples +1,
                        n := n + 1;
              (false): decision n = paketsize;
                          (true) : task packet!nrofsamples :=
                                          packet!nrofsamples +1;
                                   output frame(packet);
                                  task packet!nrofsamples := 0,
                                       n := 1;
                                                packet!nrofsamples + 1,
                          (false): task packet!nrofsamples :=
                                       n := n + 1;
                                enddecision;
          enddecision;
          task nextsig := 1.0 / samplerate;
          task ergebnis:= call minimum(nextsig,nextoff);
          nextstate on;
 enddecision;
endprocess;
```

Abbildung 5-18 **Prozeß *source* Zustand *on***

Gleiches gilt für den Zustand *off* und die Schranke *fillintime*. Daraus folgt nun, daß eine solche Quelle kein exakter Interrupted Poisson Prozeß (IPP) ist, sondern die Darstellung von Sprachquellen durch IPPs lediglich als Approximation anzusehen ist. Diese Approximation ist allerdings für viele Anwendungen hinreichend genau und erlaubt eine analytische Lösung des Modells.

Die in einer Aktivitätsphase anfallenden Abtastwerte werden nicht direkt und einzeln übertragen, sondern zu Paketen zusammengestellt. Die Anzahl der Abtastwerte pro Paket sollte nicht zu gering sein, um den zusätzlichen Aufwand, der durch die Paket-Header entsteht, zu minimieren. Allerdings sollten auch nicht zuviele Abtastwerte in einem Paket übertragen werden, um die Störungen durch beschädigte Pakete zu minimieren. Typischerweise befinden sich auf etwa 30-50 Abtastwerte in einem Paket. Falls die Quelle in den Zustand *off* wechselt, werden aber unvollständig gefüllte Pakete abgesendet, um den isochronen Charakter des Verkehrsstromes nicht unnötig zu stören.

Der QSDL-Prozeß *source* (siehe Abbildung 5-17) bildet das Verhalten einer durch Sprachaktivitätserkennung kontrollierten Quelle nach. Die Quelle ist durch folgende Parameter einstellbar:

- *samplerate*, Anzahl Abtastungen pro Sekunde
- *quantum*, Anzahl der Quantisierungsstufen
- *packetsize*, Anzahl von Abtastwerten pro Paket
- *fillintime*, minimale Länge für Pausenintervalle
- *throwawaytime*, minimale Länge für Aktivitätsintervalle
- *meanactivity*, mittlere Länge je Aktivitätsphase
- *meanpause*, mittlere Länge je Pausenphase

Der durch eine solche Quelle erzeugte Paketstrom und die verwendeten Werte für die Parameter sind in Abbildung 5-19 dargestellt.

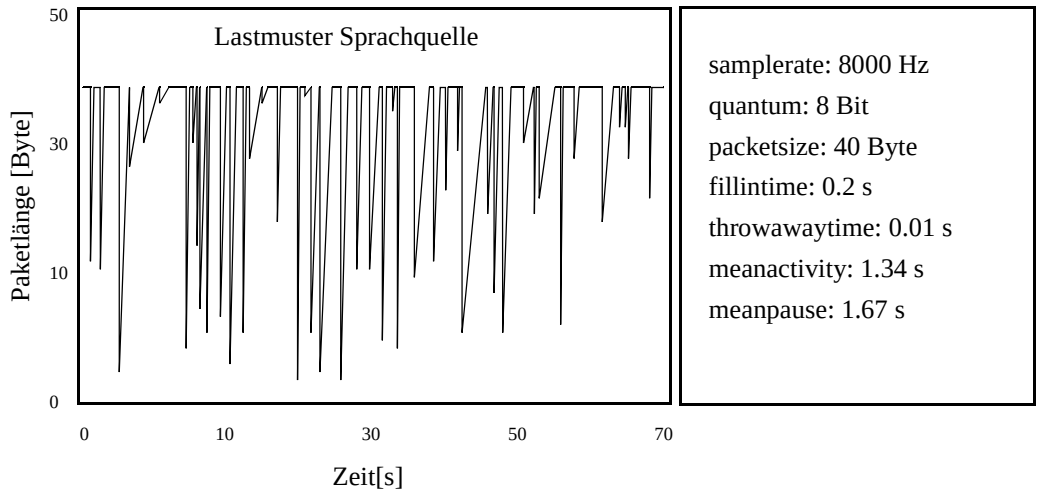


Abbildung 5-19 **Lastmuster Sprachquelle ohne Kopplung**

Dieses einfache Lastmodell berücksichtigt keine Rückkoppelungseffekte durch die Gesprächspartner. So wird der Redefluß eines Menschen nicht nur durch seine Sprechgewohnheiten beeinflußt, sondern auch maßgeblich durch das Verhalten des Partners. Brady entwickelte daher ein erweitertes analytisches Modell, in das weitere Zustände als Verfeinerungen bereits im einfachen Modell existierender Zustände integriert wurden [22]. Falls beide Sprecher pausieren, wird zusätzlich beachtet, welcher der beiden zuletzt aktiv war, da die Übergangsraten in die Zustände *Ich spreche alleine* bzw. *Ich höre zu* davon abhängig sind, welcher der beiden Sprecher zuletzt aktiv war. Gleiches gilt für den Fall, daß ein Sprecher den anderen unterbricht, vgl. Abbildung 5-20.
Die durch gestrichelte Kanten eingezeichneten Transitionen werden durch das Verhalten des Gegenübers, das durch den Sprecher beobachtbar ist, ausgelöst. Alle anderen Transitionen werden durch das Verhalten des Sprechers selbst ausgelöst und durch einen Poisson-Prozeß gesteuert. Die β^k_{Ruhe} sind Übergangsraten von *aktiv* nach *passiv*, während die α^k_{Rede} die Übergänge von *passiv* nach *aktiv* quantifizieren.

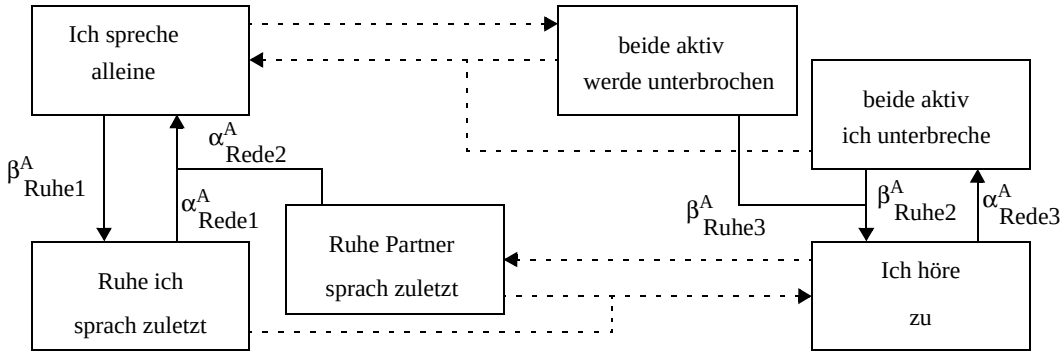


Abbildung 5-20 **6-Zustandsmodell eines Sprechers nach Brady**

Sprachmodell mit Rückkopplung

Im folgenden wird nun ein QSDL-Prozeß spezifiziert, der in seinem zeitlichen Verhalten dem eines Sprechers entspricht, der durch seine eigenen Sprechgewohnheiten, aber auch durch das Verhalten des Gegenüber beeinflußt wird. Der durch diesen Prozeß erzeugte Laststrom entspricht dem einer durch Sprachaktivitätserkennung kontrollierten Quelle, wobei die versendeten Signale in Pakete verpackte Abtastwerte repräsentieren.

Es wird weiterhin vorausgesetzt, daß die beiden Sprecher direkt zueinander sprechen, d.h. die Signale erfahren keine nennenswerte Verzögerung, die das zeitliche Verhalten beeinflussen. Der Zustandsautomat sowie die Ereignis-Trigger sind in Abbildung 5-21 dargestellt.

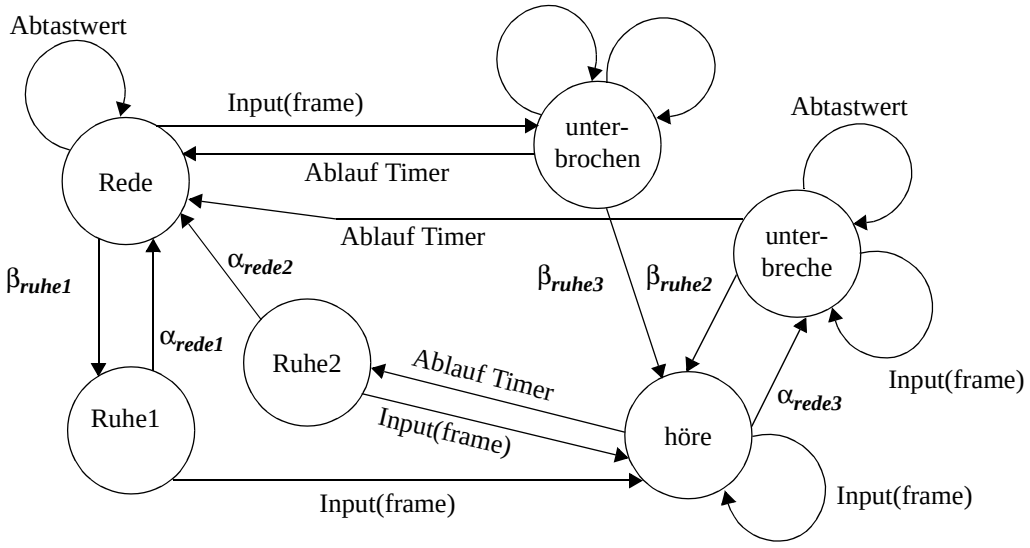


Abbildung 5-21 **Zustandsautomat und Trigger QSDL-Prozeß *Sprecher***

Die Parameter α und β sind wie im Modell von Brady Übergangsraten zwischen den *aktiv*- und *passiv*-Zuständen des Sprechers. Das Verhalten des Gesprächspartners ist durch zwei verschie-

dene Ereignisse beobachtbar. Wenn ein Paket mit Abtastwerten ankommt (*Input(frame)*), kann daraus die Aktivität des Partners abgeleitet werden. Falls dieser bereits als aktiv erkannt ist, wird der momentane Zustand beibehalten, andernfalls wird ein Zustandswechsel durchgeführt. Der Nachfolgezustand beschreibt dann einen Zustand, in dem der Sprecher sein Verhalten nicht geändert hat und der Gesprächspartner wieder aktiv wird.

Der Ablauf des Timers bedeutet, daß seit n Zeiteinheiten keine Pakete vom Partner eingetroffen sind. Dies wird als Zeichen von Passivität gewertet, der Prozeß wechselt in jedem Fall seinen Zustand, vergleiche Abbildung 5-21. Genau wie im einfachen Sprechermodell sind die beiden Sprechphasen in ihrer minimalen Länge durch die Schranken *fillintime* und *throwawaytime* begrenzt.

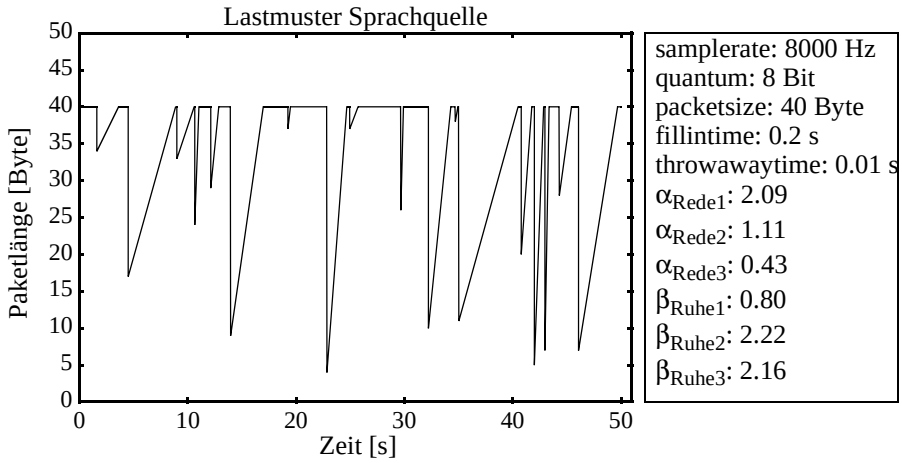


Abbildung 5-22 **Lastmuster Sprachquelle mit Kopplung**

Ein Lastmuster des rückgekoppelten Modells sowie die verwendeten Parameter sind in Abbildung 5-22 dargestellt. Die Parameter sind aus einer Reihenuntersuchung mit 32 Gesprächspaaren in der Studie von Brady hervorgegangen. In dieser Studie sind die Übergangsraten in Gesprächen zwischen Männern, Frauen und gemischten Gesprächspaaren veröffentlicht worden. Aus diesen Raten werden gewichtete Mittelwerte gebildet und in das QSDL-Modell übernommen. Im Gegensatz zum einfachen Modell fällt auf, daß das Lastmuster der rückgekoppelten Quelle deutlich regelmäßiger ausfällt.

In einer Untersuchung mit QUEST wurde das Verhalten von Sprechern, die entsprechend des rückgekoppelten Modells agieren, untersucht. Dabei wurde die Übertragungsverzögerung zwischen den beiden Sprechern variiert, um den Einfluß der Verzögerung auf das entstehende Lastprofil zu untersuchen. In einer Experimentserie mit QUEST ist die Verzögerung der ausgesendeten Signale von 125 Millisekunden bis 5 Sekunden gesteigert worden. In Abbildung 5-23 ist das zeitliche Verhalten der Quelle bei einer Verzögerung von 500 Millisekunden dargestellt. Auf den ersten Blick lassen sich keine markanten Unterschiede im Vergleich zur verzögerungsfreien Übertragung erkennen. Lediglich die Länge der Pausenintervalle scheint vergrößert zu sein.

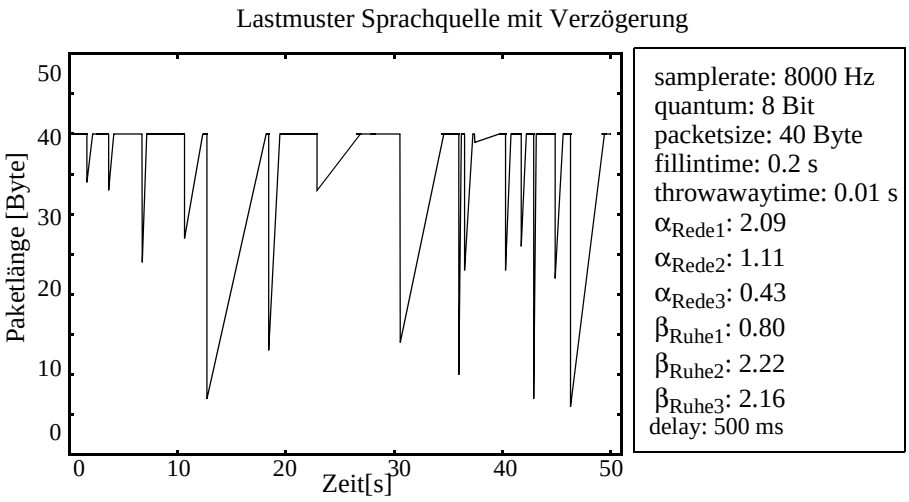


Abbildung 5-23 **Lastmuster Sprachquelle mit Kopplung und Verzögerung**

Dieser Eindruck kann durch die Betrachtung der Zustandshäufigkeiten erhärtet werden, vgl. Abbildung 5-24 links. Während der Anteil der Zustände, in denen beide Quellen schweigen, im verzögerungsfreien Fall bei etwa 30% liegt, steigt er auf etwa 42% bei einer Verzögerung von 5 Sekunden.

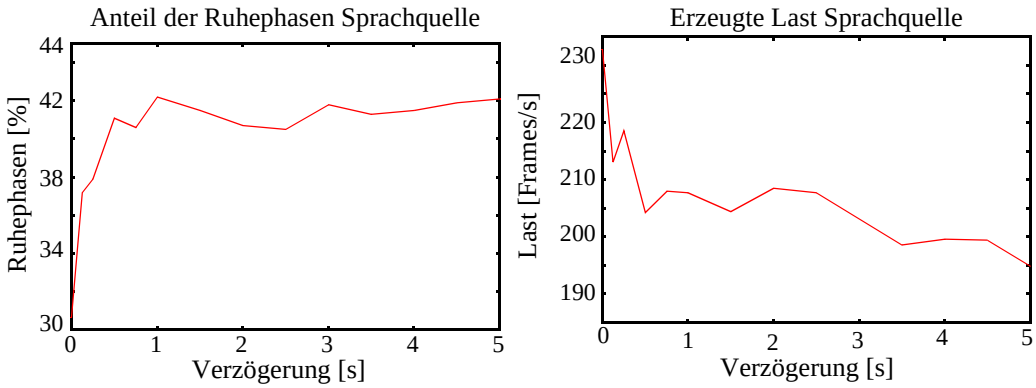


Abbildung 5-24 **Anteil Ruhephasen und erzeugte Last der verzögerten Quelle**

Daraus resultiert eine geringere Anzahl von erzeugten Rahmen pro Zeit, was in der Endkonsequenz dazu führt, daß für die Erzeugung einer bestimmten Informationsmenge ein größerer Zeitraum benötigt wird, siehe Abbildung 5-24 rechts. Dies führt zu einer längeren Dauer der Sprachverbindung, die wiederum eine längere zeitliche Belastung des Übertragungsmediums nach sich zieht.

Aus diesen Experimenten kann gefolgert werden, daß bei der Modellierung von Lastquellen für digitalisierte Sprache die Ende-zu-Ende-Verzögerung eine große Rolle spielt und daher in einem Lastmodell berücksichtigt werden muß.

Damit soll die Modellierung von Sprachquellen abgeschlossen werden und im folgenden das Augenmerk auf Verkehrsquellen für die beiden wesentlichen Anwendungen zukünftiger Multimedia-Systeme, Audio und Video gelenkt werden.

5.2.2 Lastquellen für Audio-Verkehr

Für die Speicherung und Verarbeitung von Audiodaten, genauer Musikdaten, hat sich unter anderem der CD-Standard, der mit einer Abtastrate von 44,1 kHz und einer Quantisierung von 16 Bit arbeitet, durchgesetzt. Die aus diesen Werten resultierende Datenrate ergibt sich zu 705,6 kBit/s. Der zweite defacto-Standard ist Digital Audio Tape (DAT), der wahlweise mit 32 kHz oder 48 kHz abtastet. Auch hier wird mit 16 Bit quantisiert, so daß sich ein Datenstrom von 512 kBit/s bzw. von 768 kBit/s ergibt. Diese Werte gelten je Musikkanal, was bei Stereosignalen zu einer Verdoppelung der benötigten Bandbreiten führt.

Als Standard zur Audiokodierung bzw. Audiokompression sind MPEG1 [23], [69] bzw. MPEG2 von der I*nternational Standardization Organization* in Zusammenarbeit mit *International Electrotechnical Commission* (ISO/IEC) verabschiedet worden.

Die Grundidee besteht darin, die benötigten Datenmengen zu reduzieren, ohne die subjektiv hörbare Qualität zu verschlechtern. MPEG nutzt dazu Eigenschaften des menschlichen Gehörs. Durch sogenannte psychoakustische Modelle wird die Kodierung des Musiksignals gesteuert, siehe Abbildung 5-25.

Die Kompression erfolgt durch Transformation des Audiosignals vom Zeitbereich in den Frequenzbereich durch eine *Fast Fourier Transformation* (FFT). Zusätzlich wird das transformierte Signal in 32 disjunkte Frequenzbänder zerlegt. Diese Quantisierung wird durch psychoakustische Modelle gesteuert.

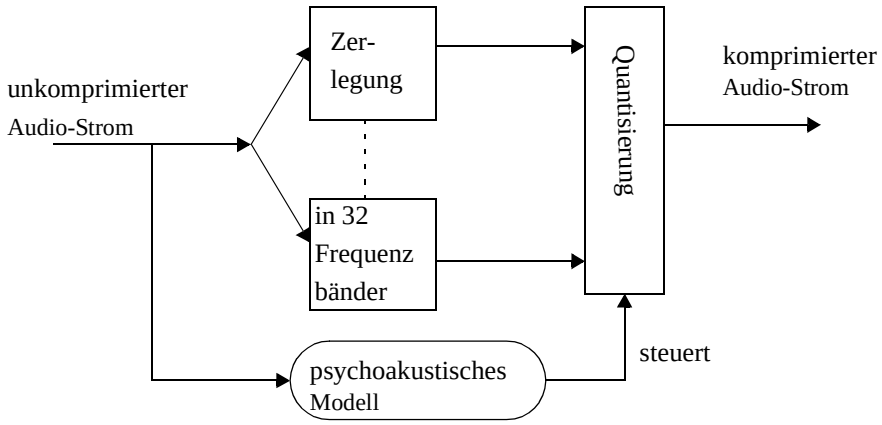


Abbildung 5-25 **MPEG-Audio-Kodierung**

Dabei werden Maskierungseffekte genutzt. Unter einem Maskierungseffekt versteht man, daß leise Töne durch vorherige laute Töne in einem ähnlichen Frequenzspektrum überdeckt werden und somit vom menschlichen Gehör nicht wahrgenommen werden. Zusätzlich werden in jedem Frequenzband die Rauschpegel bestimmt. In Bändern mit hohem Rauschanteil kann mit geringerer Auflösung quantifiziert werden. Durch diese Maßnahmen können die benötigten Bandbreiten im Verhältnis 1:6 bis 1:7 reduziert werden, ohne daß subjektive Qualitätseinbußen auftreten.

Im MPEG-Standard sind drei verschiedene Qualitätsstufen definiert worden, die dort als Layer bezeichnet werden. Mit jeder Stufe steigt der Kodieraufwand und damit die zeitliche Verzögerung, die das Signal im Kodierer erfährt. Gleichzeitig sinkt die benötigte Bandbreite, um Signale mit derselben Qualität zu übertragen, bzw. bei gleicher Bandbreite steigt die Signalqualität.

In der einfachsten Stufe, dem Layer-I werden Bitraten von 32 kBit/s (mono) bis zu 448 kBit/s (stereo) unterstützt. Wie in allen Layern stehen wahlweise die aus dem CD bzw. DAT-Standard bekannten Abtastfrequenzen von 44,1 kHz bzw. 32 kHz oder 48 kHz zur Verfügung. Daraus ergibt sich, daß bei einer Abtastfrequenz von 32 kHz und einer Bitrate von 32 kBit/s mit nur einem Bit quantisiert wird, während bei einer Bitrate von 448 kBit/s noch 14 Bit verwendet werden. Allerdings ist zu beachten, daß es sich dabei nur um Mittelwerte handelt, da die Signale aufgrund der Maskierungseffekte nicht mehr linear quantisiert werden. Für Stereosignale, die mit guter Qualität (CD-Qualität) übertragen werden, benötigt man in einem Layer-I Kodierer 256 kBit/s bis 384 kbit/s, was einer Quantisierung von 6 bis 12 Bit entspricht.

Layer-II bietet Bitraten von 32 kbit/s bis zu 384 kBit/s und benötigt für eine gute Qualität Bitraten von 128 kBit/s bis zu 256 kBit/s. Diese können auf 192 kBit/s reduziert werden, wenn der joint stereo-Modus gewählt wird. Dabei wird nicht jeder Stereokanal separat übertragen, sondern die Interkanalabhängigkeiten werden berechnet und ausgenutzt. In diesem Modus wird der hochfrequente Anteil beider Signale (über 2kHz) zusammen mit einem Skalierungsfaktor für die Quantisierung übertragen. Der MPEG1-Layer-II Standard gilt als guter Kompromiß zwischen dem Aufwand für die Kodierung und erzielbarer Qualität und ist daher der am meisten verwendete Ansatz. Aufgrund der definierten Bitraten ergeben sich Quantisierungsstufen von 2 bis 12 Bit bei einer Abtastfrequenz von 32 kHz. Für Stereosignale in CD-Qualität werden 4 bis 8 Bit benötigt. Bei einer Abtastfrequenz von 48 kHz reduziert sich diese Anzahl auf 3 bis 5 Bit, dafür werden aber die Abstände zwischen zwei Abtastwerten reduziert.

Layer-III erfordert den höchsten Aufwand, liefert dafür aber die beste Qualität. Zusätzlich zur Zerlegung in einzeln quantisierbare Subbänder ermöglicht der Layer-III eine nachgeschaltete Entropie-Kodierung, wobei eine Huffman-Kodierung verwendt wird. Als einziger Layer ermöglicht Layer-III eine variable Bitrate, d.h. ein nach Layer-III codierter Signalstrom kann zu beliebigen Zeitpunkten die Bitrate und somit die Quantisierung ändern. Bereits mit einer Bitrate von 128 kbit/s werden ausgezeichnete Qualitäten erzielt.

MPEG kodierte Audiodaten bestehen aus einer Folge von sog. Frames. Jeder Frame enthält eine feste Anzahl von Abtastwerten. Im Falle einer Layer-I-Kodierung besteht ein Frame aus 384 Abtastwerten, während Layer-II- und Layer-III-Frames aus 1152 Abtastwerten bestehen. Daraus ergeben sich je nach Abtastfrequenz Ausschnitte aus dem Audiostrom von 12 ms (bei 32 khz) bis zu 8 ms (bei 48 kHz) in Layer-I und von 36 ms bis zu 24 ms in Layer-II und Layer-III.

In allen Fällen bleibt der isochrone Charakter eines MPEG-kodierten Audiostromes erhalten, da zu festen Zeitpunkten eine feste Anzahl von Abtastwerten zu einem Audio-Frame zusammengefaßt werden. Da auch die Zielbitraten nur in einem Layer-III-Strom gewechselt werden können, stellt sich der kodierte Ausgangsstrom in den meisten Fällen als streng periodischer Strom dar.

Im folgenden wird nun ein QSDL-Modell vorgestellt, das das zeitliche Verhalten eines MPEG-kodierten Audiostroms nachbildet. Das Modell ist soweit parametrisiert, daß durch entsprechende Wahl der konkreten Parameterwerte jeder dem Standard entsprechende Ausgangsstrom erzeugt werden kann.

Die Parameter, durch die ein MPEG-Kodierer eingestellt werden kann, zerfallen in zwei Gruppen. Da sind zum einen die Arbeitsmodi und Qualitätsstufen und zum anderen die Abtastraten und die gewünschte Zielbandbreite. Im MPEG-Standard sind ein Mono-Modus und 3 Stereo-Modi festgelegt. Darüberhinaus existieren die drei Layer. Faßt man die 3 Stereo Modi zu einem sog. *dual-channel-mode* zusammen, ergibt sich für die MPEG-Audioquelle ein QSDL-Modell mit 6-Zuständen. Berücksichtigt man weiterhin, daß im Layer-I-Modus keine Unterschiede bezüglich der unterstützten Zielbandbreiten zwischen Ein- und Zweikanal-Modus existieren, so läßt sich der Zustandsautomat auf 5 Zustände reduzieren, vgl. Abbildung 5-26.

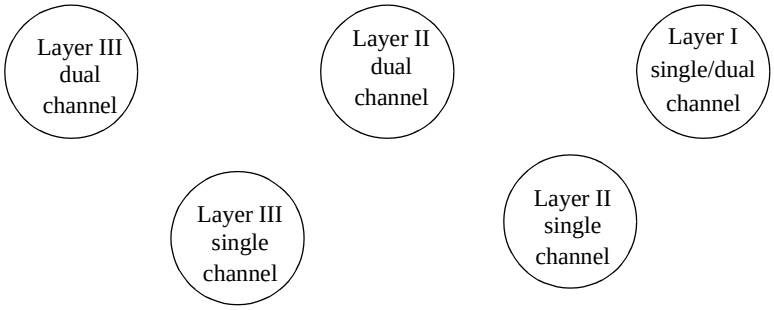


Abbildung 5-26 **Zustände einer MPEG-Audio-Quelle**

In allen Zuständen werden wahlweise die Abtastfrequenzen 32 kHz, 44.1 kHz und 48 kHz nach MPEG1-Standard bzw. die Frequenzen 16 kHz, 22,05 kHz und 24 kHz nach MPEG2-Standard unterstützt. In allen Fällen wird eine feste Anzahl von Abtastwerten in einem MPEG-Audio-Frame übertragen. Die Frame-Größe und der Abstand zwischen zwei Frames läßt sich wie folgt berechnen:

Sei N die Anzahl der Abtastwerte pro Frame und f_s die Abtastfrequenz. Dann ergibt sich der Abstand zwischen zwei Frames zu $T_F = \frac{1}{f_s} \cdot N$ [s].

Mit R wird die gewünschte Zielbandbreite in Bit/s bezeichnet. Falls ein Stereosignal kodiert wurde, ergibt sich daraus eine Bandbreite von R/2 pro Kanal.

Die Größe eines MPEG-Audioframes berechnet sich zu $S_F = R \cdot T_F$ [Bit]. In Abbildung 5-27 sind für einige ausgewählte Parameterkombinationen die resultierenden Lastmuster dargestellt.

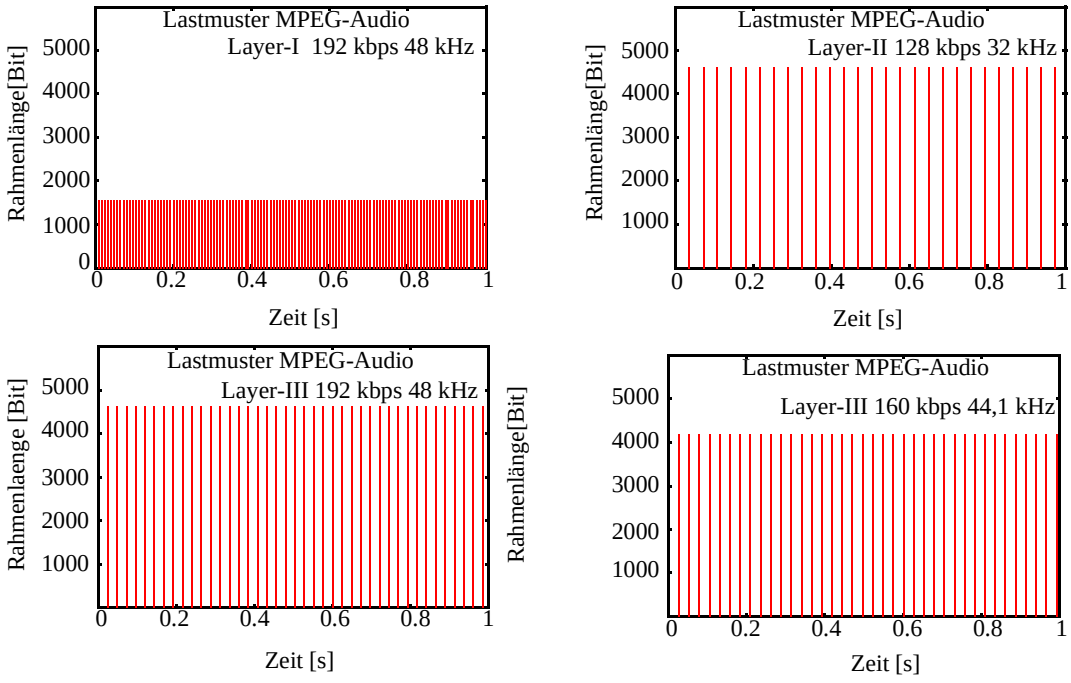


Abbildung 5-27 **Lastmuster MPEG-Audio-Quellen**

Man sieht sehr gut, daß im Layer-I viele kleine Rahmen erzeugt werden, während zum Beispiel im Layer-III wenige große Rahmen erzeugt werden. In allen Fällen wird durch die MPEG-Kodierung ein streng gleichmäßiger Datenstrom erzeugt, was MPEG-Audioquellen sehr gut für unterliegende Netze kalkulierbar macht.

5.2.3 Lastquellen für Video-Verkehr

Einen wesentlichen Einfluß auf die Leistungsfähigkeit zukünftiger Multimedia-Netze werden Video-Anwendungen haben, da sie eine enorme Anforderung an die Kapazität von Zwischenknoten, Übertragungseinrichtungen und Endsystemen haben. Zur Reduktion der Datenmenge von Videofilmen hat die **M**oving **P**icture **E**xpert **G**roup der ISO/IEC den nach ihr benannten MPEG-Standard entwickelt. MPEG erlaubt ein sogenanntes *multi layer coding*, d.h. der Videostrom wird in einen Basisstrom und mehreren Ergänzungsströmen codiert. Während der Basisstrom alle wesentlichen Informationen enthält, um den Strom zu dekodieren, enthalten die Ergänzungsströme lediglich Informationen, die die Qualität verbessern. Der MPEG-Standard verwendet drei verschiedenen Rahmentypen.

1. *Intra-Frames* (I-Frames) nutzen lediglich Redundanzen innerhalb eines Bildes aus und enthalten keinerlei Bezüge auf andere Bilder des Videostromes.
2. *Forward-Predicted Frames* (P-Frames) nutzen neben den im Bild vorhandenen Redundanzen noch die Tatsache aus, daß sich im Vergleich zum letzten I-Frame möglicherweise nur sehr wenig geändert hat. Die Inhalte von P-Frames werden durch Differenzbildung zum vorgehenden Referenzbild (I-Frame) berechnet. P-Frames werden ebenfalls als Referenzbilder verwendet.
3. *Bidirectional-Predicted Frames* (B-Frames) entstehen durch Differenzbildung zum vorhergehenden und zum nachfolgenden Referenzbild. Sie erzielen dadurch die höchsten Kompressionsraten

Jede Anwendung kann selbst das Verhältnis der 3 Rahmentypen festlegen. Die Rahmen werden in einem deterministischen Muster, der sog. Group of Pictures (GoP) übertragen. Die Länge einer solchen GoP N wird durch den Abstand zweier I-Frames festgelegt. Das Muster ist zusammen mit dem Verhältnis zwischen P-Frames und B-Frames M sowie der Länge der GoP festgelegt. Für $N = 12$ und $M = 2$ wird das folgende Muster festgelegt:

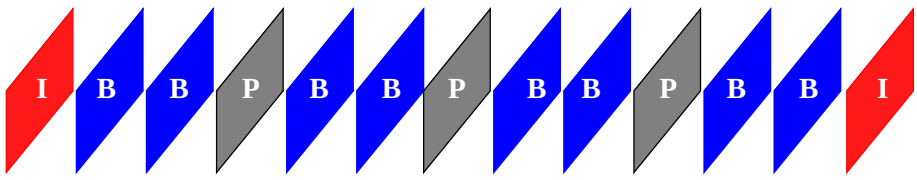


Abbildung 5-28 **Rahmenfolge eines MPEG-codierten Videostromes**

Das Lastverhalten einer MPEG-Videoquelle ist festgelegt durch das Lastmuster, d.h. durch die GoP-Länge, das Verhältnis von B- zu P-Frames, den Abständen zwischen zwei benachbarten Rahmen, sowie der Größe jedes kodierten Bildes. Da eine Kodierung durch entsprechende Hardware in Realzeit durchgeführt werden kann, ergibt sich der Abstand zwischen zwei Rahmen

durch die Anzahl von Bildern pro Sekunde, mit der der Film abgespielt wird. Für einen Standard-Videofilm ergibt sich dieser Abstand zu vier Millisekunden bei 25 Bildern pro Sekunde.

Den entscheidenden Einfluß auf alle beteiligten Komponenten hat die Größe der Rahmen. In den Arbeiten von Rose [94] und [95] sind statistische Eigenschaften von MPEG-codierten Videofilmen untersucht und analytische Verkehrsmodelle für MPEG-Ströme entwickelt worden. Diese lassen sich für eine simulative Leistungsbewertung so nicht nutzen, allerdings können aus den statistischen Eigenschaften Verhaltensmuster abgeleitet werden, die in QSDL-Quellenmodelle überführt werden können.

Eine MPEG-Quelle hat prinzipiell drei Zustände, in denen sie I-, P- oder B-Frames versendet, vgl. Abbildung 5-29. Die Pausen in den Zuständen sind konstant. Die Quelle wechselt aus dem Zustand *sende_I-Frame* immer in den Zustand *sende_B_Frame*. In diesem Zustand sind Wechsel in die Zustände *sende_P-Frames* oder *sende_I_Frames* möglich. Die Quelle kann aber auch in diesem Zustand bleiben. Zustandswechsel werden immer nach dem Senden eines Rahmens durchgeführt.

Neben diesen drei Zuständen werden noch zwei Zähler c_1 und c_2 verwaltet. Der Zähler c_1 hält die Gesamtzahl der gesendeten Rahmen seit Verlassen des Zustandes *sende-I-Frame* fest und der Zähler c_2 speichert die Anzahl der Rahmen, die im Zustand *sende-B-Frame* versendet werden, ohne den Zustand zu verlassen. Solange $c_2 < M$ gilt, bleibt die Quelle im Zustand *sende_B_Frames*. Erreicht c_2 den Wert *m*, aber noch nicht den Wert *N*, so geht die Quelle in den Zustand *sende P_Frame* und c_2 wird zurückgesetzt. Erreicht der Zähler c_1 den Wert *n*, so wechselt die Quelle in den Zustand *sende_I_Frame* und beide Zähler werden zurückgesetzt. Mit jedem Sendeereignis wird c_1 um eins erhöht, während c_2 nur durch das Senden von *B-Frames* um eins erhöht wird.

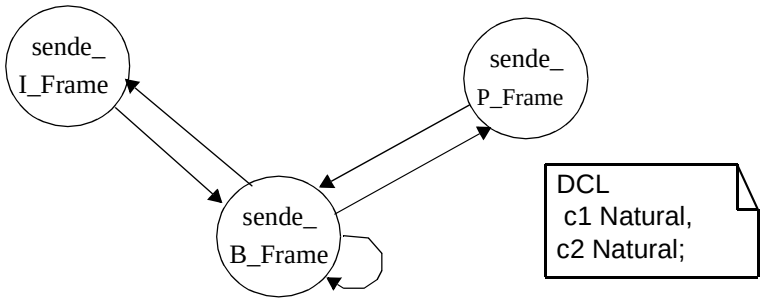


Abbildung 5-29 **Zustandsautomat und lokale Variablen für MPEG-Quelle**

Die so spezifizierte Quelle erzeugt ein Lastmuster, wie es in Abbildung 5-28 dargestellt ist. Durch die Analyse von realen Videos hat sich ergeben, daß die Größe von B-Frames annähernd $\log_2$-normalverteilt ist. Im Gegensatz dazu ist die Rahmengröße sowohl von I-Frames als auch von P-Frames Gamma-verteilt. Die Gamma-Verteilung ist eine Verallgemeinerung der Erlang-Verteilung, erlaubt aber positive reelle Werte für b.

Ihre Dichtefunktion f ist gegeben durch $f(x) = \frac{\left(\frac{x}{a}\right)^{b-1} \cdot e^{\frac{-x}{a}}}{a \cdot \Gamma(b)}$.

Gamma-verteilte Zufallszahlen lassen sich sehr einfach generieren, wenn man berücksichtigt, daß die Summe von b exponential verteilten Zufallszahlen einer Gamma-Verteilung genügen.

Falls $u_i \sim U(0, 1)$ dann gilt: $\gamma(a, b)) \sim -a \cdot \ln \prod_{i=1}^{b} u_i$

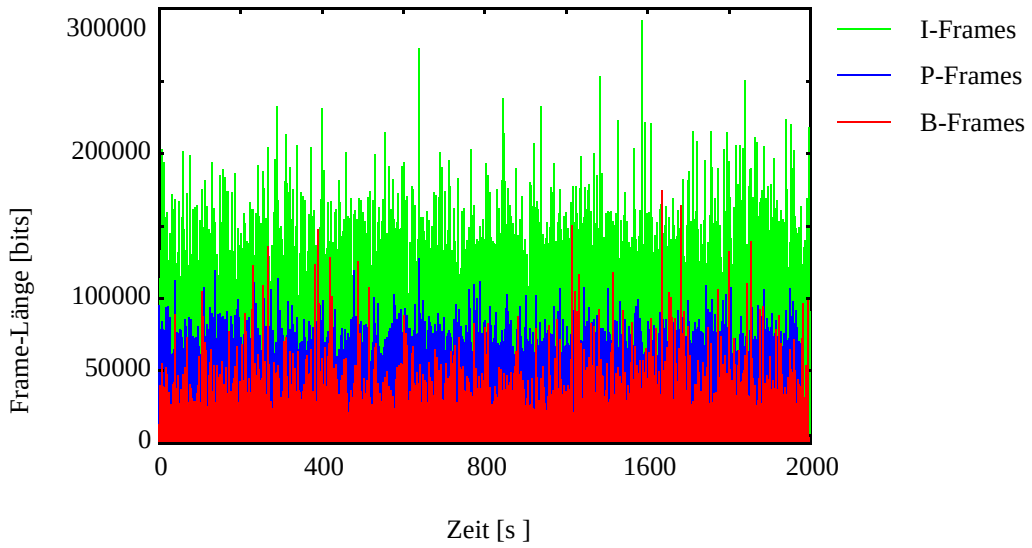


Abbildung 5-30 **Lastmuster MPEG-Video-Quelle**

Die so modellierte Lastquelle erzeugt ein Lastmuster entsprechend Abbildung 5-30. In der Abbildung sind die Längen der von einer MPEG-Quelle ausgesendeten Frames dargestellt. Es läßt sich gut erkennen, wie durch die Kompressionstechnik die Größe der B-Frames reduziert werden kann. So weisen diese Frame-Größen von etwa 50 kBit auf, während I-Frames Größen von etwa 140 kBit aufweisen. Das Lastmuster der modellierten Quelle entspricht den Lastmustern, wie sie in den Studien von Rose von realen Videofilmen vorgestellt wurden.

5.3 Zusammenfassung Kapitel 5

Eine modellgestützte Leistungsanalyse von Kommunikationsprotokollen setzt die Existenz von Lastmodellen für die Anwendungen voraus, die sich auf die Protokolle abstützen. Daher sind in diesem Kapitel für die wichtigsten Anwendungen im multimedialen Internet Modelle für Lastprozesse entwickelt worden. Zu diesen Anwendungen zählen sowohl diskrete Datenströme wie FTP, Telnet oder WWW, als auch kontinuierliche Datenströme wie Sprache, Audio und Video.

Damit der Vorteil des homogenen Beschreibungsparadigmas, das durch die Sprache QSDL erreicht wurde, erhalten bleibt, müssen die Lastmodelle ebenfalls mit Hilfe der QSDL-Konstrukte beschrieben werden.

Daher sind alle Lastprozesse automatenbasiert als Mehrzustandsquellen spezifiziert worden, wobei die Mechanismen von QSDL für zeitbehaftete Zustände zur zeitlichen Steuerung der Zustandswechsel und Sendeereignisse verwendet wurden. Damit wird eine zeitgetreue Reaktion der Lastprozesse gewährleistet, die bei Verwendung von SDL-Timern wegen der Timer-Semantik nicht möglich ist. Die Eignung des gewählten Ansatzes wird verdeutlicht durch die Überein-

stimmung der Lastmuster realer Anwendungen mit den Lastmustern der in dieser Arbeit entwickelten Quellprozesse. Über die reine Spezifikation hinaus konnte durch die Analyse der Modelle mit Rückkopplung gezeigt werden, wie wichtig die Berücksichtigung der zeitlichen Verzögerung dieser Rückkopplung ist. Somit konnten verbesserte Modelle für diese Anwendungsfälle erstellt werden.

Die in diesem Kapitel spezifizierten Quellenprozesse werden in den nachfolgenden Anwendungsstudien benutzt, um die Analyse von Transportsystemen unter realitätsnahen Belastungen durchführen zu können.

Kapitel 6

Medienmodellierung mit QSDL

Im Kontext von Kommunikationsprotokollen hat sich die Aufteilung komplexer Aufgaben in Schichten gemäß des von der *International Standardization Organisation* (ISO) entwickelten Referenzmodells für *Open System Interconnection* (OSI) etabliert [16]. Die Schicht i bietet für Instanzen der Schicht i+1 den (i)-Dienst an und benutzt dazu den (i-1)-Dienst, den die unter ihr liegende Schicht i-1 anbietet, vgl Abbildung 6-1.

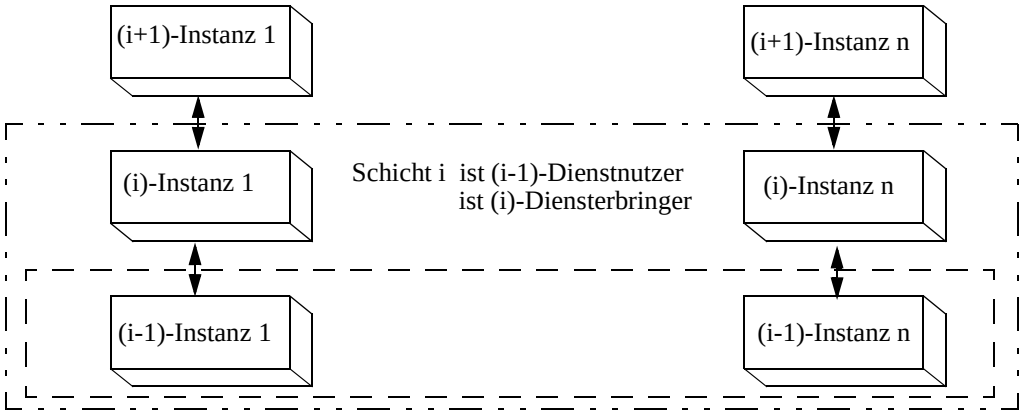


Abbildung 6-1 **Schichtenprinzip nach OSI**

Die Regeln, wie die (i)-Instanzen auf der Grundlage des (i-1)-Dienstes den (i)-Dienst erbringen, wird als Kommunikationsprotokoll bezeichnet. Während im Kontext von Protokollvalidationen und -verifikationen von den lokal benötigten Betriebsmitteln der Instanzen sowie von den quantitativen Merkmalen der unterliegenden Dienste abstrahiert werden kann, müssen diese in Leistungsbewertungsstudien berücksichtigt werden. Insbesondere die Spezifikation und Analyse physikalischer Medien und der ISO-OSI Schicht-2 sind notwendig, um realistische Leistungskennziffern komplexer, verteilter Gesamtanwendungen zu erhalten.

Auch hier sollte im Sinne eines homogenen Gesamtmodells die Beschreibung der unterliegenden Medien mit Hilfe der Methoden vorgenommen werden, mit denen auch das neu zu entwerfende System beschrieben wird. Daher werden nun für einige der gängigsten Link-Layer-Protokolle mit Hilfe des QSDL-Ansatzes Bausteine entworfen, die das funktionale und quantitative Verhalten von Links beschreiben. Diese Bausteine können dann dazu genutzt werden, Protokolle und Architekturen der höheren Schichten zu bewerten, wenn diese über die vorhandenen Link-Layer-Dienste abgewickelt werden.

6.1 Punkt-zu-Punkt-Verbindungen

Durch Punkt-zu-Punkt-Verbindungen werden zwei Knoten (Rechner) direkt verbunden. Diese Verbindungen können uni- oder bi-direktional ausgelegt werden. Weiterhin werden sie durch die maximale Paketgröße, die zur Verfügung stehenden Kanäle und deren Bandbreite, sowie durch ihre Länge und die Fehlerraten beschrieben. Punkt-zu-Punkt-Verbindungen können paket- oder leitungsvermittelt realisiert werden.

6.1.1 Paketvermittelter Simplex-Kanal

Falls zwei Knoten durch einen paketvermittelten Simplex-Kanal verbunden sind, befinden sich in den Endknoten Puffer, in denen ausgehende Pakete bzw. einkommende Pakete bis zu ihrer Weiterverarbeitung gespeichert werden können.

Pakete, die einen Knoten erreichen, werden gemäß ihrer Zieladresse in den zuständigen Ausgangspuffer gestellt. Von hier werden die Pakete gemäß ihres Eintreffens auf die Ausgangsleitung geschickt. Die Pakete werden auf dieser Ausgangsleitung übertragen und erreichen den am anderen Ende befindlichen Eingangspuffer des Zielknotens, vgl. Abbildung 6-2.

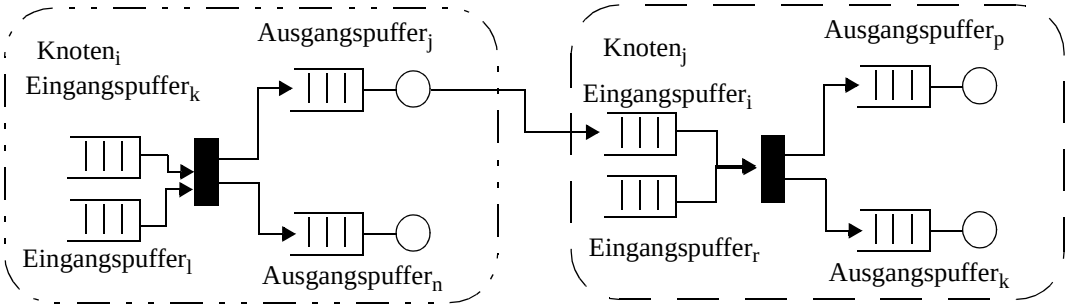


Abbildung 6-2 **Knotenmodell für paketvermittelnde Netze**

Bei dieser Übertragung können Pakete fehlerfrei oder fehlerhaft übertragen werden. Weiterhin erfahren sie zwei Arten von Verzögerungen. Die erste Verzögerung ergibt sich als Differenz der Zeitpunkte zwischen dem Aufbringen des ersten und letzten Bits eines Paketes. Für diese Zeit ist die Instanz, die die Pakete auf das Netz bringt, blockiert und kann keine weiteren Aktionen durchführen. Die Zeit, die dafür benötigt wird, ergibt sich als Quotient von verwendeter Paketlänge [Bit] und Bandbreite des Kanals [Bit/s].

Zusätzlich ergibt sich eine nahezu konstante Laufzeitverzögerung, die von der physikalischen Länge des Mediums abhängt. Diese Signalausbreitungsverzögerung ergibt sich als Quotient von Mediumlänge [m] und Signalausbreitungsgeschwindigkeit [m/s]. Für diese Zeit ist die aufbringende Instanz nicht blockiert und kann gegebenfalls weitere Pakete versenden.

In klassischen Modellierungsstudien wird die Signalausbreitungsverzögerung vernachlässigt, weil sie in der Regel um einige Zehnerpotenzen geringer ausfällt als die Übertragungsverzögerung. Im Kontext von Hochgeschwindigkeitsmedien im Bereich von 1000 MBit/s können bei Vernachlässigung der Signalausbreitungsverzögerung im Modell deutliche Abweichungen zur Realität entstehen. Bei einem Paket von einem KBit Größe und einer Kabellänge von einem Kilometer ergibt sich die Laufzeitverzögerung zu $5* 10^{-6}$ Sekunden. Unterstellt man eine Bandbreite von 1000 MBit/s, so ergibt sich eine Verzögerung von $1* 10^{-6}$ Sekunden, um das Paket vollständig auf das Netz aufzubringen. Vernachläßigt man die Laufzeitverzögerung, so ergibt sich eine Abweichung von fünf Mikrosekunden oder 83%. Wählt man hier nur die Laufzeitverzögerung, so ergibt sich eine Abweichung von einer Mikrosekunde oder 16%.

Somit wird die Notwendigkeit deutlich, beide Verzögerungen von Paketen bei der Übertragung hinreichend zu berücksichtigen und in entsprechende Leistungsbewertungsmodelle zu integrieren. Das QSDL-Modell eines Transitknotens mit einem Eingangs- und einem Ausgangskanal hat eine Struktur gemäß Abbildung 6-3.

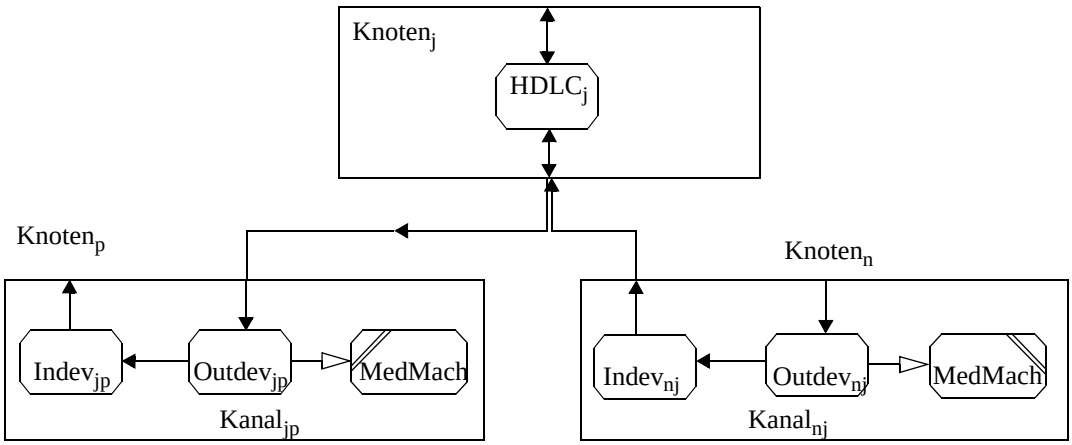


Abbildung 6-3 **Simplex-Kanal (Blockstruktur)**

Signale (Pakete), die über den Ausgangskanal von *Knoten j* an *Knoten p* gesendet werden sollen, werden vom Prozeß *HDLCj* an den Prozeß $OutDev_{jp}$ im Block $Kanal_{jp}$ gesendet. Pakete, die die maximale Paketgröße überschreiten, werden segmentiert. Der Prozeß $Outdev_{jp}$ sendet die Signale (Pakete) an den Prozeß $Indev_{jp}$ innerhalb desselben Blockes.

Die Verzögerung, die das Paket beim Aufbringen auf das Medium erfährt, wird durch den request *transmit(eingabe!inhalt),* vgl. Abbildung 6-4 an die Maschine *MedMach* modelliert

Als Bedienwunsch wird die Paketlänge in Bit verwendet, während die Bandbreite des Mediums als Bediengeschwindigkeit der Maschine eingeht. Die konstante Laufzeitverzögerung wird durch Verwendung des Output-delay-Konstrukts berücksichtigt.

```
process Indev;
dcl
  eingabe pakettype,
  fehl Boolean,
  stoerung Bernoulli;
start;
task stoerung := init(stoerung,fehlerw);
nextstate ready;
state ready;
  input Paket(eingabe);
    task fehl := sample(stoerung);
    decision fehl;
      (True) :task eingabe!fehler := ja;
      (False) :
    enddecision;
    output Paket(eingabe) via toB;
    nextstate -;
endstate;
endprocess Indev;
```

```
/*## machine MedMach;
  server 1;
  discipline fcfs;
  offers transmit: bandbreite;
endmachine; ##*/
```

```
process Outdev;
dcl eingabe Pakettype;
start;
    nextstate ready;
state ready;
  input paket(eingabe);
 /*## request transmit(eingabe!inhalt);##*/
    output Paket(eingabe)
        via net /*## delay siglaufzeit ##*/;
     nextstate -;
endstate;
endprocess Outdev;
```

Abbildung 6-4 **Prozesse und Maschinen eines Simplexkanals (QSDL/PR)**

Im Prozeß *Indev* wird lediglich entschieden, ob ein übertragenes Paket einen Fehler enthält, oder ob es fehlerfrei übertragen wird. Dazu wird im Prozeß *Indev* ein sogenanntes Bernoulli-Experiment durchgeführt. Der Aufruf der sample-Methode liefert mit der Wahrscheinlichkeit *fehlerw* den Wert true, d.h. es tritt ein Fehler auf, und mit der Wahrscheinlichkeit *1- fehlerw* den Wert false, d.h. das Paket wird fehlerfrei übertragen.

Neben der funktionalen Beschreibung kann nun für einen Simplex-Kanal das Verzögerungs- und Durchsatzverhalten in Abhängigkeit von Paketgröße und Verkehrsangebot bestimmt werden, vgl. Abbildung 6-5.

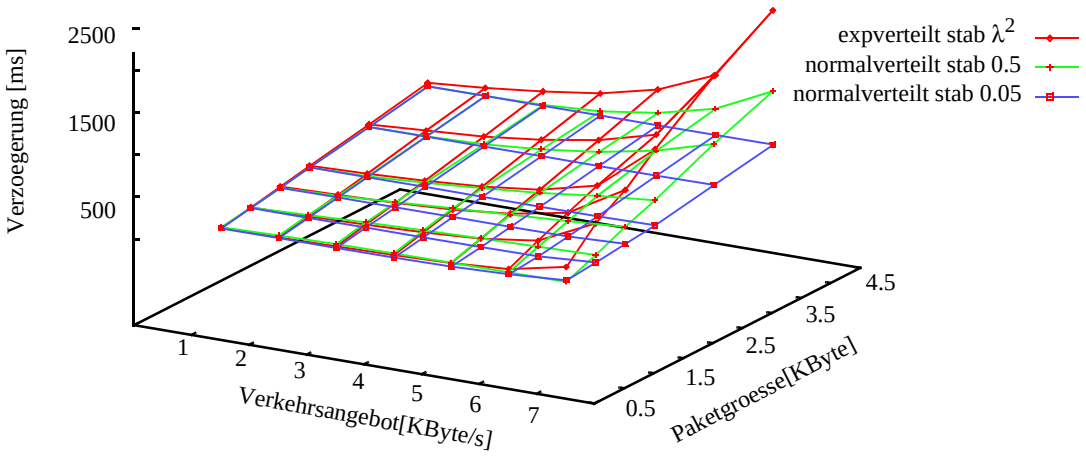


Abbildung 6-5 **Paketverzögerung Simplex-Kanal als Funktion von Paketgröße, Verkehrsangebot und Ankunftsstromverteilung**

Das Verkehrsangebot umfaßt neben einer quantitativen Beschreibung, die die mittlere Last in KByte/s enthält, auch die Verteilung, die den Verkehrsstrom beschreibt und gegebenenfalls die Standardabweichung als Maß für die Streuung der Ankünfte. So kann beim Entwurf größerer, verteilter Systeme mit Hilfe solcher Charakteristiken entschieden werden, ob bei dem zu erwartenden Lastaufkommen ein Kanal der verwendeten Art ausreichend ist und wie die Betriebsparameter am günstigsten eingestellt werden können.

In Abbildung 6-5 ist die Kanalverzögerung eines Simplex-Kanals mit einer Bandbreite von 64 KBit/s dargestellt. Als erster Eingangsparameter variiert die Paketgröße zwischen 0,5 und 4 KByte. Weiterhin wird das Verkehrsangebot von einem KByte/s bis auf sieben KByte/s, entsprechend 56 KBit/s erhöht. Die Untersuchungen sind mit unterschiedlichen Ankunftsstromverteilungen durchgeführt worden, wobei allerdings der mittlere Abstand zwischen Paketankünften konstant gehalten wurde. Man sieht deutlich den Anstieg der Paketverzögerung mit zunehmender Paketlänge aber auch mit steigender Varianz des Ankunftstroms.

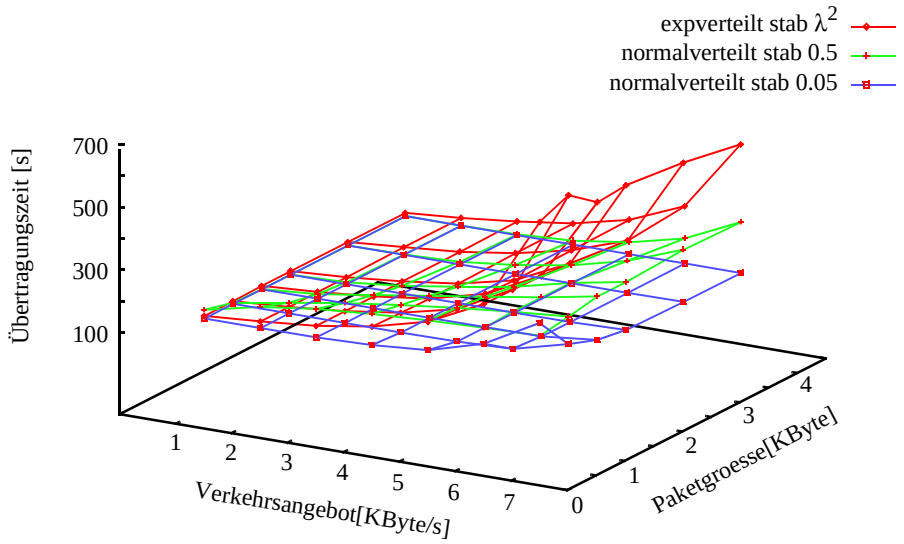


Abbildung 6-6 **Übertragungszeit für 1 MB Daten als Funktion von Paketgröße, Verkehrsangebot und Ankunftsstromverteilung**

Da die Verzögerungszeiten pro Paket wegen der unterschiedlichen Paketgrößen nicht direkt vergleichbar sind, sind die resultierenden Übertragungszeiten für ein MByte Daten hochgerechnet und visualisiert worden, vgl. Abbildung 6-6. Der Einfluß der gewählten Paketgröße wächst sehr deutlich mit zunehmender Last und ist vor allem sehr stark von der Streuung der Zwischenankunftabstände abhängig.

Während bei sehr gleichmäßigen Lastmustern die Übertragungszeiten im Niederlastbereich von der gewählten Paketgröße nahezu unabhängig sind, sollten bei stärker schwankenden Ankunftströmen die Pakete vergrößert werden, vgl. Abbildung 6-6 bzw. im Detail Abbildung 6-8. Dadurch reduziert sich die Anzahl der zu versendeten Pakete und die Übertragungszeit sinkt von 190 s auf 128 s pro MByte.

Im Hochlastbereich stellen kurze Pakete vor allem für sehr gleichmäßige und stark schwankende Ströme ($VK \to 0$ oder $VK \gg 1$) die schlechtere Wahl dar, während für eine mittlere Schwankungsgröße die Übertragungszeit mit der Paketgröße leicht ansteigt, vgl. Abbildung 6-7.

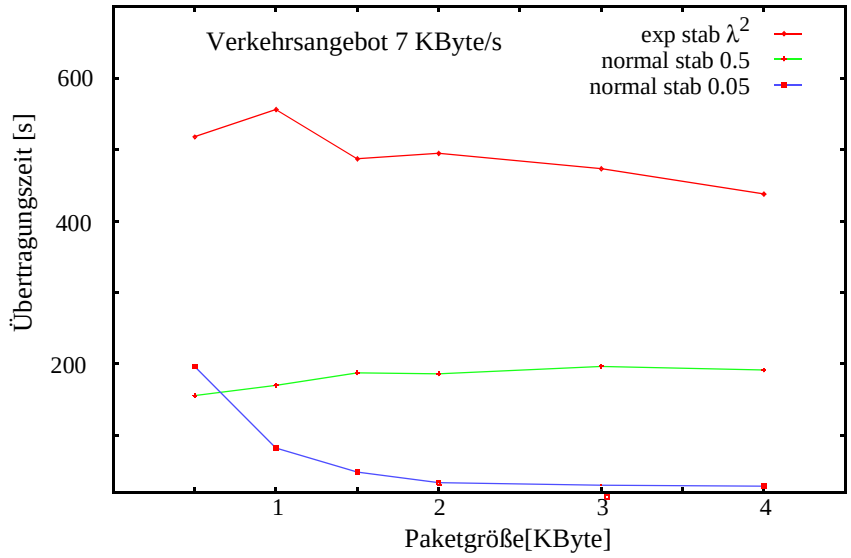


Abbildung 6-7 **Übertragungszeit für 1 MB bei Hochlast**

Weiterhin läßt sich der starke Einfluß der Streuung des Ankunftstroms auf die Verzögerung des Kanals ablesen. So steigt die Verzögerung bei gleichem Mittelwert von sieben KByte/s von etwa 180 Sekunden auf fast 600 Sekunden, wenn man die Varianz des Verkehrsangebotes erhöht.

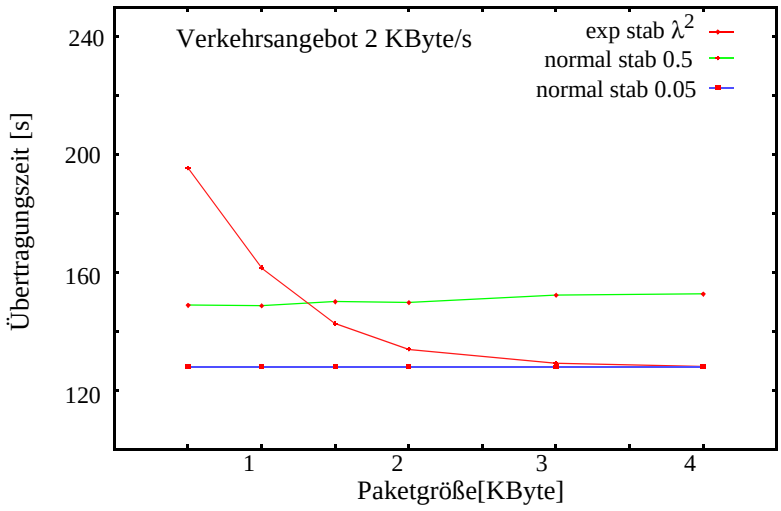


Abbildung 6-8 **Übertragungszeit für 1 MB bei Niedriglast**

Zusammenfassend läßt sich feststellen, daß gleichmäßiger Verkehr und größere Paketlängen die Verweilzeiten in Übertragungsmedien reduzieren. Diese Aussagen sind bereits in der Entwurfsphase auf Basis einer QSDL-Beschreibung erzielbar und ermöglichen so die gezielte Auswahl von Übertragungssystemen beim Entwurf verteilter Systeme.

Die mit Hilfe des Werkzeugs QUEST ermittelten Leistungsmaße sind mit Ergebnissen verglichen worden, die durch Simulationen mit Hilfe des Werkzeugs COMNET III [27] erzielt wurden. COMNET III ist ein voll ausgebautes Simulationswerkzeug zur Bewertung großer Kommunikationsnetze. Dabei sind in das Werkzeug Modellbausteine integriert, die die gängigen Netzwerk-

protokolle und aktive Komponenten beinhalten. Die konkreten Parameter der Bausteine basieren auf Leistungsmessungen, z.B. der Hersteller aktiver Komponenten. So konnte das Simplex-Szenario sehr einfach als Simulationsmodell erstellt und die Ergebnisse der QSDL-Studie auf diese Weise verifiziert werden.

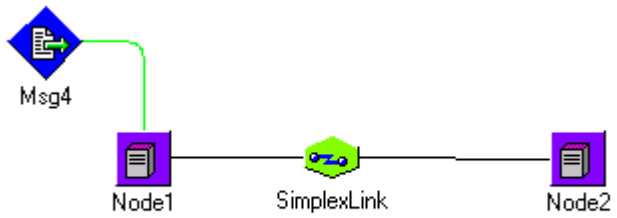


Abbildung 6-9 **COMNET III-Szenario Simplex-Link**

Das Szenario besteht aus zwei Knoten (*Node1* und *Node2*), die durch einen Simplex-Link verbunden sind, vgl. Abbildung 6-9. Die Nachrichtenquelle *Msg4* erzeugt Nachrichten, die vom Knoten *Node1* über den Link *SimplexLink* zum Knoten *Node2* gesendet werden. Die Zwischenankunftsabstände und die Nachrichtengröße sind entsprechend der Lastszenarien aus Abbildung 6-5 variiert worden. Die verwendeten Parameter des Simplexkanals entsprechen denen des QSDL-Modells und sind in Abbildung 6-10 dargestellt.

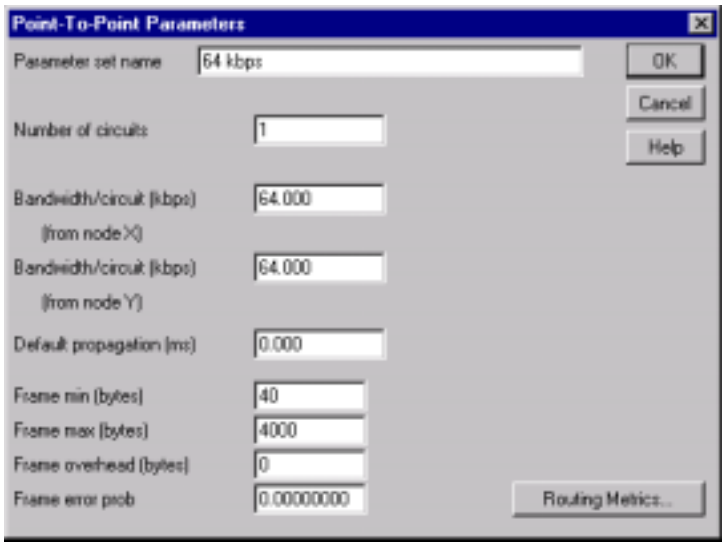


Abbildung 6-10 **Parameter des Modellbausteins 64-kbps-Link**

Der Vergleich der Simulationsergebnisse zeigt, daß die mit QUEST erzielten Ergebnisse mit denen durch COMNET III ermittelten Werten im wesentlichen übereinstimmen, siehe Abbildung 6-11. Lediglich im Hochlastbereich ergeben sich Abweichungen zwischen beiden Werkzeugen, die aber deutlich unter 2% liegen. Damit ist gezeigt, daß die mit Hilfe des QSDL-Ansatzes ermittelten Ergebnisse mit denen durch eine “klassische” Performance-Simulation erzielbaren Ergebnissen übereinstimmen. Als entscheidender Vorteil des QSDL-Ansatzes ergibt

sich, daß die dem klassischen Performance-Modell zugrundeliegende Abstraktion durch das Laufzeitsystem von QUEST geleistet wird.

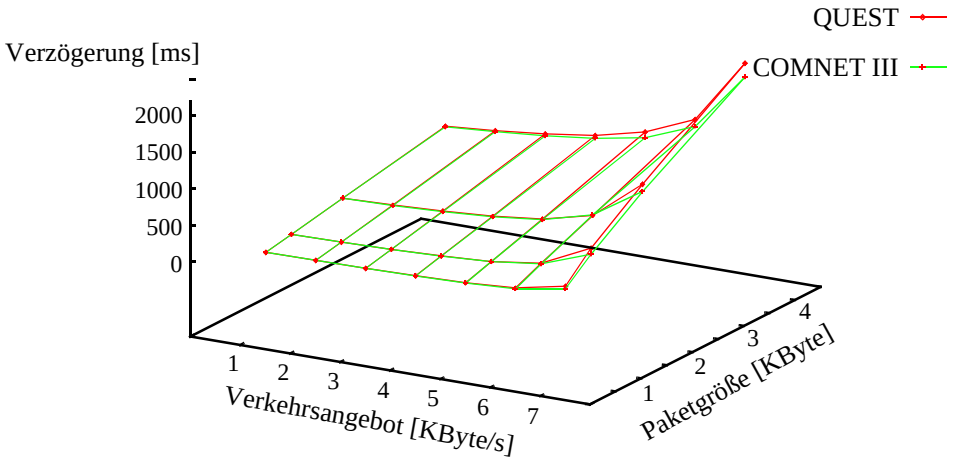


Abbildung 6-11 **Ergebnisvergleich QUEST vs. COMNET III**

6.1.2 Paketvermittelter Duplex-Kanal

Ein Duplexkanal besteht aus zwei unabhängigen Simplex-Kanälen. Daraus ergibt sich, daß im entsprechenden Block für jede Richtung ein Prozeß *Inputdev* und ein Prozeß *Outputdev*, sowie für jede Richtung eine Maschine spezifiziert werden muß. Die Parameter sind für jeden Kanal separat anzugeben, und die Kanäle beeinflussen sich nicht gegenseitig. Alle Aussagen, die für die Simplex-Kanäle gelten, sind entsprechend auf Duplex-Kanäle übertragbar.

6.1.3 Leitungsvermittelter Kanal

Im Gegensatz zur Paketvermittlung wird bei Leitungsvermittlung zwischen den beteiligten Partnern eine physikalische Verbindung geschaltet, die diesen beiden für die Dauer der Verbindung exklusiv zur Verfügung steht. Diese Technik wird in den weltweit vorhandenen Telefonnetzen verwendet. Der Verbindungsaufbau wird durch Signalisierungsprotokolle wie z.B. *common channel signalling system 7* (CCSS7) geregelt. In sog. Vermittlungsknoten werden Eingangsleitungen mit einer geeigneten Ausgangsleitung entweder per Raummultiplex oder per Zeitmultiplex verbunden. Falls keine freien Kapazitäten vorhanden sind, kommt die Verbindung nicht zustande. Da eine exklusiv nutzbare, physikalische Verbindung zwischen beiden Kommunikationspartnern hergestellt wird, entstehen keine Verzögerungen durch Warten auf Netzzugang oder durch Verarbeitung in Zwischenknoten.

Daher entsteht in der Datentransferphase lediglich eine Verzögerung, die der Signalverzögerung entspricht. Die Bandbreite steht ebenfalls exklusiv zur Verfügung, so daß das Leistungsverhalten einer leitungsvermittelten Verbindung durch das Verhältnis von Anwendungslast und Bandbreite berechnet werden kann. Ihre Eignung zur Unterstützung von Realzeit-Diensten ist immer dann gewährleistet, wenn die benötigte Bandbreite durch eine genügend große Anzahl von Leitungen bereitgestellt werden kann. Da leitungsvermittelte Netze im Rahmen dieser Arbeit keine Rolle spielen und auch kein Signalisierungsprotokoll oder Switche bewertet werden sollen, werden leitungsvermittelte Kanäle nicht weiter betrachtet. Im folgenden werden Bussysteme als Netztechnologie für lokale Netze untersucht.

6.2 Bussysteme

Im Gegensatz zu Punkt-zu-Punkt-Verbindungen, die genau zwei Knoten miteinander verbinden, sind an Bussysteme i.a. mehrere Rechner angeschlossen. Sie werden daher auch als sogenannte Rundfunksysteme bezeichnet, da prinzipiell jeder Rechner alle auf dem Medium gesendeten Nachrichten mithört. Bussysteme sind im wesentlichen auf lokale Netze (LAN) und durch Einführung von FDDI auf mittlere Netze (MAN) beschränkt. Neben der Netzwerktopologie unterscheiden sich LANs im wesentlichen durch das Buszugriffsverfahren. Es existieren Zugriffsverfahren mit Kontrolle, sog. *Token-Verfahren* und solche ohne Kontrolle wie *Aloha*, *slotted Aloha* und als Erweiterung Zugriffsverfahren, die vor einem Sendeversuch das Medium auf eine bereits laufende Übertragung abhören. Diese Verfahren sind unter dem Namen *Carrier Sense Multiple Acces* (CSMA) bzw. *Carrier Sense Multiple Acces Collision Detection* (CSMA/CD) bekannt und standardisiert worden [103], [108]. Die IEEE Standards sind als ANSI-Standards und später auch als ISO-OSI-Standards übernommen worden. Sie umfassen die Schichten eins und zwei des ISO-OSI-Modells und gliedern sich dabei in drei Schichten, von denen die beiden obersten *Data Link Layer* Funktionen haben, während die unterste Schicht die physikalischen Ausprägungen festlegt. Die Schicht zwei zerfällt dabei in die Subschichten *Medium Access Control* (MAC) und *Logical Link Control* (LLC). Die LLC-Schicht ist für alle Bussysteme identisch, während die MAC-Schicht und die physikalische Schicht die verfahrensspezifischen Gegebenheiten festlegt, vgl. Abbildung 6-12.

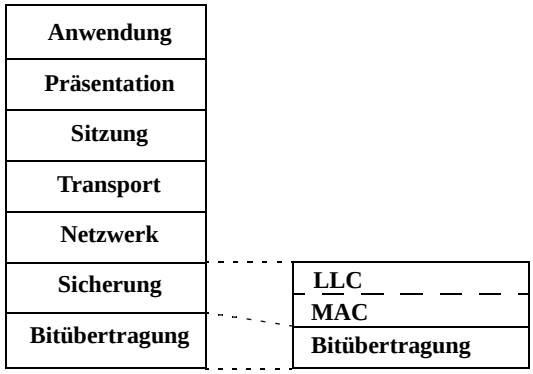
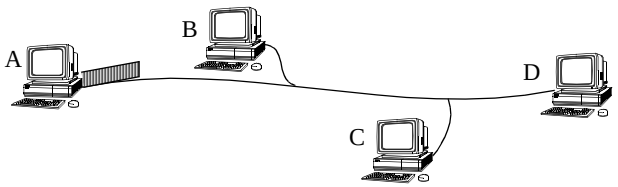


Abbildung 6-12 **LAN-Architektur im Vergleich zum OSI-Modell**

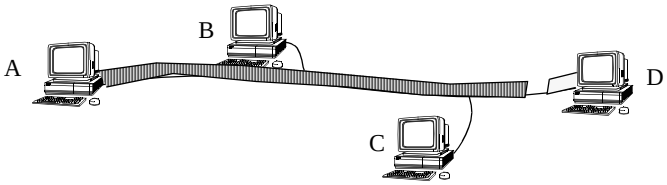
In den nun folgenden Abschnitten werden die QSDL-Spezifikationen der Bussysteme vorgestellt, die in der Anwendungsstudie zusammen mit den Protokollen der höheren Schichten bewertet werden.

6.2.1 CSMA/CD-Systeme

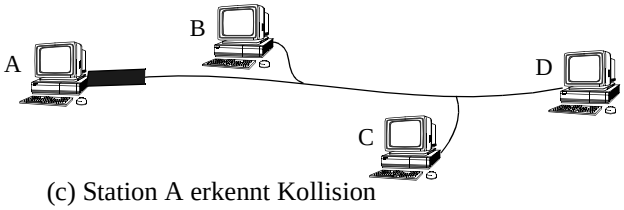
Das am weitesten verbreitete System, welches das Zugriffsverfahren CSMA/CD verwendet, ist das Ethernet [100]. In der Standardversion werden die Rechner an ein Koaxialkabel mit Hilfe von *Transceivern* angeschlossen. Eine sendewillige Station hört das Medium ab und beginnt, falls es frei ist, mit der Sendung des Paketes, vgl. Abbildung 6-13 (a). Alle Stationen hören die Sendung mit und die Stationen, an die es adressiert ist, kopieren es in die Puffer ihrer Netzwerkkarten.



(a) Station A beginnt zu senden; Zeitpunkt t_0



(b) Station D beginnt zu senden; Zeitpunkt $t_1 < t_o + p_t$



(c) Station A erkennt Kollision

Abbildung 6-13 **Paketübertragung auf einem Ethernet**

Falls eine zweite Station mit der Sendung beginnt, kann das nur im Intervall $[t_0, t_0 + p_t]$ passieren, vgl. Abbildung 6-13 (b). Dabei bezeichnet t_0 den Sendebeginn der ersten Station und p_t entspricht der Zeit, die benötigt wird, damit sich das Signal im gesamten Kabel ausbreiten kann.

Falls es zu einer Kollision kommt, dauert es wiederum maximal p_t Zeiteinheiten, bis alle beteiligten Stationen die Kollision erkannt haben und ihren Sendevorgang beenden. Alle an der Kollision beteiligten Stationen warten nun eine zufällige Zeitspanne, bis sie erneut versuchen, eine Übertragung zu starten.

Die Spezifikation von Schicht-1- und Schicht-2-Protokollen ist in SDL nur sehr eingeschränkt möglich, da diese unteren Schichten des OSI-Modells kontinuierliche Phänomene beschreiben und in der Regel in Hardware realisiert sind.

Phänomene wie Kollisionsvermeidung oder zeitliche Signalverzögerung sind mit reinem SDL nicht zu beschreiben. Um jedoch ein homogenes Leistungsbewertungsmodell des Gesamtsystems zu erreichen, können die Hardware-nahen Protokolle mit Hilfe von QSDL-Elementen zumindest insoweit beschrieben werden, daß eine Leistungsbewertung der höheren Protokolle auf der Grundlage einer SDL-Spezifikation erfolgen kann.

Dies wird nun anhand eines Systems erläutert, das aus drei Rechnern besteht, die an ein gemeinsames Übertragungsmedium angeschlossen sind, vgl. Abbildung 6-14.

.

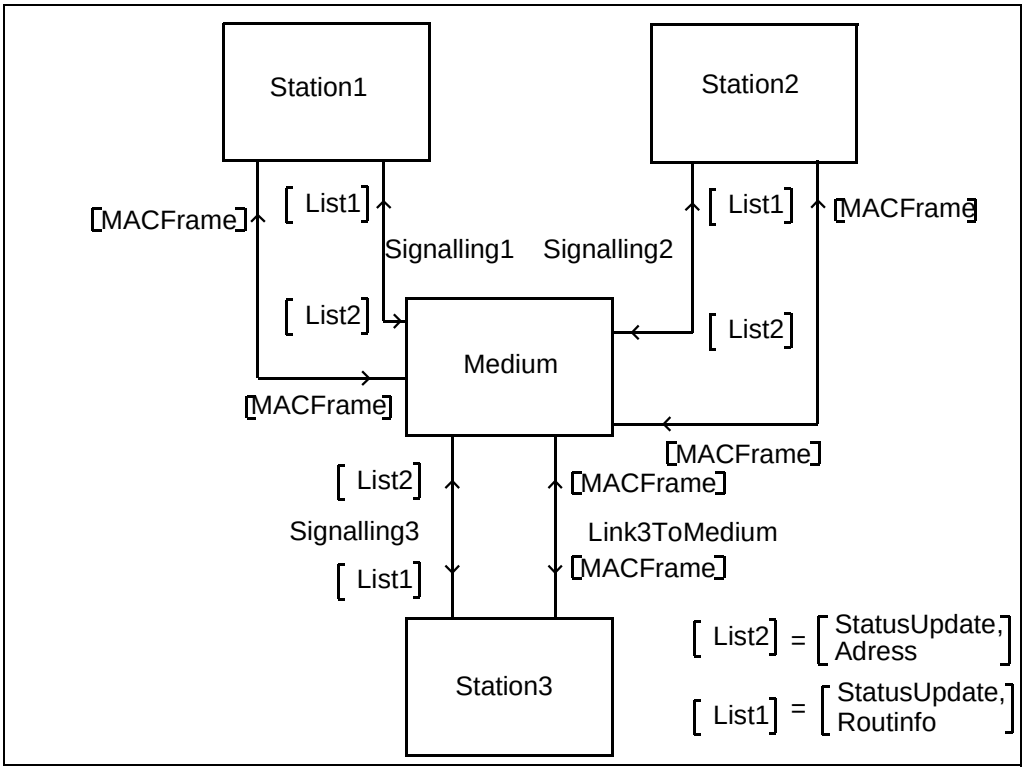


Abbildung 6-14 **QSDL System Ethernet (Block-Level)**

Die Blöcke *Station1* bis *Station3* enthalten die Prozesse *LLC*, *Transmission* und *Receiving*, in denen die Funktionen Link-Kontrolle sowie Senden und Empfangen von *MACFrames* realisiert sind, vgl. Abbildung 6-15. Weiterhin enthält der Block die Maschine *Mach1*, die den Dienst *Transmit* anbietet. Diese Maschine modelliert den Zeitverbrauch, den das Versenden der Rahmen in Anspruch nimmt.

Neben den Nutzdaten, die in *MACFrames* versendet werden, müssen im QSDL-System die Zustände des Mediums als diskrete Signale versendet werden. Dazu sind die Signalisierungskanäle zusätzlich eingefügt. Das Übertragungsmedium ist zu jedem Zeitpunkt in einem der drei Zustände *frei, belegt* oder *kollidiert*. Immer wenn eine Station sendet, schickt sie das Signal *StatusUpdate* an den Prozeß *Manager*, der sich im Block *Medium* befindet. Dieser Prozeß dient lediglich dazu, die logischen und zeitlichen Abläufe auf dem physikalischen Medium zu modellieren.

Zur Adressierung der Signale an die Zielrechner wird die Prozeßidentifikation (PID) des Prozesses *Receiving* als Ersatz für die normalerweise verwendete Ethernet-Adresse genutzt. Die Prozeßidentifikationen werden dynamisch zur Laufzeit vergeben. Deshalb können die Adressen nicht zu Beginn fest vorgegeben werden.

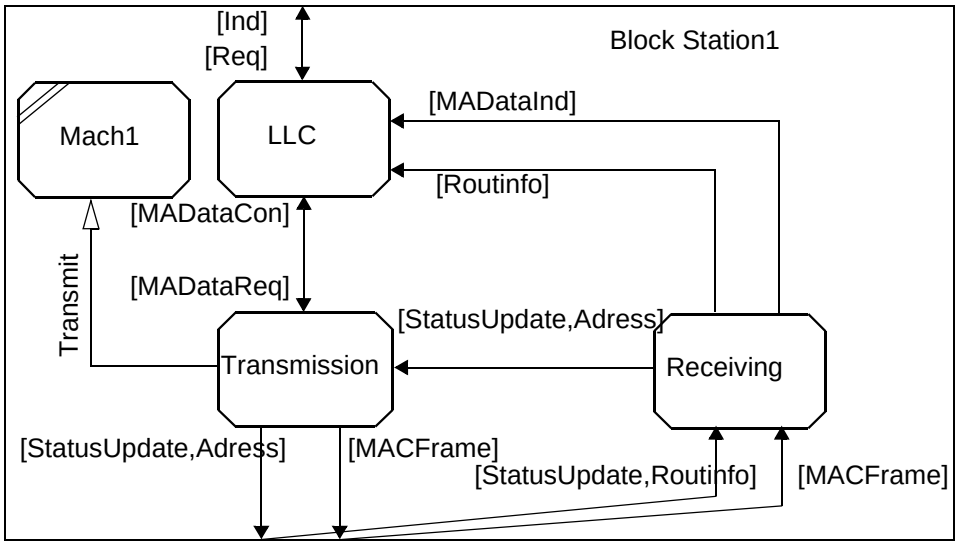


Abbildung 6-15 **Prozesse des Blocks Station**

Alle beteiligten Stationen teilen vor Beginn des eigentlichen Datenverkehrs ihre *PID* dem *Manager*-Prozeß mit, der daraus eine Routing-Tabelle erstellt und diese an alle beteiligten Stationen versendet. Zusätzlich wird neben den eigentlichen Nutzsignalen und den Statusmeldungen die Adresse der Station vor dem eigentlichen Start des CSMA/CD-Protokolls ausgetauscht, vgl. Abbildung 6-15. Im folgenden wird nun erläutert, wie der Datenaustausch zwischen den beteiligten Knoten über den *Manager* abläuft.

```
PROCESS Transmission;

 DCL  id Dim,
      ident PId,
      servclass classpara,
      trials Integer,
      minframe Real,
      len Integer,
      cont Data,
      src PId,
      dest PId,
      bitnr Real,
      stat CableStatus,
      k Integer,
      dist  Randint;
      backofftime Real,
      r Integer,
      upper Integer;
```

```
START;
TASK id := 1,
 bitnr := (bandwidth * pt) +1,
 stat := busy;
TASK minframe := bandwidth * 2.0 * pt;
NEXTSTATE initialize;
STATE initialize;
INPUT Adress(, ident);
OUTPUT Adress(id, ident) VIA MACSignalling1;
NEXTSTATE ready;
```

Abbildung 6-16 **Prozeß Transmission (Variablen, Starttransition)**

Die *MAC-Frames* werden vom Prozeß *Transmission* auf das Übertragungsmedium aufgebracht. In seiner Starttransition legt der Prozeß die minimale Paketgröße fest, die ein Paket haben muß,

damit Kollisionen erkannt werden können. Außerdem wird hier die Zeit berechnet, die benötigt wird, um das Signal von einem Ende des Kabels zum anderen zu senden. Weiterhin wird die Adresse dem Mediumprozeß per Signal bekanntgegeben und erst dann ist der Prozeß *bereit*, siehe Abbildung 6-16.

Die eigentliche Paketübertragung wird durch den Empfang des Signals *MADataReq* initiiert, siehe Abbildung 6-17 oben links. Der MAC-Frame wird zusammengestellt und hat die Mindestlänge *minframe*. Wenn der Rahmen zusammengestellt ist, wird das Medium “abgehört”. Den Zustand des Mediums zeigt die Variable *stat* an. Nur falls diese den Wert *idle* zeigt, darf gesendet werden. Da die Statusmeldungen gegenüber den Nutzdaten priorisiert werden, können Statuswechsel immer sofort den Prozessen zur Kenntnis gelangen.

```
STATE ready;
INPUT MaDataReq(dest, cont, servclass);
TASK trials := 1;
DECISION cont < Fix(minframe);
      (TRUE)  : TASK len := Fix(minframe + 1);
      (FALSE) : TASK len := cont + 192;
ENDDECISION;
NEXTSTATE frameassembled;
STATE frameassembled;
PRIORITY INPUT StatusUpdate(stat);
NEXTSTATE -;
PROVIDED stat = idle;
OUTPUT StatusUpdate(busy) VIA MACSig1;
/*## REQUEST transmit(bitnr); ##*/
 NEXTSTATE starttrans;
```

```
STATE waitbackoff /*## awake backofftime ##*/;
  PRIORITY INPUT StatusUpdate(stat);
    NEXTSTATE -;
  INPUT NONE;
    NEXTSTATE frameassembled;
```

```
STATE starttrans;
  PRIORITY INPUT StatusUpdate(stat);
    DECISION stat;
      (busy) :    /*##  REQUEST transmit(len - bitnr) ##*/;
            OUTPUT MACFrame(dest, src, len, cont);
            OUTPUT MaDataCon(transOK);
            NEXTSTATE ready;
      (collide) :  TASK trials := trials + 1;
                   DECISION trials > maxtrials;
        (FALSE) :  TASK k := CALL minimum(trials,10),
                     upper := CALL pow(2,k),
                     dist := init(dist,0,upper),
                     r := sample(dist),
                     backofftime := FLOAT(r) * 2.0 * propagationtime;
      NEXTSTATE waitbackoff;
         (TRUE) :  OUTPUT MaDataCon(ExcessiveCollisionError);
      NEXTSTATE ready;
     ENDDECISION;
   ENDDECISION;
```

Abbildung 6-17 **Prozeß Transmission (Fortsetzung)**

Falls ein Rahmen auf das als frei erkannte Medium aufgebracht wurde, wird dem Manager dies angezeigt, indem mit Hilfe des Signal *StatusUpdate* das Medium als belegt gekennzeichnet wird, siehe Abbildung 6-17 oben links. Der sendende Prozeß führt einen Request (*Request transmit(bitnr)*) aus. Der Bedienwunsch entspricht der Zeit, die benötigt wird, um allen Stationen die Belegung des Mediums anzuzeigen.

Danach geht er in den Zustand *starttrans*. In der Zwischenzeit empfängt der Prozeß *Manager* im Zustand *ready* das Signal *StatusUpdate* und geht in den Zustand *linebusy*, vgl. Abbildung 6-18 oben rechts.

```
PROCESS Manager;
DCL
   Routingtabel Tabeltype,
   count Integer,
   Num Dim,
   id PId,
   medstat CableStatus,
   dest PId,
   waitbusy Real,
   src PId,
   cont Data,
   len Integer,
   collisiondetected Boolean;

START;
   TASK count := 0,
     collisiondetected := FALSE,
     medstat := busy;
   NEXTSTATE wait;

STATE ready;
 PRIORITY INPUT StatusUpdate(medstat);
 TASK waitbusy := propagationtime;
   NEXTSTATE linebusy;
```

```
STATE linebusy /*## awake waitbusy ##*/;
  PRIORITY INPUT StatusUpdate(medstat);
    TASK collisiondetected := TRUE;
    NEXTSTATE -;
  INPUT NONE;
  DECISION collisiondetected;
     (TRUE)  : OUTPUT StatusUpdate(collide) VIA S1Signalling;
               OUTPUT StatusUpdate(collide) VIA S2Signalling;
               OUTPUT StatusUpdate(collide) VIA S3Signalling;
               NEXTSTATE linecollide;
     (FALSE) : OUTPUT StatusUpdate(busy) VIA S1Signalling;
               OUTPUT StatusUpdate(busy) VIA S2Signalling;
               OUTPUT StatusUpdate(busy) VIA S3Signalling;
               TASK waitbusy := (bandwidth / maxlength);
               NEXTSTATE -;
  ENDDECISION;
    INPUT MACFrame(dest, src, len, cont);
    OUTPUT MACFrame(dest, src, len, cont) TO dest;
    OUTPUT StatusUpdate(idle) VIA S1Signalling;
    OUTPUT StatusUpdate(idle) VIA S2Signalling;
    OUTPUT StatusUpdate(idle) VIA S3Signalling;
    NEXTSTATE ready;
```

Abbildung 6-18 **Prozeß Manager (Ausschnitt)**

Dieser Zustand ist zeitbehaftet. Nach Ablauf der Signalausbreitungszeit wird der Prozeß durch Ablauf der awake-Zeit wieder aktiviert. Falls in der Zwischenzeit eine weitere Station den Beginn einer Sendung angezeigt hat, so wird die Variable *collisiondetected* auf den Wert *true*

gesetzt. Wenn zum Aufweckzeitpunkt diese Variable den Wert *true* besitzt, wird allen Stationen der Zustand *kollidiert* mitgeteilt, andernfalls der Zustand *belegt*.

Nach Ablauf der Bedienzeit des Prozesses *Transmission* findet dieser das Medium im Zustand *belegt* oder *kollidiert*. Falls es *belegt* ist, ist seine Sendung nicht gestört worden und er belegt das Medium für die Zeit, die zur Übertragung benötigt wird (*Request(transmit(len-bitnr))*). Der Bedienwunsch hat diesmal die Größe $G = \frac{Paketgroesse}{Bandbreite} - Signalausbreitungszeit$ und entspricht der Restbedienzeit des ursprünglich zu sendenden Paketes. Nach Ablauf der Bedienzeit wird das Paket an den Manager gesendet, der daran erkennt daß der Rahmen vollständig gesendet ist. Er leitet das Paket entsprechend der Zieladresse an den Empfangsprozeß weiter und kennzeichnet das Medium wieder als frei.

Sollte der Prozeß *Transmission* nach Beendigung des ersten Requests das Medium im Zustand *kollidiert* finden, so ermittelt er gemäß des *exponential Backoff* Algorithmus die Wartezeit bis zum nächsten Sendeversuch oder er bricht diesen Sendeversuch endgültig ab, wenn die maximale Anzahl der Sendeversuche erreicht ist. Das meldet er der LLC-Instanz mit *MAData-Con(Excessive CollisionError)*.

Auf diese Art und Weise kann der logische und zeitliche Ablauf des CSMA/CD Protokolls mit Hilfe von QSDL modelliert und in seiner Leistungscharakteristik bewertet werden.

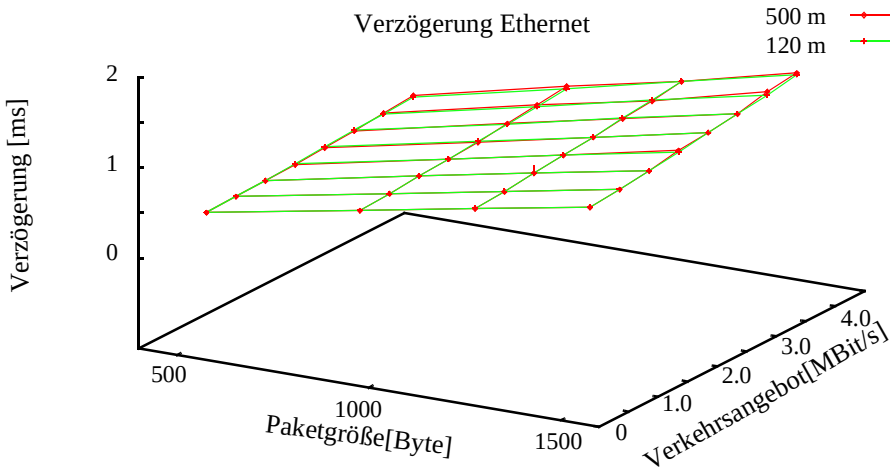


Abbildung 6-19 **Verzögerung eines Paketes im Ethernet**

In der vorliegenden Arbeit ist exemplarisch die Verzögerung eines Paketes bei drei aktiven Stationen und einem Verkehrsangebot von bis zu vier MBit/s untersucht worden, vgl. Abbildung 6-19. Zusätzlich ist die Verzögerung in Abhängigkeit von der gewählten Paketgröße für die Segmentlängen 120 m und 500 m untersucht worden. Dabei konnte beobachtet werden, daß die Verzögerung bis zu einem Angebot von etwa vier MBit/s moderat ansteigt.

Für höhere Verkehrsangebote konnte kein stabiler Betrieb des Ethernets mehr garantiert werden, da durch häufige Kollisionen und dadurch bedingte Paketwiederholungen der effektive Durchsatz unter das Verkehrsangebot sank und somit mehr Pakete zur Übertragung anstanden, als das System durchsetzen kann.[a]

a. Die Eingangspuffer aller Prozesse im System wurden nicht beschränkt.

Die Paketverzögerung steigt sehr stark mit der Paketgröße an. Allerdings ist dieser Wert irreführend, da natürlich die Bedienzeit mit zunehmender Größe ansteigt. Um ein aussagefähiges Bild zu bekommen, ist die benötigte Dauer für eine Übertragung von 1 MByte Daten unter Beibehaltung der bereits oben verwendeten Paketgrößen und Verkehrsangebote ermittelt worden.

Dabei zeigt sich, daß die Verwendung von wenigen großen Paketen deutliche Vorteile gegenüber der Variante mit vielen kleinen Paketen besitzt, da durch sie weniger Kollisionen stattfinden und so der effektive Durchsatz steigt, was zu einer geringeren Gesamtverzögerung führt, vgl. Abbildung 6-20. Die ermittelten Leistungswerte sind im wesentlichen unabhängig von der verwendeten Segmentlänge (in Abbildung 6-20 120 m bzw. 500 m).

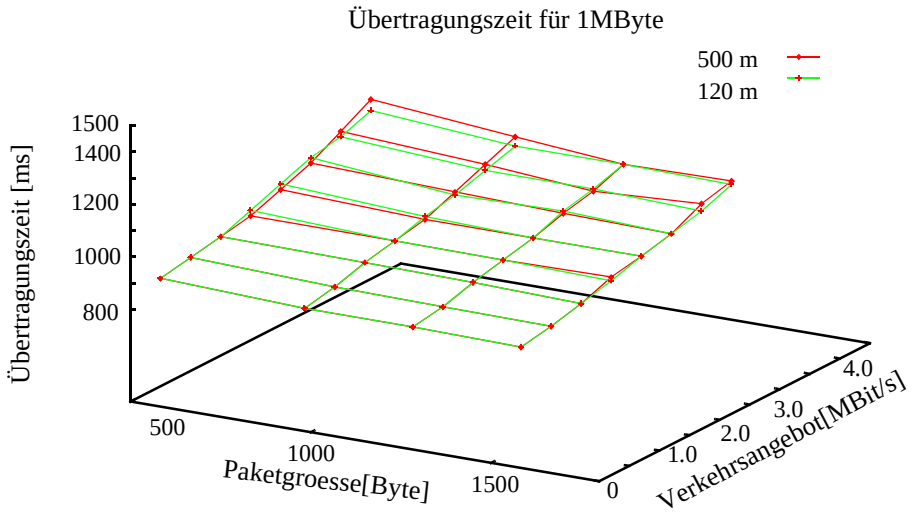


Abbildung 6-20 **Übertragungszeit für 1 MB Daten**

Wie schon in Abschnitt 6.1.1 ist das QSDL-Modell mit Hilfe eines COMNET III-Simulationsmodells verifiziert worden.

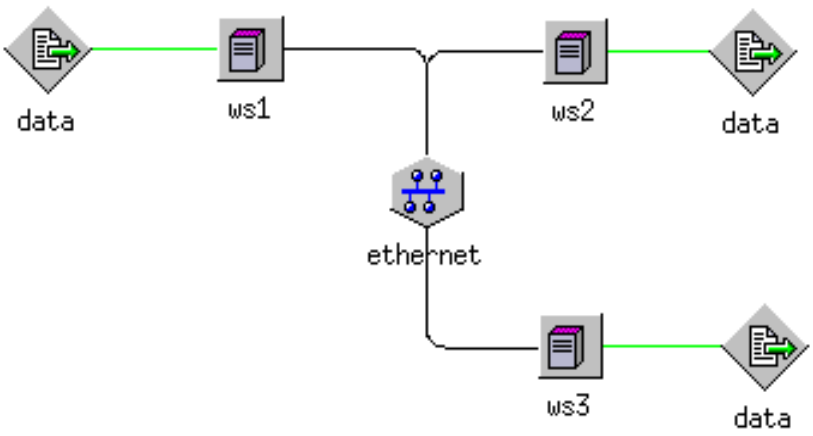


Abbildung 6-21 **COMNET III-Modell**

Das COMNET III-Modell besteht aus drei Rechnern, die an ein Ethernet angeschlossen sind, vgl. Abbildung 6-21. Auf jedem Rechner ist eine Lastquelle definiert, die mit exponentiell verteilten Zwischenankunftsabständen Nachrichten einer festen Größe erzeugt. Die Empfängerstation wird gleichverteilt aus den jeweils anderen beiden Stationen ermittelt. Durch Variation der Paketgröße und der mittleren Zwischenankunftabstände sind die Experimente analog zu denen im QUEST-Modell durchgeführt worden.

Der Netzknoten *ethernet* ist aus der mitgelieferten Modellbibliothek von COMNET III entnommen worden. Die vorgegebenen Modellparameter sind so angepaßt worden, daß sie den Werten des QSDL-Modells entsprechen, siehe Abbildung 6-22.

Ethernet-Parameter

LinkTyp:	CSMA/CD
Parameter:	802.3 Ethernet 10 Base 5
Bandbreite:	10000 k Bit/s
Verzögerung:	0.0025 ms
Frame (max):	1526 Byte
Frame (OH):	26 Byte
Kollisionsfenster:	0.01 ms
Jam Intervall:	0.0032 ms
Interframe Gap:	0.0046 ms
Wiederholung:	IEEE exp. BackOff
Slot-Zeit:	0.0512 ms
Wiederholungsanzahl:	3

Last-Parameter

Zwischenankunftzeit:	exp.-verteilt
Datenmenge:	fest
Ziel:	gleichverteilt

Abbildung 6-22 **Ethernet- und Lastparameter**

Die Werte für Bandbreite, Verzögerung, maximale Frame-Größe und Frame-Overhead sind wie im QSDL-Modell gewählt worden. Die Werte für Kollisionsfenster, *Jam*-Intervall, *Slot*-Zeit und *Interframe-Gap* sind auf den Default-Werten belassen worden, da diese im QSDL-Modell nicht berücksichtigt worden sind. Die maximale Wiederholungszahl und die Verwendung des IEEE exponential Backoff-Algorithmus sind in beiden Modellen identisch.

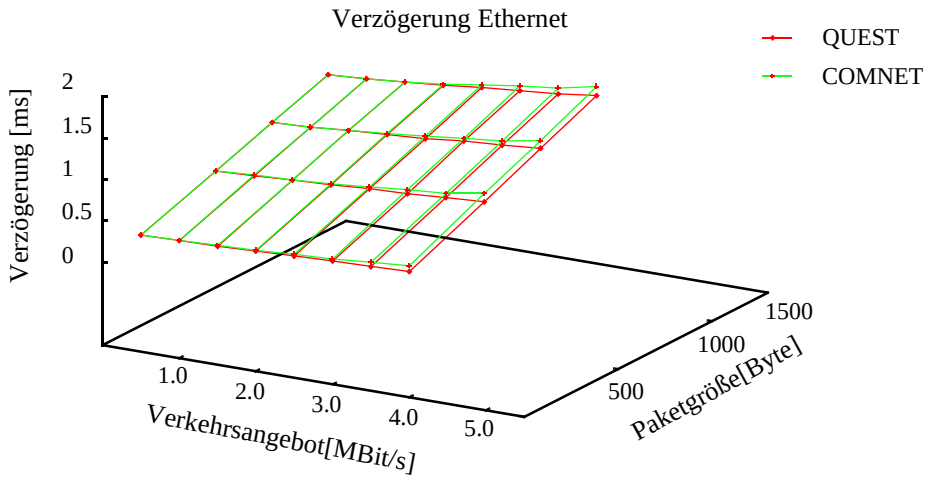


Abbildung 6-23 **Ergebnisse QUEST vs. Comnet III**

Ein Vergleich der Ergebnisse zeigt, daß das QSDL-Modell und das COMNET III-Modell nahezu identische Ergebnisse liefern. Das bedeutet, daß die Güte der mit der QUEST-Methode erzielten Performance-Ergebnisse absolut vergleichbar mit der Qualität der Ergebnisse von klassischer Performance Simulation ist. Darüberhinaus verdeutlicht dies die Eignung der entwickelten Methodik zur entwurfsbegleitenden Bewertung von Kommunikationssystemen.

Mit der zunehmenden Verbreitung von Switching-Komponenten können sich die Einsatzgebiete und die Topologien von Ethernets entscheidend ändern. Verbindet man mit einem Port eines Switch genau einen Rechner über 10 MBit/s, so steht diese Bandbreite exklusiv für diesen Rechner zur Verfügung [46]. Dadurch reduziert sich die Anzahl der Kollisionen und die zur Verfügung stehende Bandbreite kann ausgeschöpft werden. So kann das Verkehrsangebot eines Segmentes bis auf 9 MBit/s anwachsen, ohne daß die hohe Zahl der Kollisionen dazu führt, daß die Übertragungsversuche abgebrochen werden und dies der überliegenden Schicht durch *Excessive Collision Error* angezeigt wird, siehe Abbildung 6-24. Die Verzögerungszeiten der MAC-Frames liegen bei Niedriglast im selben Bereich wie im nicht geswitchten Fall. Allerdings verringert sich die Übertragungszeit ab einem Angebot von 2.5 MBit/s und ab 3.0 MBit/s sind Vergleiche nicht mehr anzustellen, da eine solche Verkehrsintensität von einem klassischen Ethernet nicht bewältigt wird[b].

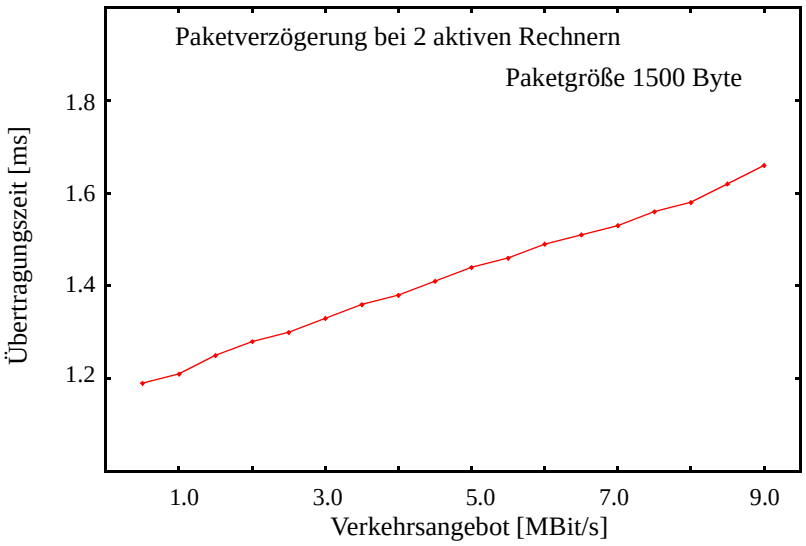


Abbildung 6-24 **Verzögerung bei Verwendung von Switchen**

b. Man beachte, daß die Eingangspuffer in den Prozessen nicht beschränkt wurden.

6.2.2 FDDI-Systeme

FDDI (Fiber Distributed Data Interface) ist ein von der ANSI (American National Standards Institute) entwickelter Standard für lokale Hochgeschwindigkeitsnetze, der auf optischer Übertragungstechnologie für Übertragungsraten von 100 Mbit/s basiert [8]. Analog zu den IEEE Standards für lokale Netze sind Festlegungen für den Medienzugang [3] sowie für die physikalische Schicht [4], [5] getroffen worden. Darüberhinaus ist für FDDI-Stationen ein Schichtübergreifendes Management-Protokoll definiert worden [6].

Der FDDI-Standard spezifiziert eine Topologie, die in einen Ringbereich und einen Baumbereich zerfällt. Alle Stationen bilden einen logischen Ring, der durch zwei gegenläufige Lichtwellenleiter gebildet wird, siehe Abbildung 6-25.

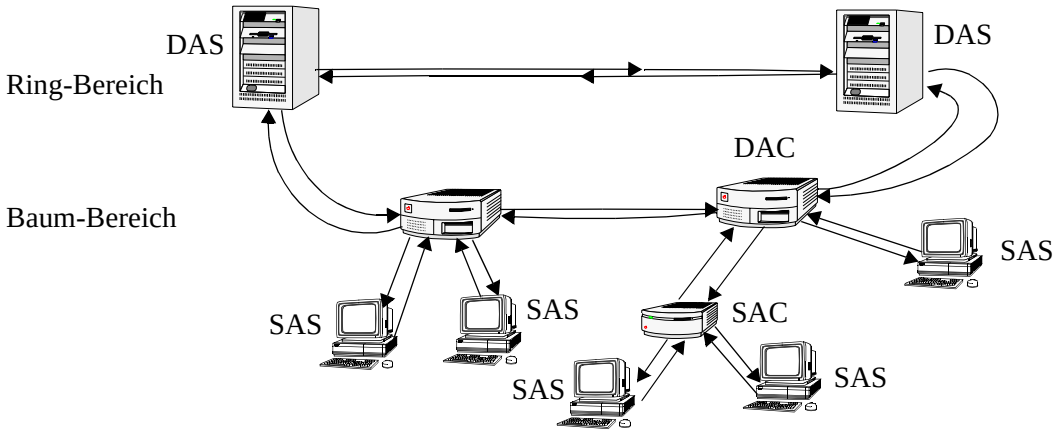


Abbildung 6-25 **FDDI-Topologie**

Im FDDI-Standard sind mehrere Stationstypen definiert, die benötigt werden, um die in Abbildung 6-25 beschriebene Topologie aufzubauen. Als wichtigster Typ, der vor allem bei hochverfügbaren Servern eingesetzt wird, ist die Doppelanschluß-Station (Double Attached Station, DAS) zu nennen, die sowohl an den Primärring, als auch an den Sekundärring angeschlossen ist, während eine Einzelanschluß-Station (Single Attached Station SAS) nur an den Primärring angeschlossen ist und daher bei Ausfall des Primärringes nicht mehr erreichbar ist. Um vom Ringbereich der Topologie in den Baumbereich zu gelangen, sind Doppelanschluß-Konzentratoren (DAC) notwendig. Zur Bildung von Teilbäumen können auch Einzelanschluß-Konzentratoren (SAC) eingesetzt werden, die wie die SAS nur an den Primärring angeschlossen sind. Bei Ausfall des Primärrings sind somit alle Teilbäume, die nur über solche Konzentratoren erreichbar sind, nicht mehr ansprechbar.

Da im Rahmen dieser Arbeit keine Untersuchungen zur Fehlertoleranz bzw. zur Verfügbarkeit von Komponenten gemacht werden, sind alle Stationen als DAS-Komponenten spezifiziert. Eine solche DAS-Komponente besteht aus einer Managementinstanz (SMT), je einer PHY/PMD-Instanz für Primär- und Sekundärring, einem Konfigurationsschalter, einer LLC-Instanz und einer oder zwei MAC-Instanzen, vgl. Abbildung 6-26.

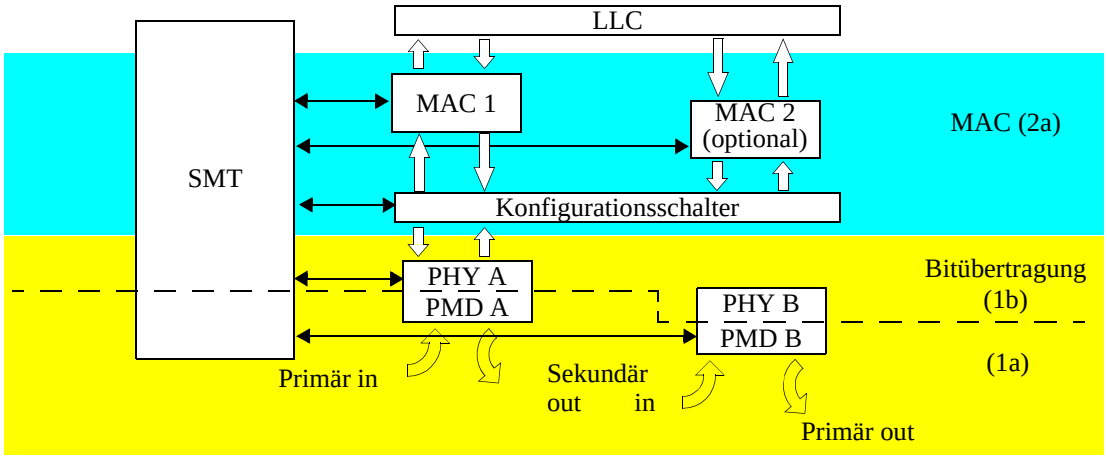


Abbildung 6-26 **Funktionsschaubild einer Doppelanschluß-Station DAS**

Schichtübergreifend ist die SMT-Komponente definiert, zu deren Aufgaben die Initialisierung von FDDI-Stationen, das Einfügen von Stationen in den Ring, Fehlererkennung und -behebung sowie das Sammeln von Statistiken gehört. In diesem Kapitel wird sie nur sehr einfach ausgelegt, da zunächst das Augenmerk auf der Umsetzung des FDDI-Standards in ein (Q)SDL-System liegt. Allerdings wird die SMT-Komponente bei der Unterstützung multimedialer Anwendungen wieder in das Zentrum der Betrachtungen rücken.

In den PHY/PMD-Instanzen werden die physikalischen, optischen und mechanischen Festlegungen getroffen. Zu diesen Festlegungen gehört unter anderem auch die verwendete Bandbreite von 100 MBit/s, die sich in der Bediengeschwindigkeit der QSDL-Maschine wiederfindet, vgl. Abbildung 6-35 auf Seite 128.

Analog zum Vorgehen in Abschnitt 6.2.1 werden die Festlegungen der Schicht-1 nur insoweit berücksichtigt, wie sie für die Leistung des Systems relevant sind. Daher werden zwischen den MAC-Komponenten über die Konfigurationselemente direkt MAC-Frames ausgetauscht. Die Konfigurationselemente werden lediglich dazu benutzt, wahlweise über den Primär- oder den Sekundärring zu übertragen.

Die MAC-Komponente bildet das zentrale Element innerhalb einer FDDI-Station. Zu ihren Aufgaben zählen die Realisierung des Mediumzugriffsverfahrens für synchronen und asynchronen Verkehr, die Adresserkennung, die CRC-Funktionen, das Versenden von MAC-Frames und somit letztlich die Bereitstellung des MAC-Dienstes für die Link-Layer-Schicht. Diese Aufgaben werden anhand des QSDL-Systems im folgenden detailliert erläutert, siehe Abbildung 6-27.

Das Medienzugangsprotokoll innerhalb des FDDI-Standards enthält ein zeitgesteuertes Verfahren, in dem jede Station abwechselnd für eine gewisse Zeit das Senderecht durch eine Token-Zuteilung erhält. In der Ringinitialisierungsphase wird durch den sog. *Claim-Prozeß* der Abstand zwischen zwei Token-Zuteilungen und die Zeitdauer, die eine Station das Token festhalten darf, zwischen den beteiligten Stationen ausgehandelt. Dazu senden alle aktiven Stationen *Claim-Frames*, in denen sie ihren gewünschten Wert *(t-bid*) in das Datenfeld eintragen. Den Wert erhalten die MAC-Komponenten während ihrer Initialisierung durch die Managementkomponente.

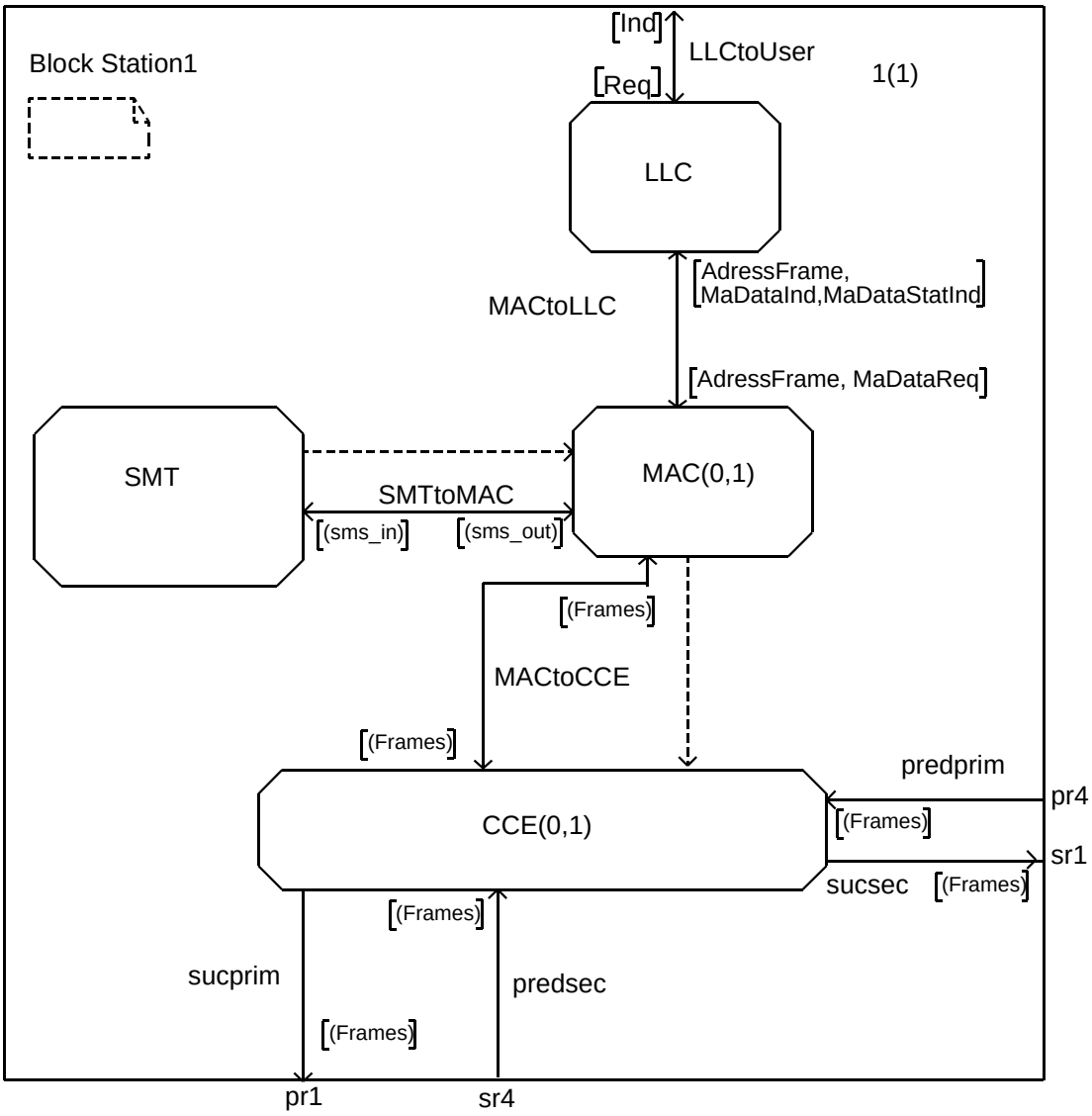


Abbildung 6-27 **FDDI-Station als QSDL-Block**

Jede Station, die einen solchen *Claim-Frame* empfängt, vergleicht den enthaltenen Wert mit dem eigenen Wunsch. Sollte der Wert des eigenen Wunsches kleiner sein als der Eintrag im *Claim-Frame*, sendet diese Station weiterhin Frames mit dem eigenen Wert, andernfalls leitet sie den empfangenen Frame weiter und stellt die Sendung eigener Frames ein.

Am Ende des Claim-Prozesses “gewinnt” die Station, die den kürzesten Abstand zwischen zwei Token-Zuteilungen wünscht. Diese Station generiert ein erstes *nonrestricted*-Token und gibt es auf den Ring. Alle Stationen, die dieses Token empfangen, wechseln in einen sogenannten *operation mode* und initialisieren die Variablen und Timer, die zur Ringüberwachung definiert sind. Der im Claim-Prozeß ausgehandelte Wert wird als Ringparameter *Target Token Rotation Time* (*TTRT)* von allen Stationen gespeichert.

Die in einem FDDI-Ring zur Verfügung stehende Bandbreite kann für sehr unterschiedliche Verkehrsarten genutzt werden, siehe Abbildung 6-28.

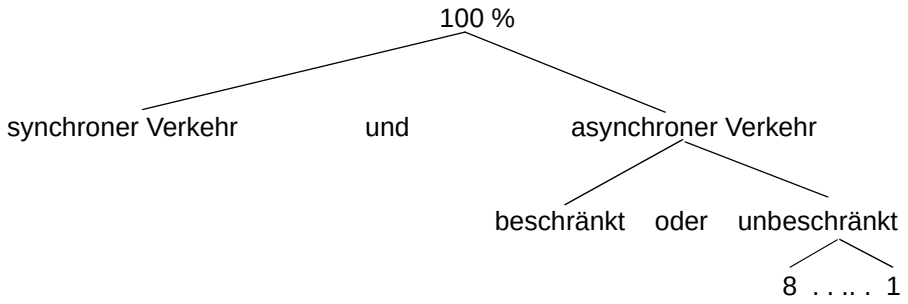


Abbildung 6-28 **Bandbreiten Nutzung in einem FDDI-Ring**

Asynchroner und synchroner Datenverkehr können parallel in einem FDDI-Ring betrieben werden. Die für asynchronen Verkehr zur Verfügung stehende Bandbreite kann entweder auf zwei Stationen beschränkt werden, die diese für eine schnelle Verbindung exklusiv nutzen, oder allen Stationen zur Verfügung stehen. Der unbeschränkte asynchrone Datenverkehr kann in bis zu 8 Prioritätsklassen aufgeteilt werden. Dann verwaltet jede Station für jede Klasse einen Schwellwert, anhand dessen entschieden werden kann, ob bei Zuteilung des Tokens in diesem Zyklus noch asynchroner Verkehr der Klasse i gesendet werden darf.

Die Zeitdauer, die das Token für sog. synchronen Verkehr gehalten werden darf, wird als synchrone Reservierung (*Synchronous Allocation, SA*) bezeichnet. Für die Festlegung der SA-Werte ist das *Station Management Protocol* (SMT-Protokoll) zuständig. Dabei muß die Einhaltung der folgenden Ungleichung gewährleistet sein:

$$\sum_{i=1}^{n} SA_i + Dmax + Fmax + TokenTime \leq TTRT \qquad \text{(Gl. 1)}$$

Dmax entspricht der Signallaufzeit für einen Ringumlauf, *Fmax* der Übertragungszeit für einen Frame maximaler Länge und *TokenTime* der Übertragungszeit für das Token selbst.

Eine Station, die einen Sendewunsch hat, wartet auf den Empfang eines Tokens. Empfängt sie eines, überprüft sie zuerst, wie lange seit dem Empfang des letzten Token vergangen ist. Liegt dies tatsächliche *Token Rotation Time* TRT unterhalb des Wertes für TTRT, so darf diese Station für die Zeit *Token Holding Time* (THT = TTRT - TRT) asynchrone Daten senden. Zunächst werden aber synchrone Daten-Frames versendet. Die Sendung synchroner Frames wird eingestellt, wenn keine synchronen Daten mehr vorhandent sind, oder die durch SA definierte Zeitschranke erreicht ist..

Erst danach dürfen für die Dauer THT asynchrone Frames versendet werden. Falls TRT > TTRT bei Empfang des Token gelten sollte, darf diese Station nur synchrone Frames versenden. Falls alle Daten gesendet wurden oder die entsprechenden Zeitschranken erreicht sind, wird das Token an die Nachfolgerstation im Primärring weitergereicht. Jede Station empfängt auf dem Ring befindliche Frames, kopiert solche, die für sie bestimmt sind und sendet alle Frames an die Nachfolgerstation. Erreicht ein Frame nach einem Ringumlauf wieder den Absender, so wird er von ihm aus dem Ring entfernt.

Die korrekte Funktion des FDDI-Ringes wird durch eine Reihe von Timern und Variablen überwacht, siehe Abbildung 6-29. Jede Station besitzt einen sog. Aktivitäts-Timer TVX, der immer dann zurückgesetzt wird, wenn ein gültiger Frame die Station erreicht. Sollte dieser Timer ablaufen, wird der Claim-Prozeß initiiert. Dabei wird der TRT auf seinen maximalen Wert gesetzt. Sollte dieser dann ablaufen, muß der Ring unterbrochen sein und es setzt der *Beacon*-Prozeß ein, in dem die defekte Stelle entdeckt und durch Umkonfiguration die defekte Stelle überbrückt wird. Details hierzu entnehme man der entsprechenden Literatur, z.B. [3], [8] oder [104]. Der Claim-Prozeß wird ebenfalls gestartet, falls das Token zweimal verspätet ist, da dann offensichtlich der Wert für TTRT neu verhandelt werden muß.

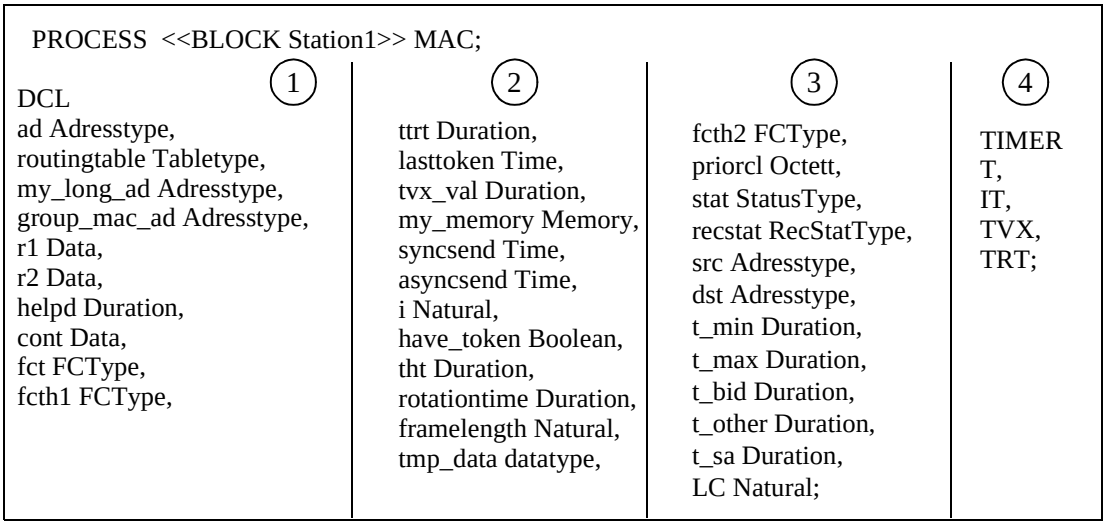


Abbildung 6-29 **Prozeß MAC Variablen und Timer**

Bevor nun die QSDL-spezifischen Ergänzungen der SDL-Spezifikation erläutert werden, sollen hier zunächst einige Probleme und deren Lösung aufgezeigt werden, die sich ergeben, wenn die Angaben der X3T9-Dokumente ([3],[4],[5] und [6]) in eine SDL-Spezifikation umgesetzt werden sollen.

Laut ANSI-Standard ist eine FDDI-Station optional in der Lage, acht Prioritätsklassen für asynchronen Verkehr zu unterstützen. Da der SDL-Prozeß *MAC* die Frame-Übertragungsanforderung *MADataReq* in der Reihenfolge des Eintreffens in seiner Eingangswarteschlange bearbeitet und er nur dann MAC-Frames versenden kann, wenn er im Besitz des Sende-Tokens ist, müssen die sendebereiten Daten gespeichert werden. Das kann entweder durch Definition einer geeigneten Datenstruktur innerhalb des Prozesses MAC geschehen oder als Signal in der Eingangswarteschlange eines separaten SDL-Prozesses. Dann wäre aber für jede Prioritätsklasse ein SDL-Prozeß nötig, der durch gesonderte Trigger vom MAC-Prozeß aktiviert wird.

Einfacher und auch spezifikationsstechnisch naheliegender ist die Definition eines lokalen Speichers für Anwenderdaten. Dieser besteht aus einem *Array* mit neun Warteschlangen als Inhalt. In diesen Warteschlangen können die acht asynchronen Prioritätsklassen und die synchronen Daten zwischengepuffert werden, bis die Station das Sende-Token erhält, vgl. Abbildung 6-31. Durch Variation der Konstanten *maxbuf* kann die Größe der Puffer begrenzt werden, und so können implementierungsspezifische Angaben integriert werden. Die Werte für *first* und *last*, die auf das aktuelle Ende bzw. den aktuellen Anfang der Warteschlange je Klasse zeigen, werden modulo *maxbuf* erhöht. Somit stellt die Warteschlange eigentlich einen FIFO-Ringpuffer dar. Diese Ring-

puffer werden in der Starttransition des MAC-Prozesses als leer initialisiert, vgl. Abbildung 6-30.

```
START;
  TASK i := 0,
    have_token := false,
    t_sa := 0;
  TASK helpd := 1.0,
    tvx_val := (2.0 * helpd * F_Max) + (helpd * Token_Time) + (helpd * S_Min);
  lab1 :      /* leere Puffer initialisieren */;
  DECISION i = 9;
  (False) :
    TASK my_memory(i)!firstelem := 1,
      my_memory(i)!last := 1,
      my_memory(i)!filled := 0,
      i := i + 1
    JOIN lab1;
  (TRUE) :
    NEXTSTATE waitinit;
  ENDDECISION;
```

Abbildung 6-30 **Prozeß MAC Starttransition**

Ein weiteres Problem ergibt sich bei der Überwachung der Ringfunktion durch Timer. Die Ermittlung der TRT sowie die Überwachung der THT soll laut FDDI-Dokumentation durch Timer erfolgen, die aufwärts laufen. So wird bei Empfang eines Token der Wert des THT auf die aktuelle TRT gesetzt. Sind alle synchronen Frames versendet, wird dieser Timer aktiviert. Wenn er den Wert TTRT erreicht hat, muß das Token weitergegeben werden.

```
SYNTYPE Dim1 = Natural     CONSTANTS 1 : maxbuf    ENDSYNTYPE;
SYNTYPE priorlevel = Natural     CONSTANTS 0 : 8    ENDSYNTYPE;
NEWTYPE Dataframes STRUCT
     dst Adresstype;
     src Adresstype;
     fc_val FCType;
     cont Data;
     stat StatusType;
ENDNEWTYPE;
NEWTYPE contents_type     ARRAY( Dim1, Dataframes)       ENDNEWTYPE;
NEWTYPE FrameQueue STRUCT
    firstelem Dim1;
    last Dim1;
     filled Natural;
    contents contents_type;
 ENDNEWTYPE;
NEWTYPE Memory    ARRAY( Priorlevel, FrameQueue)     ENDNEWTYPE;
```

Abbildung 6-31 **Datenstruktur zur Verwaltung der Anwenderdaten in Prioritätsklassen**

SDL-Timer hingegen laufen rückwärts bis auf den Wert Null zurück. Eine einfache Lösung besteht darin, im SDL-System den Token Holding Timer auf den Wert TTRT - TRT zu setzen und ihn dann bis auf Null zurücklaufen zu lassen. Dann würde zum Ende der Sendezeit das Timer-Signal in die Eingangswarteschlange des Prozesses geschrieben, wo es erst dann konsu-

miert wird, wenn alle vorher eingegangenen Signale, z.B. Anwenderdaten, verarbeitet worden sind. Somit kann wegen der SDL-Timer-Semantik die zeitgenaue Reaktion innerhalb der Zeitschranke nicht garantiert werden. Vielmehr wird durch die Verwendung von SDL-Timern eine **untere** Schranke für die Reaktion festgelegt und keine **exakte** Schranke.

Die Verwendung des Priority-Inputs von SDL schafft an dieser Stelle zwar eine Bevorzugung der Timer-Signale gegenüber allen anderen Signalen, allerdings kann das Problem wieder auftreten, wenn mehrere Timer-Signale in der Eingangswarteschlange stehen. Hier zeigt sich abermals, daß in SDL ein Interrupt-Mechanismus notwendig ist, um eine **zeitgenaue** Reaktion von Prozessen modellieren zu können.

```
STATE send_sync;
  PROVIDED have_token;
    TASK dst := my_memory(0)!contents(my_memory(0)!firstelem)!dst,
      src := my_memory(0)!contents(my_memory(0)!firstelem)!src,
      fct := my_memory(0)!contents(my_memory(0)!firstelem)!fc_val,
      cont := my_memory(0)!contents(my_memory(0)!firstelem)!cont,
      stat := my_memory(0)!contents(my_memory(0)!firstelem)!stat;
    TASK tmp_data := first(cont),
        framelength := (tmp_data!content + dataheaderlength);
    /*## Request transmit(float(framelength)); ##*/
    OUTPUT DataFrame(dst, src, fct, cont, stat) VIA MACtoCCE;
    TASK my_memory(0)!filled := my_memory(0)!filled - 1;
    DECISION NOW < syncsend;
           (true) :    DECISION my_memory(0)!filled = 0;
                          (false) :   TASK my_memory(0)!firstelem :=
                                             (my_memory(0)!firstelem + 1) MOD     (maxbuf);
                                      NEXTSTATE -;
                          (true) :
                       ENDDECISION;
           (false) :
      ENDDECISION;
    NEXTSTATE send_async;
     SAVE MaDataReq;
     ENDSTATE;
```

Abbildung 6-32 **Prozeß MAC Zustand send_sync**

Im vorliegenden SDL-System ist das Problem dadurch gelöst worden, daß kein Token Holding Timer verwendet wird, sondern analog zu den Angaben des X3T9-Dokumentes bei Empfang des Tokens die Station in den Zustand *send_sync* wechselt. Vorher wird der Zeitpunkt ermittelt, zu dem dieser Zustand spätestens verlassen werden muß. Dieser Wert wird in der Variablen *syncsend* gespeichert. Solange die zeitliche Bedingung *NOW* < *syncsend* gilt, werden vorhandene synchrone Daten gesendet, vgl. Abbildung 6-32.

Falls der Puffer für synchrone Daten leer ist bevor der Zeitpunkt *syncsend* erreicht ist, wechselt der Prozeß vorzeitig in den Zustand *send_async*, in dem er dann zuerst die prüft, ob er in dieser Runde asynchrone Daten senden darf, siehe Abbildung 6-33.

Falls diese Entscheidung positiv ausfällt, beginnt er die Daten der höchsten Priorität zu versenden, sofern solche vorhanden sind. Die Station sendet solange Daten der Klasse i, bis der zugehörige Datenspeicher leer ist, oder die Zeitschranke für die Klasse erreicht ist. Sind alle Puffer leer oder die Sendezeit abgelaufen, wird das Token an die Nachfolgestation im Ring gesendet.

```
STATE send_async;
 SAVE MaDataReq;
 PROVIDED have_token;
  TASK i := 1;
  lab3 :   DECISION my_memory(i)!filled > 0;
             (true) :    DECISION rotationtime < ttrt;
                           (true) :    TASK LC := 0,
                                              tht := ttrt - rotationtime,
                                              asyncsend := NOW + tht;
                           (false) :   TASK have_token := false;
                                        TASK framelength := tokenlength;
                                       /*##Request transmit(float(framelength));##*/
                                       OUTPUT TokenFrame(nonrestricted) VIA MACtoCCE;
                                       NEXTSTATE operation_mode;
                                     ENDDECISION;
             (false) :   DECISION i = 8;
                           (true) :TASK have_token := false;
                                    TASK framelength := tokenlength;
                                    /*##Request transmit(float(framelength));##*/
                                    OUTPUT TokenFrame(nonrestricted) VIA MACtoCCE;
                                    NEXTSTATE operation_mode;
                           (false) :    TASK i := i + 1;
                                        JOIN lab3;
                                  ENDDECISION;
            ENDDECISION;
   lab4 :   TASK dst := my_memory(i)!contents(my_memory(i)!firstelem)!dst,
                      src := my_memory(i)!contents(my_memory(i)!firstelem)!src,
                       fct := my_memory(i)!contents(my_memory(i)!firstelem)!fc_val,
                       cont := my_memory(i)!contents(my_memory(i)!firstelem)!cont,
                      stat := my_memory(i)!contents(my_memory(i)!firstelem)!stat;
            TASK tmp_data := first(cont),
                      framelength := tmp_data!content + dataheaderlength;
            /*##Request transmit(float(framelength));##*/
             OUTPUT DataFrame(dst, src, fct, cont, stat) VIA MACtoCCE;
              TASK my_memory(i)!filled := my_memory(i)!filled - 1,
                       my_memory(i)!firstelem := (my_memory(i)!firstelem + 1) MOD (maxbuf);
            DECISION NOW < asyncsend;
               (true) :  DECISION my_memory(i)!filled = 0;
                           (true) :    TASK my_memory(i)!firstelem :=
                                              (my_memory(i)!firstelem) MOD (maxbuf);
                           (false) :   TASK i := i + 1;
                                       JOIN lab4;
                         ENDDECISION;
               (false) :   TASK have_token := false;
                           TASK framelength := tokenlength;
                           /*##Request transmit(float(framelength));##*/
                          OUTPUT TokenFrame(nonrestricted) VIA MACtoCCE;
                           NEXTSTATE operation_mode;
             ENDDECISION;
ENDSTATE;
```

Abbildung 6-33 **Prozeß MAC Zustand send_async**

Nachdem die Schwierigkeiten bei der Umsetzung der ANSI-Festlegungen in das SDL-Modell aufgezeigt und Lösungen dargestellt wurden, sollen nun die QSDL-spezifischen Ergänzungen zur Erstellung des Performance-Modells erläutert werden.

Ergänzend zu den Informationen, die das SDL-Modell des FDDI-Systems enthält, sind vier Arten performance-relevanter Informationen eingefügt worden. Diese sind im einzelnen:

1. Maschinen und Maschinendienste zur Modellierung der Verzögerung, die durch das Aufbringen des Signals auf das Medium entsteht.
2. Ergänzung der SDL-Output-Aktion um Delay-Informationen, die die physikalische Signallaufzeit nachbilden.
3. Definition von QSDL-Sensoren, die die Maschinenauslastung bzw. die Ende zu Ende Verzögerung von Anwenderdaten erfassen.
4. Definition von Lastprozessen, die an den oberen Dienstschnittstellen (UI1-UI3) den durch das FDDI-System erbrachten Dienst in Anspruch nimmt, vgl. Abbildung 6-34.

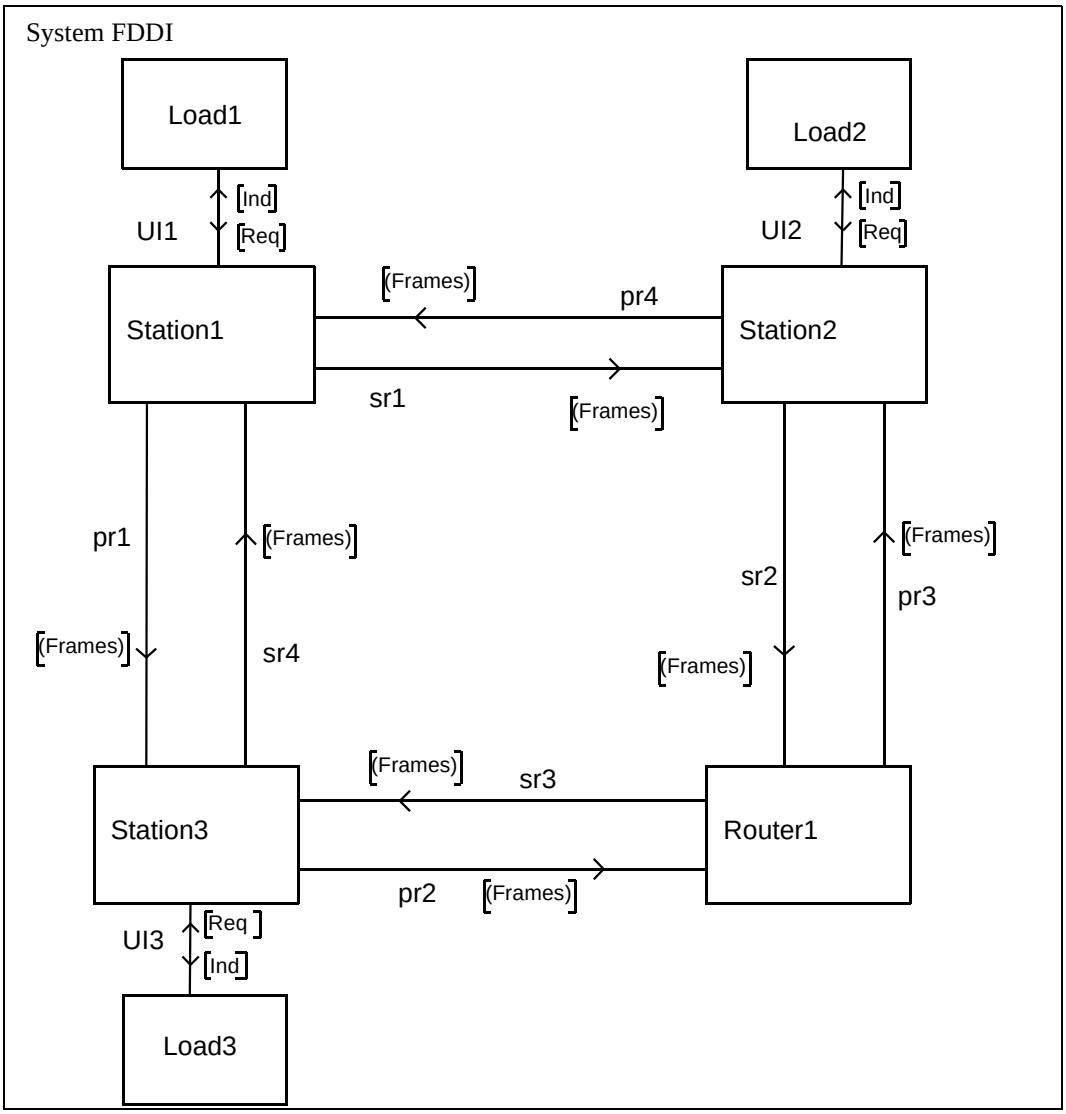


Abbildung 6-34 **QSDL-System FDDI-Ring**

Bei der Übertragung von MAC-Frames werden diese beim Übergang von der MAC-Komponente auf das Medium einer Verzögerung ausgesetzt, die sich als Quotient von Frame-Länge (Bedienwunsch) und der Bandbreite des Mediums (Maschinengeschwindigkeit) ergibt.

Daher wird eine Maschine benötigt, die diesen Dienst mit der entsprechenden Geschwindigkeit anbietet. Diese Maschine kann entweder zentral definiert und für alle MAC-Komponenten zugänglich sein, oder für jede Komponente lokal definiert werden. Da das Zugriffsprotokoll über die Token-Vergabe die Benutzung des Mediums regelt, können keine Konflikte oder Konkurrenzsituationen auftreten. Daher sind beide Varianten aus Performance-Sicht äquivalent. In der hier vorliegenden Spezifikation sind die Maschinen lokal definiert worden, vgl Abbildung 6-35.

```
/*## Machineservice transmit(Real); ##*/
/*## MACHINE fddi2;
            SERVER  1;
            DISCIPLINE FCFS;
            OFFERS transmit : bandwidth;
      ENDMACHINE fddi2; ##*/
/*## LINK machinepoint FROM MAC TO fddi2
WITH transmit; ##*/
```

Abbildung 6-35 **Maschinendienst, Maschinen und Link Definition FDDI System**

Der Maschinendienst *transmit* wird jedesmal vom Prozeß *MAC* angefordert, wenn er im Zustand *send_sync* oder *send_async* Daten-Frames oder das Token-Frame sendet, vgl. Abbildung 6-32 und Abbildung 6-33.

Daneben wird jeder Frame auf dem Medium gemäß der physikalischen Signallaufzeit verzögert. Der konkrete Wert ergibt sich als Quotient von Mediumlänge und Signalausbreitungsgeschwindigkeit. Diese Verzögerung geschieht auf jedem Teilstück zwischen zwei Knoten. Da im SDL-Modell der Prozeß *CCE* für die Weiterleitung von eingehenden Frames verantwortlich ist, wird an dieser Stelle die entsprechende Verzögerung mit Hilfe des Output-delay-Konstrukts berücksichtigt, vgl. Abbildung 6-36.

```
PROCESS  <<BLOCK Station2>> CCE;
      DCL delaypara Duration;
      START;
      TASK delaypara := distance / signalspeed;
      NEXTSTATE waitrouting;
     STATE primary;
     INPUT TokenFrame(frcheck);
     DECISION SENDER = local;
        (TRUE) :  OUTPUT TokenFrame(frcheck) VIA sucprim  /*## delay delaypara ##*/;
        (FALSE) :OUTPUT TokenFrame(frcheck) VIA MACtoCCE;
     ENDDECISION;
     NEXTSTATE primary;
```

Abbildung 6-36 **Ausschnitt QSDL-Prozeß CCE**

Diese wenigen Ergänzungen sind ausreichend, um das Verhalten eines FDDI-Rings auch unter Performance-Aspekten zu beschreiben. Das FDDI-System kann nun von Dienstnehmern an der

oberen Dienstschnittstelle belastet werden, indem diese die Übertragung von MAC-Frames anfordern. Dazu sind im QSDL-System Lastprozesse definiert worden, die kontrolliert Lastmuster erzeugen. Die erzielten Performance-Ergebnisse, speziell die beobachtete Verzögerung, die durch den FDDI-Ring hervorgerufen wird, ist für eine Reihe unterschiedlicher Betriebsparameter untersucht worden. Auf diese Art und Weise lassen sich günstige und ungünstige Werte für die wichtigsten Betriebsparameter bestimmen, ohne aufwendige und teure Messungen in einem prototypisch implementierten System durchzuführen.

Zu den Betriebsparametern, die die Performance des FDDI-Ringes entscheidend beeinflussen, zählen neben der maximal zulässigen Paketgröße, die ausgehandelte TTRT, sowie die Anzahl sendewilliger Stationen. Weiterhin hat die dem System übergebene Arbeitslast einen großen Einfluß auf die beobachtbare Performance. In einer Experimentserie ist für TTRT-Werte von 25 Millisekunden und 100 Millisekunden die Paketverzögerung bei Verkehrsangeboten bis zu 90 MBit/s untersucht worden. Dabei sind die maximal zulässigen Paketgrößen von 1000 Byte bis 4000 Byte variiert worden. Die Anzahl der sendebereiten Stationen wurde konstant gehalten.

Die Simulationen mit QUEST haben ergeben, daß bei einer geringen Anzahl von Stationen die Wahl der TTRT keinen nennenswerten Einfluß auf die Paketverzögerung hat. Der Einfluß der gewählten Paketgröße und insbesondere das Verkehrsangebot beeinflussen die Verzögerung in sehr starkem Maße. Dabei zeigt sich, daß die Verzögerungswerte bis zu einem Verkehrsangebot von 70 MBit/s (entsprechend 70% Auslastung) nur sehr moderat ansteigen, während ein überproportional starker Anstieg bei etwa 75 MBit/s einsetzt. Das zeitgesteuerte Verfahren läßt einen stabilen Betrieb des FDDI-Rings selbst im Hochlastbetrieb zu, vgl. Abbildung 6-37

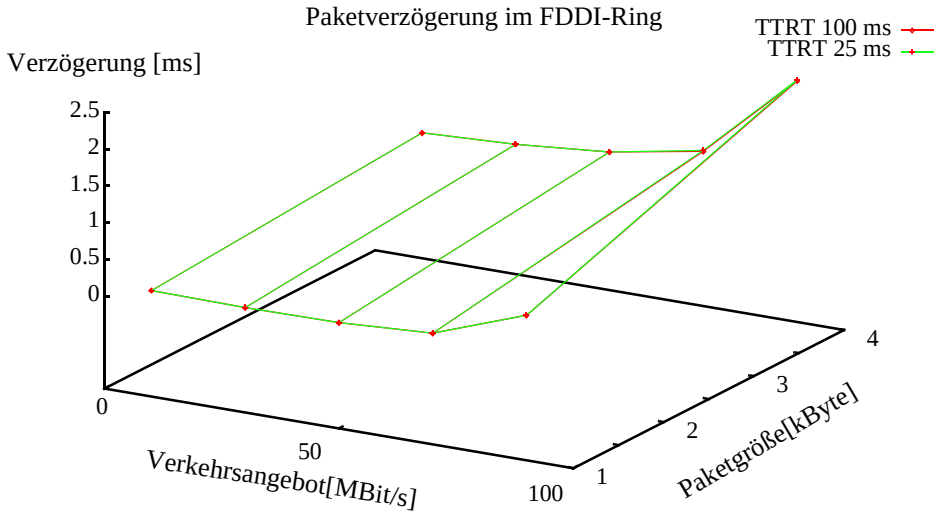


Abbildung 6-37 **Paketverzögerung in einem FDDI-Ring**

Da die Verzögerung eines Paketes mit zunehmender Größe schon alleine durch die wachsende Bedienzeit zunimmt und daher der Vergleich der Paketverzögerungszeiten irreführend ist, sind die Auswirkungen der Paketgrößenwahl auf die Übertragungszeit einer größeren Datenmenge untersucht worden. Für ein MByte Anwenderdaten ist die Übertragungszeit bei einer Target Token Rotation Time von 100 Millisekunden für Paketgrößen von 1000 Byte bis zu 4000 Byte und für die Verkehrsangebote von 10 MBit/s bis zu 90 MBit/s untersucht worden.

.

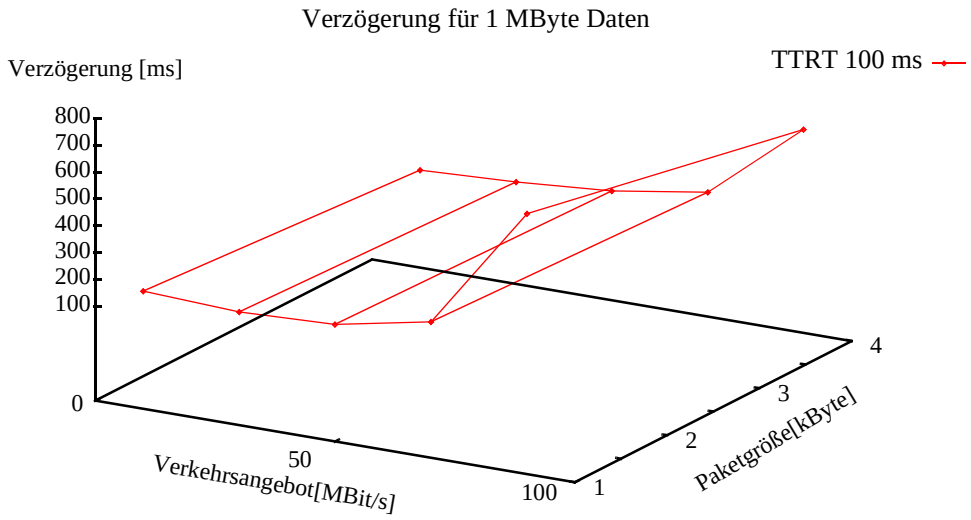


Abbildung 6-38 **Verzögerung für ein MByte Anwenderdaten**

Hier zeigt sich deutlich der Vorteil, der sich bei Verwendung längerer Pakete ergibt, da nicht nur die Gesamtverzögerung sinkt, sondern auch der Anstieg der Verzögerungswerte im Bereich von 75 MBit/s Verkehrsangebot verringert werden kann, siehe Abbildung 6-38

So steigt die Verzögerung in einem solchen FDDI-Ring für 1 MB Daten von 255 Millisekunden auf 718 Millisekunden bei Verwendung von Paketen der Größe ein KByte, während sie lediglich von 213 Millisekunden auf 506 Millisekunden steigt, wenn Pakete der Größe vier KByte verwendet werden.

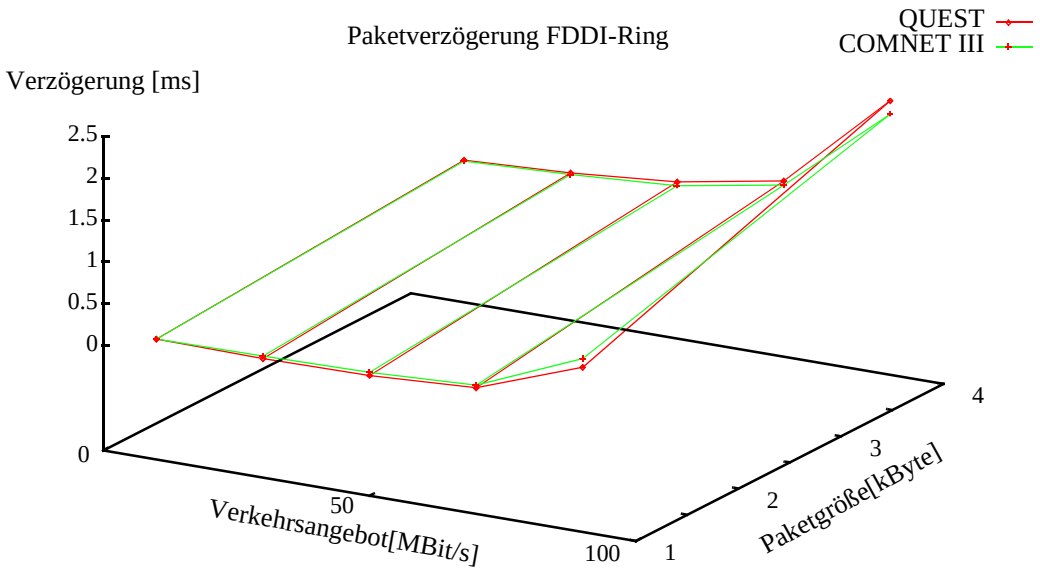


Abbildung 6-39 **Vergleich Modellergebnisse COMNET III vs. QUEST**

Die erzielten Ergebnisse und somit das erstellte Modell wurde wiederum mit Hilfe des COMNET III-Simulators und den dort vorhandenen vorgefertigten Modellbausteinen verifiziert. Wie

schon in der Studie des Zugangsprotokolls CSMA/CD konnte festgestellt werden, daß die Ergebnisse der Modelle nahezu identisch waren, vgl. Abbildung 6-39.

An dieser Stelle ist anzumerken, daß die Simulationszeit für das QSDL-Modell höher lag, als die Zeit, die zur Lösung des COMNETIII-Systems benötigt wurde. Das liegt daran, daß das QSDL jedes funktionale Detail, das in der formalen SDL-Spezifikation festgelegt worden ist, in der Performance-Simulation ebenfalls berücksichtigt. Damit ergibt sich der Vorteil, Performance-Problemen im Detail nachgehen zu können. Vor allem die Analyse transienten Verhaltens ist mit einem QSDL-Modell sehr gut durchführbar. So kann ein QSDL-System zum Beispiel durch entsprechende Quellen getriggert zunächst gezielt in eine Überlastsituation manövriert werden. Danach können die Quellen abgedreht werden, um dann zu beobachten, ob das System wieder in einen stabilen Zustand gelangt und falls ja, wie lange dies dauert. Erholt sich das System nicht von dieser Streßsituation, kann durch eine genaue Beobachtung aller beteiligten Prozesse überprüft werden, an welcher Stelle das Problem auftritt.

Der Gesamtaufwand für Modellerstellung und Modellösung ist bei Verwendung des QSDL-Ansatzes vergleichbar mit dem Aufwand, der bei der Verwendung von klassischen Performance-Tools zu leisten ist. Während letztere einen wesentlich höheren Aufwand für die Abstraktion bei der Ableitung des Performance-Modells erfordern, wird diese Abstraktion beim QSDL-Ansatz durch das Werkzeug QUEST vorgenommen. Dies ermöglicht die Durchführung von Performance-Studien auch durch solche Systementwickler, die nicht als ausgesprochene Performance-Experten gelten.

6.3 Zusammenfassung Kapitel 6

Die Performance-Analyse eines Transportsystems in einer geschichteten Kommunikationsarchitektur erfordert die Berücksichtigung des zeitlichen Verhaltens der unterliegenden Schicht-2-Dienste. In diesem Kapitel sind sowohl für Punkt-zu-Punkt-Verbindungen als auch für wichtige lokale Netzstandards Performance-Bausteine entwickelt worden. Die Probleme, das zeitliche Verhalten der Hardware-nahen Schichten in SDL auszudrücken, sind beschrieben und in einem zweiten Schritt mit Hilfe der Sprachelemente von QSDL gelöst worden.

Die modellgestützte Analyse auf Grundlage der formalen Spezifikation ermöglicht die unter Performance-Aspekten günstigste Wahl der Betriebsparameter. Diese Analysen konnten ohne eine teure Implementierung und anschließender Vermessung durchgeführt werden. Somit läßt sich nicht nur der Aufbau und die Pflege teurer Meßumgebungen vermeiden, sondern auch eine Vielzahl an Experimenten ausführen, ohne die realen Systeme aufbauen zu müssen.

Der Vergleich der QUEST-Modelle mit klassischen Performance-Modellen eines kommerziell verfügbaren Netzwerkplanungswerkzeugs haben gezeigt, daß nahezu identische Ergebnisse erzielt werden. Damit konnte sowohl die Lösung der Probleme erreicht als auch die Verwendung der QUEST-Methode zur entwurfsbegleitenden Bewertung von Schicht-2 Protokollen gezeigt werden.

Kapitel 7

Multimediaprotokolle im Internet

Das Internet wurde ursprünglich als reines Datennetz konzipiert und ist aus mehreren Gründen zur Unterstützung von multimedialen Realzeitdiensten nicht geeignet. Die verwendeten Standard-Transportprotokolle wie UDP/IP oder TCP/IP sind für zuverlässige Datendienste entwikkelt und optimiert worden. Der Versuch, jedes Paket fehlerfrei beim Empfänger abzuliefern, führt zu einer verzögerten Auslieferung der Daten und stört somit den isochronen Datenstrom multimedialer Anwendungen. Darüberhinaus ist die dem Internet zugrundeliegende *best-effort-Semantik* nicht ausreichend, die Anforderungen isochroner Datenströme zu erfüllen, zumal geeignete Dienstschnittstellen fehlen, die eine adäquate Spezifikation der gewünschten Dienstgüten ermöglichen.

Daher lassen sich in den letzten Jahren verstärkte Bemühungen erkennen, das Internet zu einem Multimedia-Netz zu erweitern. Einen ersten grundlegenden Ansatz dazu entwickelte die *Integrated Services Working Group* der *Internet Engineering Task Force* (IETF). Sie stellt in [19] eine *Internet Integrated Services Architecture* (IISA) vor, die die Grundlage für Multimedia-Anwendungen im Internet bildet. Die vorgestellte Architektur enthält ein erweitertes integriertes *Dienstmodell (integrated services model, IS-Modell)*, das auch die QoS-Anforderungen multimedialer Anwendungen berücksichtigt, sowie ein *Implementierungs-Framework*, das allgemeine Richtlinien enthält, wie die definierten Dienste implementiert werden sollten. Die Trennung in Dienst und Implementierungs-Framework ermöglicht eine Separierung der Diskussion über den Dienst und dessen Realisierung. Insbesondere können Implementierungsansätze modifiziert werden, ohne den Dienst dadurch zu ändern.

Das IS-Modell enthält neben dem bereits bekannten *best-effort-Dienst* einen sog. *controlled load service*, sowie einen *predictive service* und einen *guaranteed service*. Der *best-effort*-Dienst und der *controlled-load*-Dienst sind für solche Anwendungen gedacht, die weder harte noch weiche Anforderungen an die maximale Ende-zu-Ende-Verzögerung haben. Dabei hält der *controlled-load*-Dienst die Auslastungen in den Netzen niedrig, so daß dieser Dienst ein Verhalten aufweist, wie es in wenig ausgelasteten Netzen auch von *best-effort*-Diensten zu erwarten ist. Während zur Realisierung eines *best-effort*-Dienstes keinerlei Zugangskontroll- und Überwachungsfunktionen implementiert werden müssen, sind zur Realisierung von *controlled-load*-Diensten Zugangskontroll- und Überwachungsfunktionen zu implementieren.

Im Gegensatz dazu unterstützen sowohl *guaranteed service* als auch *predictive service* solche Anwendungen, die entweder harte oder zumindest weiche Realzeitanforderungen haben. Während Anwendungen mit harten Realzeitanforderungen eine garantierte obere Schranke für die maximale Ende-zu-Ende-Verzögerung und eine garantierte minimale Bandbreite benötigen, können Anwendungen mit weichen Realzeitanforderungen kurzzeitige Verletzungen dieser Werte tolerieren. Zur Unterstützung dieser Dienste sind Netzzugangs- und Überwachungsfunktionen, aber auch Scheduling- und Reshaping-Funktionen in den Netzelementen zu implementieren.

Das Implementierungs-Framework beschreibt einen Ansatz zur Realisierung der vorgestellten Dienste. Dabei sind vor allem die Dienste zur Unterstützung von Anwendungen mit Realzeitanforderungen durch geeignete Funktionen zu implementieren. Damit die strikten Anforderungen eingehalten werden können, müssen in den beteiligten Knoten die notwendigen Verarbeitungskapazitäten reserviert werden. Dazu sind die Anforderungen geeignet zu spezifizieren, um dann entlang des gewünschten Kommunikationspfades die Kapazitäten zu reservieren. In den einzelnen Knoten sind dann aufgrund der lokalen Informationen die Entscheidungen zu treffen, ob die gewünschte Verbindung eingerichtet werden kann. Während der existierenden Verbindung ist dafür zu sorgen, daß sich die Anwendungen an die vereinbarten Verkehrscharakteristiken halten, und daß die vereinbarten QoS-Werte eingehalten werden. Die benötigten Funktionen werden in den beteiligten Knoten durch folgende vier Komponenten erfüllt:

Packet-Scheduler:	Er erzwingt die Einhaltung der vereinbarten QoS-Werte durch Zuordnung der Ressourcen zu den Paketen eines Stroms. Darüberhinaus sorgt er für zeitgemäße Verarbeitung der Pakete durch geeignete Scheduling-Strategien.
Packet-Classifier:	Er filtert die einkommenden Pakete und übergibt sie dem Packet-Scheduler zur weiteren Verarbeitung.
Admission Controler:	Er trifft die Entscheidung, ob ein neuer Strom mit den gewünschten QoS-Werten zugelassen werden kann, ohne bereits getroffene Zusagen zu verletzen.
Policing Controler:	Er überwacht den Strom, ob sich dieser an die vereinbarten Verkehrscharakteristiken hält.

Darüberhinaus wird ein Reservierungsprotokoll benötigt, mit dessen Hilfe die gewünschten Ressourcen in den Knoten angefordert werden können und somit eine Verbindung mit der gewünschten Qualität zwischen Quelle und Senke aufgebaut und verwaltet werden kann.

Die bekanntesten Reservierungsprotokolle sind das *Session Reservation Protocol* (SRP) [1], das *Internet Stream Protocol(ST-I bzw. ST-II)* in den unterschiedlichen Versionen [48], [33], das *RCAP* -Protokoll [9], das in der Tenet Gruppe am ICSI an der Universität von Berkeley entwikkelt wurde sowie das *Resource Reservation Protocol* (RSVP) [20], das als Internet Draft verabschiedet ist. Während das RCAP-Protokoll in eine eigenständige QoS-Architektur [45] eingebunden ist, stellen das RSVP und ST-II Ergänzungen innerhalb der bisherigen Internet-Protokoll-Architektur dar, vgl. Abbildung 7-1.

Neben den Reservierungsprotokollen haben sich im Internet Protokolle zur Unterstützung adaptiver Anwendungen entwickelt. Adaptive Anwendungen sind solche, die ihre Anforderungen innerhalb gewisser Grenzen an die zu erzielende Dienstgüte anpassen können. Das bekannteste Framework dieses Ansatzes stellt das *Real-Time-Protocol* (*RTP*) [97] zusammen mit dem zugehörigen Kontrollprotokoll (*RTCP*) dar, das sehr häufig direkt in die Anwendung integriert wird. Abbildung 7-1 stellt einen Ausschnitt aus der Protokoll-Architektur zur Multimedia-Unterstützung im Internet dar.

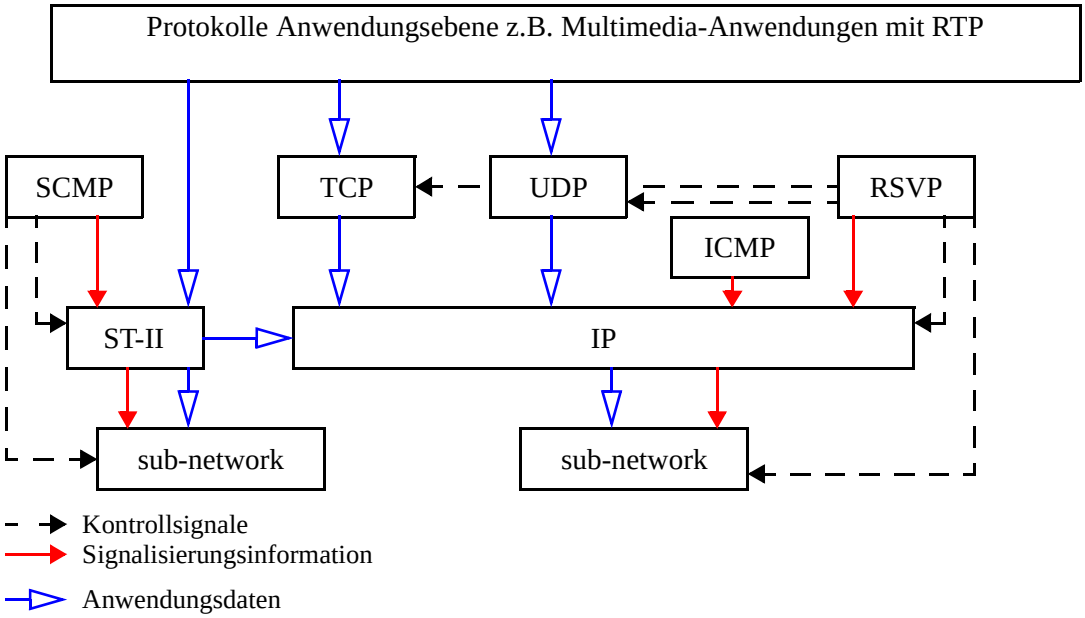


Abbildung 7-1 **Internet Protokoll-Architektur**

In den folgenden Abschnitten werden die wesentlichen Elemente der Protokolle zur Unterstützung multimedialer Anwendungen dargestellt.

7.1 Real Time Protocol

Das *Real Time Protocol* (RTP) ist in der *Internet Engineering Task Force* (IETF) entwickelt worden. RTP besteht aus zwei Teilen, einem Anteil zur Datenübertragung und einem Kontrollprotokoll (RTCP). Obwohl es häufig als Transportprotokoll bezeichnet wird, kennt RTP keine Verbindungen und kann sowohl über verbindungsorientierten als auch über verbindungslosen Netzwerkprotokollen eingesetzt werden. Sehr weit verbreitet ist der Einsatz von RTP über UDP, so daß RTP genaugenommen in der Anwendungsschicht angesiedelt werden müßte, vgl. Abbildung 7-2.

RTP und RTCP sind zwei lose gekoppelte Protokollrahmen, die zur Unterstützung isochroner Datenströme mit Realzeitanforderungen wie Audio und Video konzipiert worden sind. RTP ist kein vollständiges Protokoll, sondern wird in die Anwendung integriert. Es stellt folgende Mechanismen zur Verfügung, um Echtzeitdienste zu unterstützen:

- Zeitstempel,
- Sequenznummern,
- Payload-Typ-Identifikation,
- Beobachtung und Rückmeldung erzielter Dienstgüten (über RTCP),
- Unterstützung von Multicast, falls vom unterliegenden Protokoll unterstützt.

Da das Protokoll keine Mechanismen zur Aushandlung und Einhaltung von QoS-Parametern enthält, ist es nur für adaptive Anwendungen geeignet, die ihre benötigten Anforderungen dynamisch anpassen können. Zu diesen zählen unter anderem hierarchisch kodierte Video- und Audioströme, oder Videokonferenzen, bei denen möglicherweise nur jedes zweite oder jedes

dritte Bild übertragen wird, falls die Bandbreite kurzfristig nicht zur Verfügung steht. Bei weiterer Reduktion der verfügbaren Bandbreite kann wahlweise ein Standbild eingeblendet oder sogar ganz auf den Videokanal verzichtet werden. Zur vollständigen Protokoll-Spezifikation müssen zusätzlich sogenannte Profiles angegeben werden, in denen Ergänzungen, Modifikationen, sowie eine Menge von Payload-Typen und ihrer Abbildung auf Payload-Formate spezifiziert sind, vgl. Abbildung 7-2.

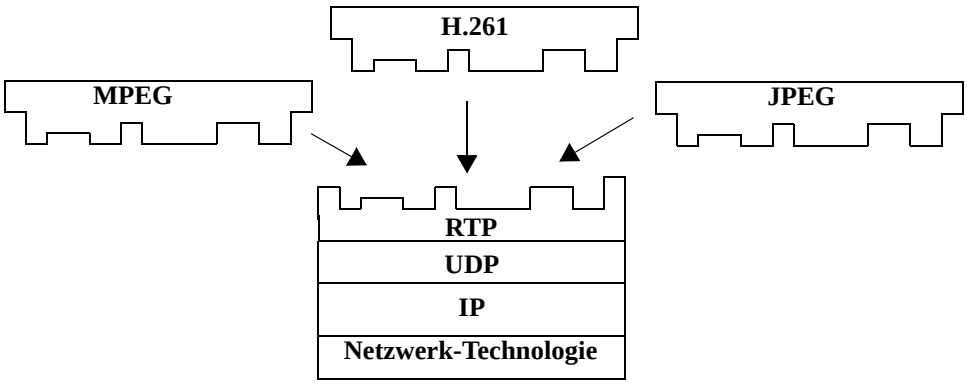


Abbildung 7-2 **RTP und Profiles zur Bildung eines Protokolls**

So existieren z.B. verschiedene Standards zur Kodierung von Videoströmen. Zu den Standards zählen H.261, MPEG oder JPEG. Das *Real Time Protocol* stellt allgemeine Mechanismen zur Verfügung, die Videoströme unabhängig von der verwendeten Kodierungstechnik unterstützen. Die kodierungsspezifischen Aufgaben müssen allerdings von bzw. in der Anwendung erledigt werden und sind daher in den *Payload- Profiles* enthalten.

RTP unterstützt sowohl Unicast- als auch Multicast-Netzwerkdienste. Jeder Rechner einer Multicast-Gruppe reserviert ein Port-Nummernpaar. Die Anwendungspakete werden innerhalb dieser Gruppe an den Port mit der niedrigeren (geraden) Nummer versendet, während die Kontrollpakete an den Port mit der höheren (ungeraden) Nummer verschickt werden.

Aus der IP-Adresse des Senders, der Port-Nummer und einer weiteren zufälligen Ziffernfolge wird eine für die Gruppe einheitliche Transportadresse gebildet, anhand derer der Informationsaustausch durchgeführt wird. Der Aufbau und die Verwaltung einer Multicast-Verbindung wird allerdings nicht innerhalb des *Real Time Protocols* geregelt, sondern als existent vorausgesetzt.

Die Multicast-Fähigkeit betrifft in RTP nicht nur die Datenpakete, siehe Abbildung 7-3, sondern auch die Kontrollpakete, die von den Empfängern der Datenpakete zu allen Teilnehmern der Multicast-Gruppe versendet werden, vgl. Abbildung 7-4. Die Verbreitung der Kontrollnachrichten an alle Beteiligten der Multicast-Gruppe hat den Vorteil, daß alle Teilnehmer darüber informiert werden, welche Last durch die Feedback-Nachrichten entsteht und so die Generierungsrate anpassen können. Darüberhinaus besteht der Vorteil für den Netzbetreiber darin, daß er z.B. seinen Management-Rechner als Mitglied dieser Multicast-Gruppe anmelden kann, um so einen Überblick über die erzielten Dienstgüten der Teilnehmer zu bekommen.

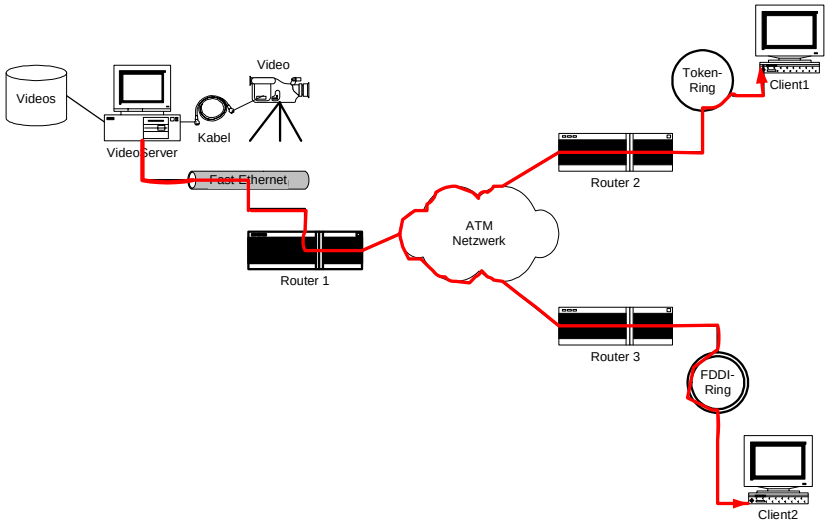


Abbildung 7-3 **Ausbreitung der Nutzdaten per Multicast**

Allerdings besteht die Gefahr, daß durch den starken Kontrollverkehr, der vor allem in sehr großen Gruppen entsteht, die beteiligten Netze sehr hoch belastet werden. Diese Gefahr besteht vor allem auch deswegen, da im Umfeld von RTP keinerlei Zugangskontrollen existieren, sondern lediglich die erzielten Dienstgüten beobachtet und rückgemeldet werden.

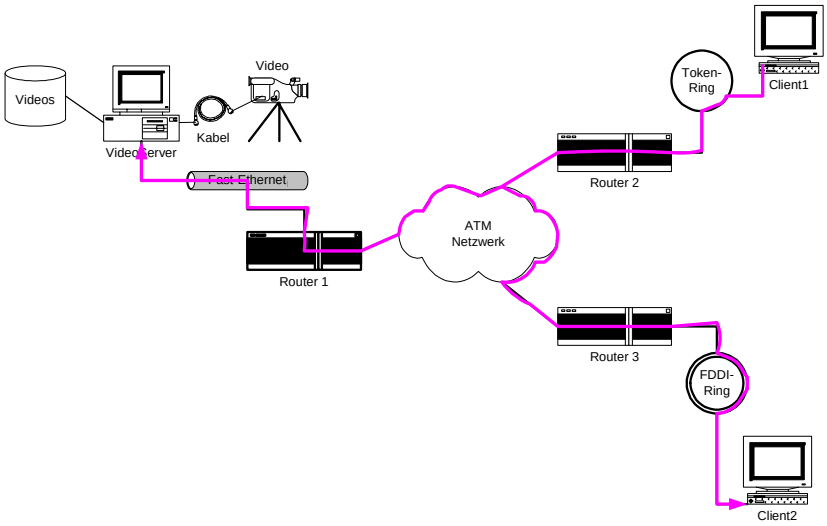


Abbildung 7-4 **Ausbreitung Kontrolldaten per Multicast**

Ein weiteres wichtiges Element der *Real Time Protocol*-Familie stellen sog. Übersetzer und Mixer dar. Diese Elemente können immer dann verwendet werden, wenn Teilnehmer an einer Realtime-Übertragung teilnehmen wollen, deren verwendete Bandbreite für sie nicht zur Verfügung steht, da sie z.B. lediglich über eine 64 kBit/s ISDN-Leitung verfügen, der gesendete Strom aber eine höhere Bandbreite erfordert. An dieser Stelle kommen Übersetzer ins Spiel. Bei ihnen handelt es sich um Zwischenrechner im “inneren” des Netzes, die Anwenderdaten von einem

Format in ein anderes transformieren, wobei Qualitätsverluste auftreten können, aber auf jedenfall Bandbreite eingespart wird. Dieser Ansatz verhindert, daß die gesamte Gruppe sich an den Möglichkeiten des schwächsten Teilnehmers ausrichten muß.

Eine ähnliche Funktion weisen Mixer auf. Allerdings wird hier nicht die Qualität eines einzelnen Stroms durch eine Umkodierung herabgesetzt, sondern es werden mehrere Ströme, die zu einer Anwendung gehören (z.B. Audioströme einer Konferenz) in einem Strom zusammengefaßt. Dabei lassen sich vor allem im Audiobereich die Bandbreiten reduzieren, ohne die Qualität herabzusetzen.

RTP ist zur Erzielung von garantierten Dienstgüten nicht geeignet, da es keine Reservierungen von Ressourcen ermöglicht. Vielmehr stellt es ein einfaches Protokoll dar, mit dessen Hilfe adaptive Anwendungen akzeptable Dienstgüten erreichen können, sofern die Auslastungen der beteiligten Komponenten innerhalb gewisser Schranken bleiben [25].

7.2 Resource Reservation Protocol

Das *Resource Reservation Protocol* (RSVP) stellt ein empfängergesteuertes Reservierungsprotokoll dar. Empfängergesteuert bedeutet, daß die Reservierungsanfragen von den Anwendungen in den Empfangsknoten initiiert werden. Diese Anfragen werden dann entlang des Datenpfades “rückwärts” bis zur Quelle propagiert. So werden Schritt für Schritt die benötigten Ressourcen in den Transitknoten und -netzen reserviert. *RSVP* basiert auf zwei wesentlichen Konzepten:

1. *Flows*: Sie beschreiben uni-direktionale Verkehrsströme zwischen genau einer Quelle und einer Menge von Empfängern. *Flows* werden anhand der IP-Adresse des Zielsystems und einer optionalen Port-Nummer identifiziert.
   Die Identifizierung kann verfeinert werden, indem z.B. die flow-label-Felder im IP-Header verwendet werden. Jeder Flow besitzt einen sog. *flowdescriptor,* der aus einer *flowspec* und einer *filterspec* besteht. Während die *flowspec* die gewünschten QoS-Werte enthält, wird mit Hilfe der *filterspec* ein Filter definiert, mit dessen Hilfe die Datenpakete des Flows in den Knoten aus dem Datenverkehr herausgefiltert werden können.
2. *Reservierung:* RSVP selbst stellt keine Reservierungsmechanismen zur Verfügung. Die zur Reservierung benötigten Funktionen in den Knoten setzt RSVP voraus. Die Reservierungswünsche werden durch *Resv*-Nachrichten vom Empfänger zum Sender des Datenstromes versendet. Der Pfad ist vom Sender durch *Path*-Nachrichten zuvor aufgebaut worden. Dieser Ansatz ermöglicht eine flexible Handhabung von Multicast-Verbindungen, da lediglich der Empfänger die von ihm gewünschte Dienstgüte verwalten muß und nicht der Sender die Wünsche aller seiner Empfänger verwalten muß.

RSVP kennt keine festen Verbindungen, die in den Zwischenknoten verwaltet werden müssen. Statt dessen müssen sowohl die *Path*-Nachrichten als auch die *Resv*-Nachrichten periodisch wiederholt werden, um den Weg und die reservierten Ressourcen zu bestätigen. Geschieht dies nicht, so werden die Ressourcen nach Ablauf von Timern freigegeben. Dieses Konzept wird als *Soft-State-Konzept* bezeichnet. Die Ressourcen können auch explizit durch *Teardown*-Nachrichten

freigegeben werden. Diese werden ebenso von den Empfängern initiiert wie die *Resv*-Nachrichten.

RSVP definiert 3 unterschiedliche Reservierungsarten, die als *Reservation Styles* bezeichnet werden.

1. *Fixed Filter*: Die reservierten Ressourcen stehen genau einem Flow zur Verfügung.
2. *Shared Explicit*: Die reservierten Ressourcen stehen einer Gruppe spezieller Flows zur Verfügung und müssen von den Mitgliedern der Gruppe geteilt werden.
3. *Wildcard Filter:* Die reservierten Ressourcen stehen einem allgemeinen Typ von Flows zur Verfügung. Alle Flows dieses Typs teilen sich die vorhandenen Ressourcen.

Neben der gemeinsamen Nutzung der reservierten Ressourcen durch eine Gruppe von *Flows*, können in Multicast-Verbindungen unterschiedliche Reservierungsanfragen verschiedener Empfänger innerhalb desselben *Flows* gemischt werden, vgl. Abbildung 7-5.

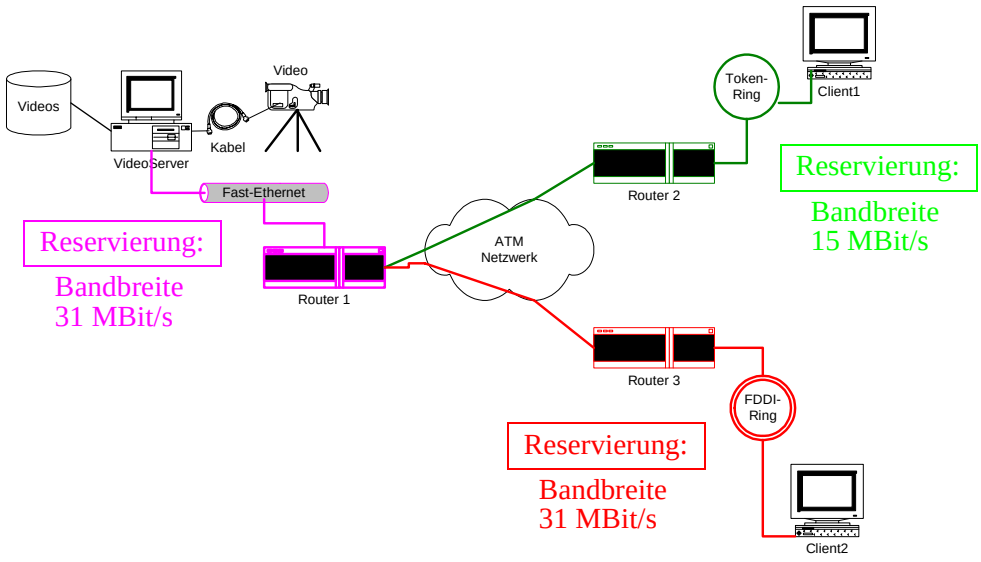


Abbildung 7-5 **Mischung zweier Reservierungswünsche**

Der Video-Client *Client1* ist über einen Token Ring an das ATM-Netzwerk angeschlossen. Er reserviert für die Videoübertragung vom Server eine Bandbreite von 15 MBit/s. Der Video-Client *Client2* hingegen ist über einen FDDI-Ring an das ATM-Netz angebunden. Er kann daher eine bessere Qualität empfangen und reserviert eine Bandbreite von 31 MBit/s für die Übertragung vom Server. Diese Reservierungswünsche werden im *Router1* gemischt, d.h. er stellt fest, daß beide Clients demselben *flow* angehören. Daher wird das Maximum der beiden Reservierungswünsche über alle Zwischenknoten an den Server weitergeleitet. Der Datenstrom wird nun auf dem Weg vom Server zu den Clients mit der höheren Bandbreite (Qualität) versendet. Im *Router1* werden dann die Ströme getrennt und entsprechend der gewünschten Qualitäten weitergeleitet. Somit können auf dem gemeinsamen Weg Ressourcen effizient genutzt werden.

RSVP ist auf die Unterstützung einer Reihe anderer Internet-Protokolle angewiesen, da es weder Mechanismen zum Verbindungsaufbau noch Mechanismen zum Beitritt zu bzw. zum Verlassen einer existierenden Verbindung bereitstellt. Darüberhinaus fehlt die Unterstützung zur Übertragung der Anwenderdaten.

Alle RSVP-Nachrichten werden in IP-Datagrammen übertragen. Da diese keinerlei Zuverlässigkeits- und Verzögerungsgarantien besitzen, sollten zur Versendung der RSVP-Kontrollinformationen Verarbeitungskapazitäten und Bandbreiten in den Netzen bereitgestellt werden.

Wünscht ein Client, einer bereits existierenden Multicast-Gruppe beizutreten, so nutzt er dazu das *Internet Group Management Protocol* (IGMP), genauer die IGMP *Group Membership Report Message*. Der Router im lokalen Netz des anfragenden Clients aktualisiert die zu der gewünschten Verbindung gehörende Informationsbasis. Damit auch alle anderen betroffenen Router ihre Informationen aktualisieren können, wird anschließend eine *OSPF Link State* Ankündigung im Netz verbreitet. Wenn alle beteiligten Router ihre Informationen aktualisiert haben, wird der anfragende Client in die Empfängerliste mitaufgenommen und erhält so auch die periodisch ausgesendeten *Path*-Nachrichten. Diese können dazu benutzt werden, den *Flow* zu identifizieren und nun die eigenen gewünschten Reservierungen mittels *Resv*-Nachrichten vorzunehmen.

Neben diesen funktionalen Festlegungen und der Einbettung in die Internet-Protokoll-Architektur wird im RFC 2205 ein Implementierungs-Framework für *Hosts* und auch für *Router* vorgeschlagen, vgl. Abbildung 7-6.

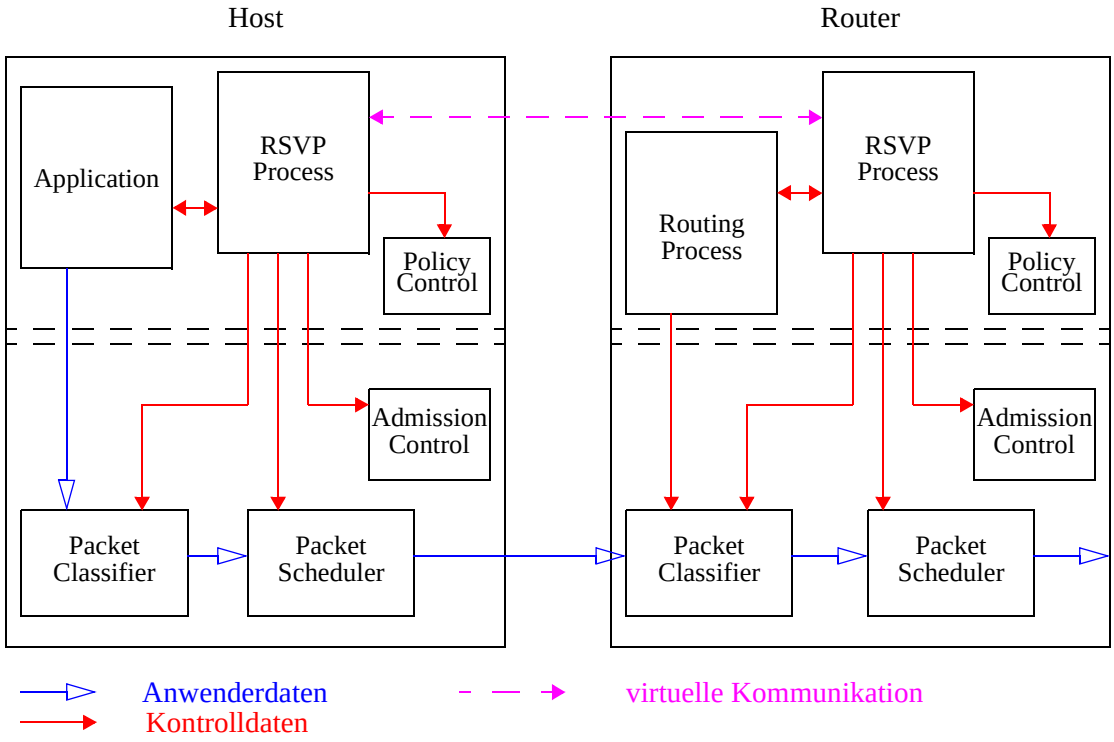


Abbildung 7-6 **RSVP in Hosts und Routern**

Wie die einzelnen Funktionen und deren Zusammenspiel geregelt wird, läßt die Spezifikation von RSVP offen. Die Existenz der Funktionen wird lediglich vorausgesetzt.

7.3 Internet Stream Protocol ST2

Während die bisher vorgestellten Protokolle in der Transportschicht oder höher anzusiedeln sind, wird nun ein Netzwerk-Protokoll, das *Internet Stream Protocol*, vorgestellt. Es ist das erste Kommunikationsprotokoll, das für Sprachübertragung über paketvermittelnde Netze entwickelt wurde. Die erste Version *ST-I* ist im Jahre 1979 verabschiedet worden [48]. Eine weiterentwikkelte Version ist als *ST-II* im Jahre 1990 als RFC1190 veröffentlicht worden. Im Jahre 1993 startete die IETF eine Arbeitsgruppe, die *ST-II* weiterentwickelten und ihre Arbeit mit Verabschiedung von *ST2*$^+$ als RFC1819 abschloß [34]. Die bisherigen Arbeiten führten zu einem experimentellen Status des Protokolls.

ST2$^+$ ist als Protokoll entworfen worden, daß zusammen mit IP die Netzwerkschicht auf den Knoten eines Internets bildet. Das Protokoll arbeitet verbindungsorientiert und ist für Anwendungen entwickelt worden, die garantierte Bandbreiten und kontrollierte Verzögerungen innerhalb des Netzwerkes benötigen. *ST2*$^+$ kann dazu benutzt werden, um Bandbreiten für Multimedia-Ströme durch Netzknoten zu reservieren. Diese Reservierung erlaubt zusammen mit geeigneten Netzzugangs- und Paket-Scheduling-Mechanismen in den Knoten die Einhaltung garantierter Dienstgüten für *ST2*$^+$-Anwendungen.

Durch Unterstützung von Multicast-Mechanismen kann zusätzlich die Netzlast reduziert werden. *ST2*$^+$ besteht analog zu IP aus zwei Protokollen. *ST* für den Datentransport und dem *Stream Control Message Protocol* (SCMP) für Kontrollfunktionen wie *Stream Management* und Ressourcenreservierung. SCMP-Pakete werden in ST-Paketen gekapselt und übertragen. *ST2*$^+$ und IP können parallel auf einem Knoten betrieben werden und nutzen denselben Dienstzugangspunkt zur Schicht-2.

Auf jedem Knoten ist ein ST-Agent angesiedelt, der IP- von ST2-Paketen anhand der ersten vier Bits unterscheidet. ST2-Pakete können in IP-Paketen gekapselt werden, um so durch Knoten geschleust zu werden, die ST2$^+$ nicht unterstützen. Allerdings ist dann für Ströme, die durch diese Knoten fließen, eine garantierte Dienstgüte nicht mehr zu gewährleisten.

Die Basis für ST2$^+$ stellen sog. Ströme (*Streams*) dar. Ströme haben genau eine Quelle und eine Menge von Zielen. Bei Strömen handelt es sich um uni-direktionale Verbindungen von der Quelle zu den Zielen. Lediglich die Kontrollnachrichten können von den Empfängern zur Quelle gesendet werden.

ST2$^+$ ist in eine umfassende Kommunikationsarchitektur eingebunden, die aus mehreren Modulen besteht. Da es sich bei *ST2*$^+$ um ein zweistufiges Kommunikationsmodell handelt, werden ein *Setup-Protokoll* für den Aufbau, die Verwaltung und den Abbau von *Streams* sowie ein Datentransferprotokoll für den Nutzdatentransfer über den etablierten Pfad benötigt. Daneben wird wie bei *RSVP* eine *Flow*-Spezifikation zur Beschreibung der Dienstgüteanforderungen von Strömen benutzt. Weiterhin ist zur Unterstützung von *ST2*$^+$-Anwendungen eine *Routing*-Funktion und ein lokaler *Ressource-Manager* auf jedem Knoten zur Verwaltung der Ressourcen notwendig.

In den offiziellen Dokumenten [34] werden lediglich das Setup-Protokoll, das Datentransferprotokoll und eine Flow-Spezifikation angegeben. Sowohl die Routing-Funktion als auch die lokalen Ressource-Manager werden nicht definiert, ihre Existenz aber vorausgesetzt.

Im Datentransfer-Protokoll (*ST-protocol*) wird das Datenformat festgelegt. Es liefert die Datenpakete an alle Empfänger entlang der etablierten Pfade. Datenpakete werden immer in der ausgehandelten Qualität ausgeliefert. Sie enthalten einen globalen *Stream Identifier*, der ihre

Zugehörigkeit zu einem bestimmten Strom festgelegt. Dieser *Identifier* wird auch vom Setup-Protokoll beim Verbindungsaufbau verwendet.

Das Setup-Protokoll etabliert, verwaltet und beendet isochrone Ströme. Die Ermittlung des Pfades obliegt der Routing-Funktion. Auf jedem Knoten, der auf dem Weg besucht wird, wird die Flow-Spezifikation dem lokalen Ressource-Manager vorgelegt. Die Entscheidung des Managers führt entweder zur Ablehnung des Verbindungswunsches oder zur Weiterverbreitung des Wunsches auf dem Weg zu den Empfängern.

Die Flow-Spezifikation ist eine Datenstruktur, die die QoS-Anforderungen der Anwendung enthält. Alle Operationen auf dieser Datenstruktur werden vom lokalen Ressource-Manager vorgenommen. Die Flow-Spezifikation, die innerhalb eines Stroms in den Knoten vorgelegt wird, ändert ihr Format nicht, allerdings können Flow-Spezifikation unterschiedlichen Formats verwendet werden.

Die Routing-Funktion versorgt das Setup-Protokoll mit den Informationen, die benötigt werden, um die Zielknoten zu erreichen. Dabei wird zunächst der nachfolgende Knoten (*nexthop*) auf dem Weg von der Quelle zum Ziel (*down stream*) ermittelt. Falls es sich um eine Multicast-Verbindung handelt und der Strom im aktuellen Knoten “geteilt” werden muß, so können auch mehrere Nachfolgeknoten durch die Routing-Funktion ermittelt werden.

Eine Route wird für die Lebensdauer einer Verbindung fest gewählt. Anwendungen, die einer existierenden Verbindung beitreten wollen, melden dies durch eine *Join-Request*-Nachricht an die Quelle. Diese kann dem Wunsch entsprechen und durch eine *Expand-Request*-Nachricht den neuen Zielknoten in die Verbindung integrieren. Durch die *Expand-Request*-Nachricht wird analog zum Aufbau einer Verbindung dafür gesorgt, daß die benötigen Ressourcen reserviert werden.

Der Ressource-Manager verwaltet die auf einem Knoten verfügbaren Betriebsmittel. Dazu zählen die Prozessoren auf allen Endsystemen und Routern, der Hauptspeicher, Datenspeicher, Netzwerkadapter sowie die Übertragungsleitungen zwischen den Knoten. Analog zum RSVP muß auch bei *ST2*$^+$ der Kontakt mit der LLC-Einheit aufgenommen werden, um die Reservierung von Links vorzunehmen, sofern dies von der verwendeten Link-Layer-Technologie unterstützt wird.

Details hierzu werden nicht festgelegt. Sogar die Schnittstelle zwischen Setup-Protokoll und lokalem Ressource-Manager ist nicht festgelegt, sondern lokal zu lösen.

7.4 Vergleich der Reservierungsprotokolle

Die Reservierungsprotokolle *RSVP* und *ST2*$^+$ sind für die Übertragung von digitalen Audio- bzw. Videodaten konzipiert worden. Diese Daten benötigen einen schnellen und effizienten Datentransfer, Multicast-Techniken zur Gruppenkommunikation und garantierte Dienstgüten bezüglich Bandbreite, Ende-zu-Ende-Verzögerung und Laufzeitschwankungen (Jitter). Daher wird der neben einem eher allgemein gehaltenen Vergleich der Protokolle ein spezieller Vergleich bezüglich dieser Kriterien durchgeführt.

ST2$^+$ ist ein vollständiges Internet-Protokoll, das sowohl über Kontrollfunktionen als auch über Funktionen zum Datentransfer verfügt. Es ist als Ersatz für das Internet-Protokoll (IP) gedacht, soll aber gleichzeitig mit diesem auf jedem Knoten arbeiten. RSVP hingegen ist lediglich ein Kontrollprotokoll, das mit Hilfe von IP Daten versendet. Es kann aber prinzipiell auch mit anderen Netzwerkprotokollen wie *IPx* zusammenarbeiten. *ST2*$^+$ ist eine vollständig integrierte

Lösung, die auch Informationen über verfügbare Ressourcen ermittelt und so zum Beispiel QoS-basiertes Routing unterstützt. Im Gegensatz zu *RSVP* ist *ST2*$^+$ vollständig verbindungsorientiert, d.h. vor dem eigentlichen Datentransfer ist erst eine Verbindung zu etablieren, und für jede existierende Verbindung sind Zustandsinformationen in allen beteiligten Knoten zu verwalten. Im Gegensatz dazu verwendet RSVP einen sog. *soft state* Ansatz, der darauf beruht, in den Zwischenknoten keine Zustandsinformation für die existierenden Ströme zu verwalten, sondern durch periodisch versendete Nachrichten die reservierten Ressourcen zu bestätigen. Diese Funktion wird von den Endsystemen ausgeführt.

Ein weiterer wesentlicher Unterschied zwischen *RSVP* und *ST2*$^+$ besteht darin, daß *RSVP* davon ausgeht, daß eine Menge heterogener Empfänger existiert, die an einem Multimedia-Strom teilnehmen. Daher sind alle Reservierungs- und Beitrittswünsche empfängergesteuert. Allerdings benötigt *RSVP* für diese Aufgaben Hilfsprotokolle.

ST2$^+$ hingegen ist senderorientiert und kann mit Hilfe des eigenen Kontrollprotokolls eigenständig Verbindungen aufbauen, verwalten und wieder abbauen.

Ein weiterer wesentlicher Unterschied zwischen beiden Ansätzen besteht in der Vorgehensweise zur gemeinsamen Nutzung von Ressourcen. Während *RSVP* durch die Verwendung von unterschiedlichen *Reservierungs-Styles* einen sehr einfachen Mechanismus zur gemeinsamen Nutzung von Ressourcen anbietet, geht *ST2*$^+$ davon aus, daß für zwei unterschiedliche Ströme auch disjunkte Mengen von Ressourcen reserviert werden müssen. Allerdings wird durch das *Stream Group-Konzept* ein Basismechanismus zur gemeinsamen Reservierung von Ressourcen durch Ströme bereitgestellt. Details hierzu entnehme man [34].

Beide Protokolle unterstützen *best-effort* Dienste. *ST2*$^+$ ist unter der Annahme eines fehlerfreien Betriebs, d.h. kein Zwischensystem fällt ganz aus, in der Lage Garantien für die zu erzielende Dienstgüte zu geben. *RSVP* hingegen ist dazu nicht in der Lage, da es keinen direkten Zusammenhang zwischen der Ressourcen-Reservierung, den Routing-Funktionen und dem Datentransferprotokoll gibt. Dadurch bedingt können Routen in der laufenden Übertragung geändert werden, was dazu führen kann, daß zeitweilig keine Datenübertragung möglich ist.

7.5 Zusammenfassung Kapitel 7

In diesem Kapitel sind die Ansätze der *Internet Engineering Task Force* (IETF) zur Weiterentwicklung des Internets zu einem Multimedia-Netz dargestellt worden. Dabei ist nicht nur die erweiterte Dienstarchitektur erläutert worden, sondern auch ein Überblick über vorgeschlagene Protokollrahmen bzw. neue Hilfsprotokolle gegeben worden, auf denen zukünftige Multimedia-Transportsysteme aufbauen können. Die Gemeinsamkeiten aber auch die vorhandenen Unterschiede der existierenden Ansätze sind dargelegt worden.

Da sowohl *ST2*$^+$ als auch *RSVP* Funktionen zur Zugangskontrolle, QoS-Aushandlung und QoS-Überwachung sowie Funktionen zur Reservierung von Ressourcen in den beteiligten Knoten voraussetzen, werden im folgenden Kapitel Ressourcen und lokale Manager vorgestellt, die die entsprechenden Funktionen im Kontext von (Q)SDL realisieren. Dabei wird insbesondere gezeigt, wie lokale Manager die unterschiedlichen Ressourcentypen verwalten können, um den existierenden Strömen die erforderlichen Kapazitäten zur Verfügung zu stellen.

Kapitel 8

Entwurf und Bewertung lokaler Ressource-Manager

8.1 Einführung

Zur geeigneten Unterstützung von Realzeitströmen werden auf allen beteiligten Knoten lokale Ressource-Manager benötigt, die die vorhandenen Kapazitäten verwalten und den Strömen entsprechend zuteilen. Passive Elemente wie Haupt- und Datenspeicher erfordern kein aufwendiges Ressource-Management. Sie werden für die Existenz einer Verbindung reserviert und den Strömen zugeteilt. Allerdings muß bei der Betrachtung der Hauptspeicherverwaltung die Auswirkung von Kontextwechseln berücksichtigt werden.
Aktive Ressourcen wie Zentralprozessor (CPU), dedizierte Signalprozessoren (DSP), Netzwerkadaptern, interne Busse, Platten, aber auch Übertragungsleitungen (Links) erfordern ein entsprechendes Ressource-Management, das die zeitlichen Aspekte der Nutzung durch die Anwendungsströme berücksichtigt. Im folgenden werden Management-Einheiten für aktive Ressourcen entworfen und deren Einfluß auf die Performance von Realzeitströmen untersucht.

8.2 Ressource-Management für Einfach-Links

Netzwerkbandbreiten zählen zu den kritischen Ressourcen für Multimedia-Ströme. Die wenigsten der heute verfügbaren Netztechnologien unterstützen die spezifischen Anforderungen isochroner Datenströme. Während zum Beispiel die meistverwendete LAN-Technologie, das Ethernet, keinerlei Aussagen bezüglich der zu erwarteten Verzögerung machen kann, können bei Token-Ringen durch Ergänzungen des Standardprotokolls die Verzögerungen beschränkt werden [82].

Im ISO-Ethernet hingegen, einer Variation des Ethernets, existieren 64 isochrone B-Kanäle, für die eine feste Verzögerung von unter einer Millisekunde garantiert werden kann. Auch die FDDI-Standards bieten ein integriertes Bandbreiten-Management an. Während die Verzögerung im FDDI-I-Standard zu Beginn der Ringinitialisierung durch die *Target Token Rotation Time* festgelegt werden kann, bietet der FDDI-II-Standard zusätzliche Unterstützung für isochronen Verkehr. Allerdings muß an dieser Stelle festgestellt werden, daß die Möglichkeiten zur Unterstützung isochroner Datenströme durch FDDI-II in der Regel in den betriebenen Netzen nicht genutzt

werden. Darüberhinaus können FDDI-I- und FDDI-II-Stationen nicht zusammen betrieben werden. Für Point-to-Point-Links, wie sie zum Beispiel für ISDN oder SONET-Verbindungen verwendet werden, läßt sich ein Bandbreiten-Management etablieren. Die wesentlichen Punkte sollen hier kurz skizziert werden, bevor die benötigten Prozesse und ihr Zusammenspiel im Detail erläutert werden.

Ausgangspunkt für den Entwurf eines lokalen Ressource-Managers ist das QSDL-System für Point-to-Point-Links, vgl. Abbildung 6-3 auf Seite 103, aus dem der Prozeß *Outdev* und die Maschine *MedMach* für den Block *P2P-Link* übernommen wurden. Sie übernehmen wie bisher die Übertragung der Link-Layer-Frames. Diese werden über den Datenpfad von der Anwendung, in diesem Fall eine MPEG-Audioübertragung, über die Policing-Einheit zur Übertragung bereitgestellt, vgl. Abbildung 8-1. Die Policing-Einheit ist dafür verantwortlich, zu überprüfen, ob die Anwendung die ausgehandelten Mindestabstände für Frames und die maximale Frame-Größe einhält.

Bevor die Anwendung mit der Nutzdatenübertragung beginnen kann, muß sie die gewünschten Bandbreiten beim lokalen Ressource-Manager (*LRM*) anfordern. Diese Rolle übernimmt in dem vereinfachten Szenario der Prozeß *Load_Generator*. Nach einer Pause, die dem Zwischenankunftsabstand von MPEG-Strömen entspricht, versucht er, den nächsten Strom zu etablieren. Dazu übergibt er dem Prozeß *LRM* die Informationen über die beabsichtigte Verkehrsintensität zusammen mit den Anforderungen über die maximal zulässige Verzögerung.

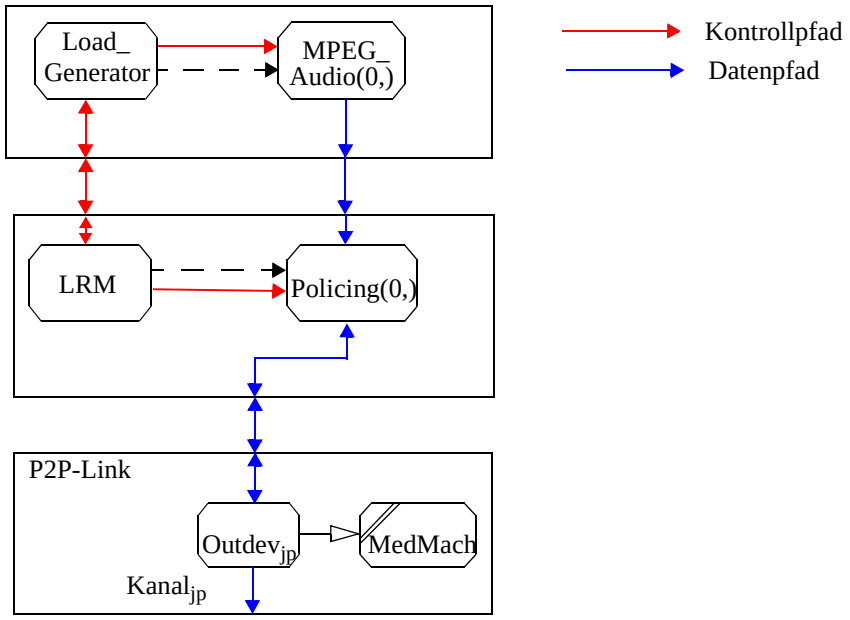


Abbildung 8-1 **Ressource-Management für Einfach-Links (Blockebene)**

Der Prozeß *LRM* prüft anhand der ihm übergebenen Informationen, ob die gewünschte Dienstgüte für diesen Strom eingehalten werden kann, ohne die Dienstgüten für die bisher existierenden Ströme zu verletzen. Ist dies möglich, wird dem Prozeß *Load_Generator* eine positive Antwort gesendet. Gleichzeitig wird vom Prozeß *LRM* eine Instanz des Prozesses *Policing* erzeugt, die den neuen Strom überwacht. Der *Load_Generator* erzeugt dann eine Instanz des Prozesses *MPEG_Audio*, der seinerseits einen Verkehrsstrom generiert, dessen Verkehrscharakteristik mit dem dem *LRM* bekanntgegebenen Wunsch übereinstimmt. Mit Hilfe dieses vereinfachten Szenarios kann gezeigt werden, daß die benötigten Ressourcen zwar nicht direkt reserviert werden

können, daß aber durch eine entsprechende Verwaltung der Ressource Bandbreite sowohl der gewünschte Durchsatz als auch die Verzögerung garantiert werden können.

8.2.1 Verkehrscharakteristik und Dienstgüte

Zur Angabe der gewünschten Dienstgüten und zur Beschreibung der Verkehrscharakteristik wird eine leichte Modifikation der Standard-Flow-Spezifikation benutzt, die jede Implementierung des Reservierungsprotokolls *ST2*$^+$ unterstützen muß. Die Modifikation besteht darin, die Angaben über die QoS-Klasse und die Priorität (precedence) nicht zu betrachten, da die Kanäle keine Priorisierung zulassen und grundsätzlich die QoS-Klasse *guaranteed* unterstützt werden soll. Auch das Feld *DesMaxDelayRange* wurde nicht übernommen. Lediglich der zu erwartende maximale und minimale Wert für die Verzögerung werden berücksichtigt, vgl. Abbildung 8-2.

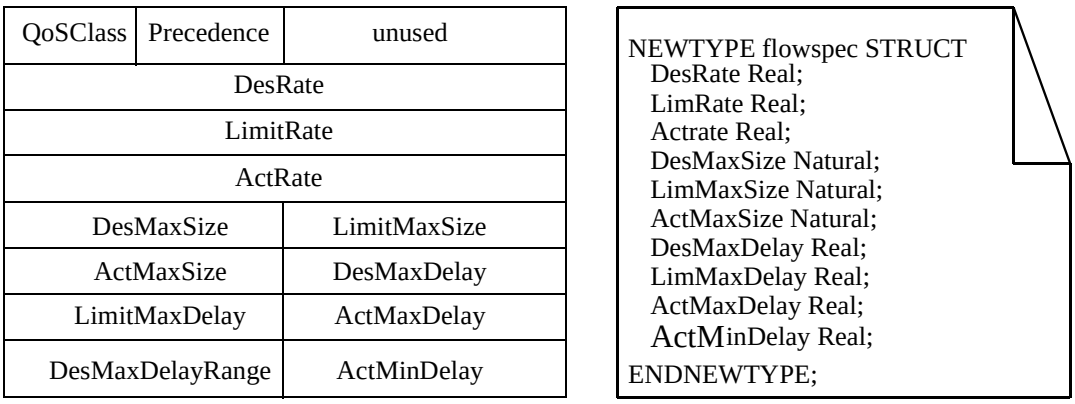


Abbildung 8-2 **ST2$^+$-Flow-Spezifikation und SDL-Newtype flowspec**

Die von einer MPEG-Audio-Quelle erzeugbaren Lastmuster werden durch die Parameter Abtastrate, Layer-Nummer und Zielbandbreite festgelegt, vgl. Abbildung 5-27 auf Seite 95. Die Verkehrscharakteristik wird durch diese drei Parameter von der Anwendung definiert. Die Parameter müßten dann z.B. durch sog. QoS-Mapper[a] auf die benötigten Parameter der verwendeten Flow-Spezifikation abgebildet werden. So ergibt sich für eine gewünschte Zielbandbreite von 160 kbps bei einer Abtastrate von 44.1 kHz für ein Layer-II-kodiertes Signal eine Frame-Rate von 39 Frames/s und eine maximale Frame-Größe von 520 Byte. Bei einer Zielbandbreite von 128 kbps kann der Wert der Frame-Größe bei gleichbleibender Abtastrate auf 416 Byte reduziert werden. Für die hier durchgeführten Untersuchungen sind folgende Werte in die Flow-Spezifikation eingetragen worden:

DesRate	39.0	gewünschte Rate in Frames/s	LimRate	39.0	niedrigste Senderate in Frames/s
DesMax-Size	520	gewünschte max. Frame-Größe in Byte	LimMax-Size	416	kleinste max Frame-Größe in Byte
DesMax-Delay	120	gewünschte max. Ende-zu-Ende-Verzögerung in ms	LimMax-Delay	150	größte max. Ende-zu-Ende-Verzögerung in ms

a. QoS-Mapper sind Einheiten, die Dienstgüteparameter zwischen unterschiedlichen Schichten transformieren. So muß zum Beispiel die Angabe Video-Frames pro Sekunde an der Schnittstelle zum Transportsystem in Pakete/s umgewandelt werden.

Die Werte für die Verzögerungen (*DesMaxDelay*, *LimMaxDelay*) sind in Millisekunden (ms) angegeben und geben eine obere Schranke für die Ende-zu-Ende-Verzögerung an. Der Wert von 150 ms entspricht dem Vorschlag aus [31] für die Verzögerung von Audioströmen. Für die Funktion des lokalen Ressource-Managers haben die (willkürlich) gewählten Zahlen nur insofern eine Bedeutung, daß nicht schon ein einziger Strom die zur Verfügung stehenden Ressourcen überfordern darf, da in diesem Falle kein Strom etabliert werden kann.

Im folgenden Abschnitt wird nun ein Ressource-Manager vorgestellt, mit dessen Hilfe Point-to-Point-Links in die Lage versetzt werden, isochrone Ströme geeignet zu unterstützen.

8.2.2 Lokale Ressource-Manager

Der lokale Ressource-Manager verwaltet die Ressource Bandbreite und garantiert den existierenden Strömen die Einhaltung der von ihnen gewünschten Dienstgüten bezüglich Durchsatz und Verzögerung. Dazu ist es notwendig, die Auslastung aber auch die Verzögerung durch den Link zu kontrollieren. Während bei der Verwaltung von Prozessoren für Realzeit-Scheduling gezeigt werden kann, daß die Begrenzung der Auslastung auf 80% als Kriterium zur Zulassung neuer Tasks ausreicht [102], soll durch eine kurze Beispielrechnung gezeigt werden, daß dies im Falle des FIFO-Links nicht ausreicht.

Beispiel 8 -1: **Performance-Werte eines FIFO-Links**

Kanal mit Bandbreite 1 MBit/s:

1 Strom mit 20 P/s und einer Paketgröße von 1000 Byte
16 Ströme mit 4 P/s und einer Paketgröße von 1250 Byte

Auslastung durch Strom 1 :	$\frac{20 \cdot 8000}{1000000} = 0,16$	Bedienzeit für Paket aus Strom 1:	8 ms
Auslastung durch Strom 2-17 :	$\frac{4 \cdot 16 \cdot 10000}{1000000} = 0,64$	Bedienzeit für Paket aus Strom 2-17:	10 ms
Gesamtauslastung: 80%		Verweilzeit Worst-Case:	$16 \cdot 10 + 1 \cdot 8 = 168$ ms
		Abstand zweier Pakete Strom 1:	50 ms
		Abstand zweier Pakete Strom 2-17:	125 ms

Obwohl der angenommene Kanal mit einer Bandbreite von einem MBit/s durch die 17 vorhandenen Ströme zu 80% ausgelastet ist, kann es im ungünstigsten Fall zu einer Verzögerung einzelner Pakete von 168 Millisekunden kommen, die deutlich über dem Wert für den Zwischenankunftsabstand der Pakete (50 ms oder 125 ms) liegt. Somit ist eine zeitgetreue Abarbeitung der Ströme nicht gewährleistet, wenn lediglich die Auslastung kontrolliert wird.

Zur Beschränkung der Verzögerung von Paketen auf einen Wert unterhalb der Zwischenankunftsabstände muß die Anzahl der maximal zulässigen parallelen Ströme beschränkt werden.

Sei *MaxSize* die größte zugelassene Paketgröße in Bit und *Speed* die vorhandene Bandbreite des Kanals in Bit/s. Beschränkt man die Anzahl der parallelen Ströme auf N, so ergibt sich die obere Schranke $\hat{VZ}$ für die Verzögerung zu:

$$\hat{VZ} = \frac{MaxSize}{Speed} \cdot N \qquad \text{(Gl. 1)}$$

Die Verweilzeit nach (Gl. 1) kann durch die maximal zulässige Auslastung ρ des Kanals wie folgt nach oben abgeschätzt werden:

Falls mit $BZ = MaxSize/Speed$ die maximale Bedienzeit eines Paketes bezeichnet wird, so ergibt sich die Kapazität eines Kanals zu $Kap = 1/BZ$. Wird diese Kapazität ausgeschöpft, so ergibt sich eine Auslastung von 100%. Beschränkt man die zulässige Auslastung im laufenden Betrieb auf ρ, so nutzt man die reduzierte Kapazität $Kap_{red} = \rho \cdot Kap$.

Für die resultierende Antwortzeit ergibt sich der *Worst-Case* bei Ausnutzung der maximal möglichen Parallelität, d.h. im beobachteten Intervall wird je Strom nur ein Paket übertragen. Damit läßt sich die Anzahl der parallelen Ströme mit Hilfe von (Gl. 2) abschätzen.

$$N' = Kap_{red} \cdot |Intervall| = \rho \cdot Kap \cdot |Intervall| \qquad \text{(Gl. 2)}$$

Damit läßt sich dann die maximale Verweilzeit eines Pakets wie folgt berechnen:

$$VZ_{max} = N' \cdot BZ = \rho \cdot Kap \cdot |Intervall| \cdot BZ \qquad \text{(Gl. 3)}$$

Da die Bedienzeit *BZ* dem Kehrwehrt der Kapazität *Kap* entspricht, kürzen sich diese Werte heraus und (Gl. 3) reduziert sich auf $VZ_{max} = \rho \cdot |Intervall|$.

Der lokale Ressource-Manager verwaltet die obere Grenze der Auslastung und die maximale Anzahl zulässiger Ströme. Aus diesen zum Systemstart festgelegten Parametern bestimmt er die obere Schranke für die Verzögerung eines Pakets durch den Link gemäß (Gl. 1). Außerdem werden die aktuelle Anzahl existierender Ströme und die aktuelle Auslastung jeweils mit Null initialisiert. In jedem Fall, in dem der Manager mit einer neuen Flow-Spezifikation konfrontiert wird, durchläuft er einen mehrstufigen Test, vgl. Abbildung 8-3.

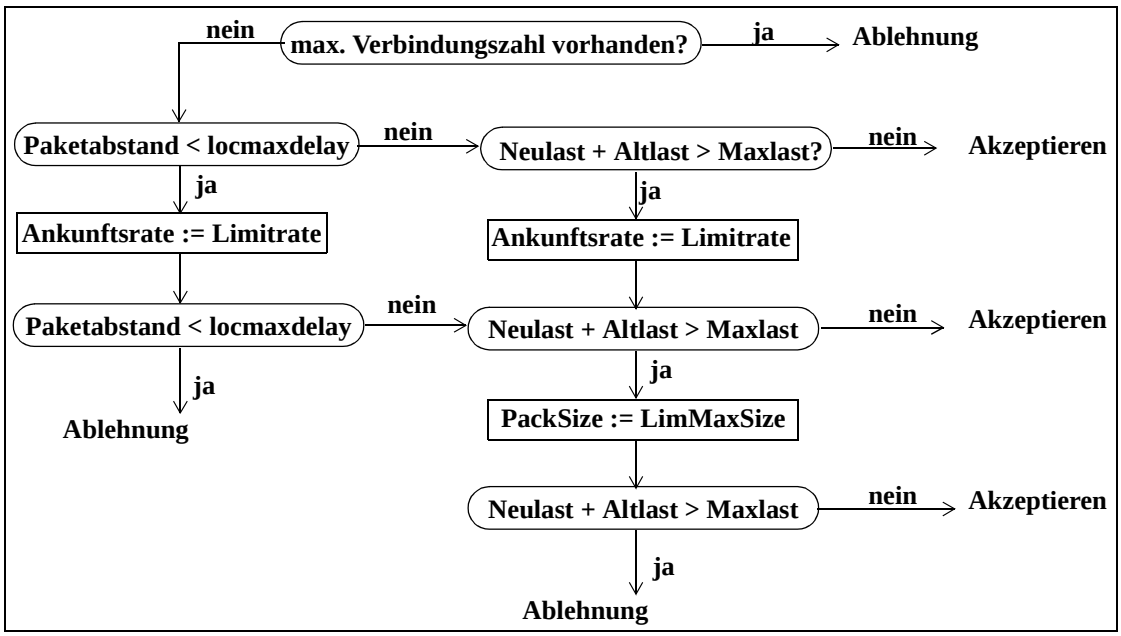


Abbildung 8-3 **Testschema zur Kanalzugangskontrolle**

Zunächst wird überprüft, ob die maximale Anzahl paralleler Sröme erreicht ist. Falls ja, wird der Verbindungswunsch sofort abgelehnt, um nicht die Dienstgütezusagen der existierenden Ströme zu verletzen.

In der nächsten Stufe wird überprüft, ob der gewünschte Zwischenankunftsabstand des neuen Stroms kleiner ist als die zu erwartende Verzögerung in diesem Knoten. Ist dies der Fall, wird dieser Test mit dem Wert *LimitRate* wiederholt. Fällt auch dieser Test positiv aus, muß die Verbindung abgelehnt werden.

Falls der Zwischenankunftsabstand der Pakete größer ist, als die zu erwartende Verzögerung, so wird getestet, ob die Auslastung des Kanals mit dieser neuen Verbindung die obere Schranke überschreitet. Ist dies nicht der Fall, kann die Verbindung etabliert werden. Überschreitet die neue Verbindung mit ihren Maximalforderungen diese obere Schranke, wird zuerst die Ankunftsrate und falls dies auch noch nicht ausreicht, auch die maximale Paketgröße reduziert. Führen alle Reduktionen nicht zur Einhaltung der oberen Auslastungsschranke, so wird die Verbindung abgelehnt. Sie kann zugelassen werden, falls eine der Kombinationen aus Paketrate und Paketgröße dazu führen, daß die maximal zulässige Auslastung nicht überschritten wird.

Die zulässigen Werte für Paketrate und Paketgröße werden in die Felder *ActRate* und *ActMaxSize* der Flow-Spezifikation eingetragen. Die Bedienzeit des Paketes wird als *ActMinDelay* und der Wert für die maximal zu erwartende Verzögerung wird als *ActMaxDelay* in die Flow-Spezifikation eingetragen und an den Prozeß *Load_Generator* zurückgegeben. Im Rahmen des Reservierungsprotokolls $ST2^+$ würde die so modifizierte Flow-Spezifikation an die Nachfolgeknoten weitergereicht, wo sie wieder dem lokalen Ressource-Manager präsentiert würden.

8.2.3 Experimente und Ergebnisse

Das in Abbildung 8-3 verwendete Testschema zur Entscheidung, ob ein neuer Strom mit den gewünschten Parametern etabliert werden kann, verwendet sehr restriktive Schranken. Daher soll nun untersucht werden, welche Dienstgütemerkmale für MPEG-Audioströme mit den gewählten Schranken eingehalten werden können. Gleichzeitig soll nachgewiesen werden, daß der oben vorgestellte Ressource-Manager in der Lage ist, Überlastsituationen abzuwenden und eine effiziente Nutzung der vorhandenen Ressourcen unter der Rahmenbedingung garantierter Verzögerungszeiten zu ermöglichen. Dazu soll der Prozeß *Load_Generator* versuchen, im Mittel fünf MPEG-Audio-Verbindungen pro Sekunde zu erzeugen. Die Zwischenankunftsabstände der aktiven Ströme seien exponentiell verteilt und die Dauer der Verbindung folge einer Normalverteilung[b] mit einem Mittelwert von 180 Sekunden und einer Standardabweichung von 25 Sekunden. Damit wäre der Link, dessen Bandbreite mit 1.5 MBit/s entsprechend eines T1[c]-Links angenommen wurde, stark überlastet, da er etwa neun parallele Verbindungen mit den in Abbildung 8-2 verwendeten Parametern unterstützen kann.

In einer ersten Experimentserie ist die mittlere Verzögerung eines Link-Layer-Frames in Abhängigkeit von der oberen Schranke für die zulässige Auslastung und der maximal zulässigen Anzahl paralleler Verbindungen untersucht worden, vgl. Abbildung 8-4. Dabei zeigt sich, daß die Verzögerung eines einzelnen Pakets im Bereich von drei bis fünf Millisekunden liegt. Insbesondere ist zu erkennen, in welchen Bereichen die Beschränkung auf die Auslastung den bestimmenden Einfluß auf die Verzögerung hat und in welchem Bereich eher die Beschränkung der Verbindungszahl die Verweilzeit beschränkt. Interessant ist, daß die Auslastung des Links selbst im Bereich von 90% nicht zu einem dramatischen Anstieg der Verweilzeit führt, wenn durch die Beschränkung der parallelen Ströme dafür gesorgt wird, daß diese hohe Auslastung durch wenige breitbandige Ströme erreicht wird.

b. Negative Werte werden verworfen, so daß es sich strenggenommen um eine abgeschnittene Verteilung handelt.

c. System von Bell Systems, in dem bis zu 24 Sprachkanäle per multiplexing gemischt werden können.

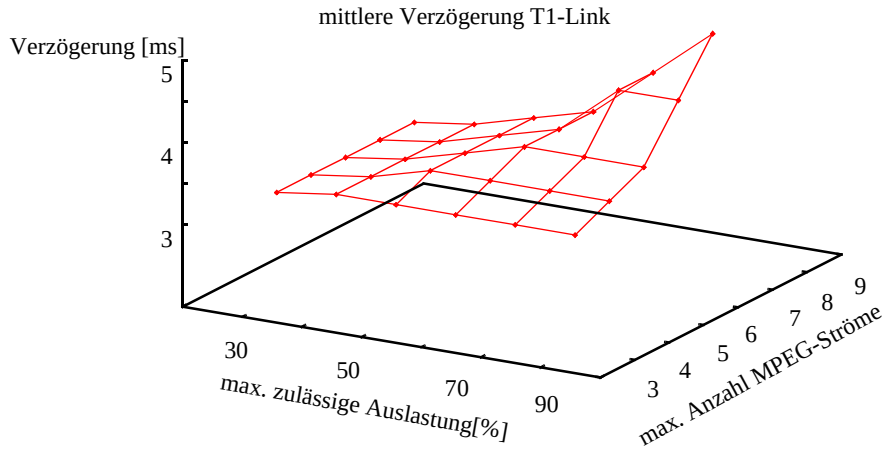


Abbildung 8-4 **Mittlere Verzögerung eines Frames durch einen T1-Link**

Diese Aussagen werden auch durch die Betrachtung der maximalen Verweilzeiten unter den obigen Randbedingungen bestätigt. So steigt die maximale Verzögerung der versendeten Frames zwar stärker an als die mittlere Verzögerung, die Einflüsse der beiden Kontrollparameter *maximal zulässige Auslastung* und *maximal zulässige Anzahl paralleler Ströme* sind jedoch nahezu identisch, vgl. Abbildung 8-5.

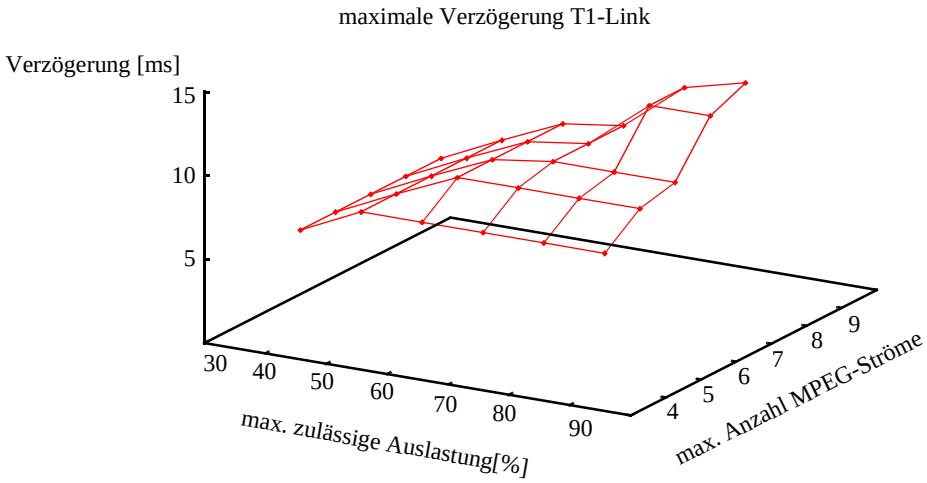


Abbildung 8-5 **Maximale Verzögerung eines Frames durch einen T1-Link**

Die maximale Verzögerung eines Frames ergibt sich bei einer oberen Schranke der Auslastung von 90% und einer Anzahl von neun parallelen Strömen zu 13.9 Millisekunden. Erhöht man die Anzahl der Ströme weiter, so kann keine Verbindung mehr etabliert werden, da die zu erwartende Verweilzeit größer ist, als der Zwischenankunftabstand der MPEG-Frames.

In Abbildung 8-6 ist zur Kontrolle die tatsächlich erreichte Auslastung des Links unter dem gegebenen Lastprofil dargestellt. Diese Graphik verdeutlicht, das eine effiziente Nutzung der vorhandenen Ressourcen unter Einhaltung von Verzögerungsschranken durch eine geeignete

Management-Komponente realisiert werden kann. Insbesondere soll hier festgehalten werden, daß diese Ergebnisse selbst bei einem weit über der vorhandenen Kapazität liegenden Verkehrsangebots eingehalten werden kann, da der vorgestellte Ressource-Manager eine geeignete Überlastabwehr durchführt. Der Vergleich der erzielten maximalen Verzögerung von 13.9 Millisekunden mit der *Worst-Case-Schranke* von 24.3 Millisekunden zeigt, wie pessimistisch der gewählte Ansatz ist. So könnte man in einem optimistischeren Szenario eine höhere Anzahl paralleler Ströme zulassen und die Verweilzeit wie folgt schätzen:

$$VZ_{max} = \frac{MaxSize}{Speed} \cdot N \cdot opt \qquad \text{(Gl. 4)}$$

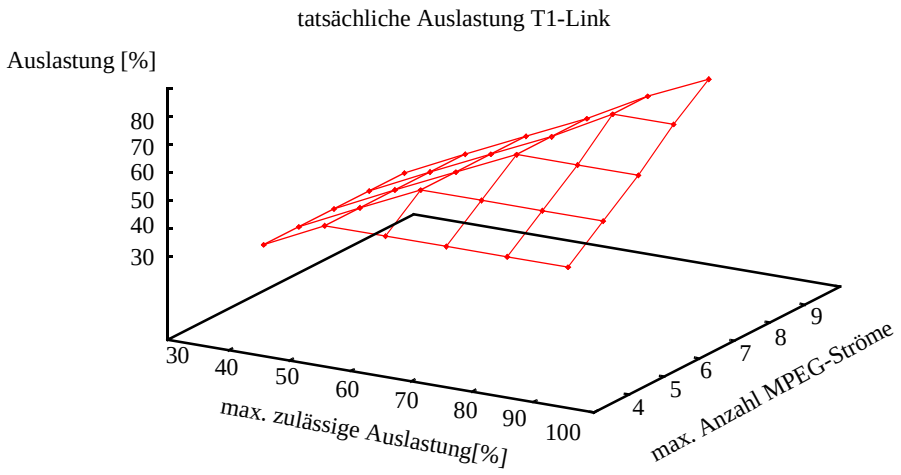


Abbildung 8-6 **Erreichte Auslastung eines T1-Links**

Dabei stellt *opt* einen sog. optimistischen Faktor ($0 < opt \leq 1$ dar, dessen Wert im obigen Fall 0.57 wäre. Zusammen mit der Kontrolle der zulässigen Auslastung kann man so eine größere Anzahl schmalbandiger Ströme zulassen, ohne die Verweilzeit der einzelnen Pakete zu stark zu erhöhen. Allerdings muß man dabei im Extremfall mit einer temporären Verletzung der Verweilzeit rechnen, die immer dann auftritt, wenn die Ankünfte der Ströme sehr stark gebündelt sind und es zu vermehrten Wartezeiten kommt. Eine detaillierte Analyse dieses Phänomens wird für sog. Mehrfach-Links im folgenden Abschnitt durchgeführt.

8.3 Ressource-Management für Mehrfach-Links

Um mit Hilfe von QSDL Mehrprozessorsysteme oder Mehrfach-Links zu modellieren, kann man die Anzahl der Server in den verwendeten QSDL-Maschinen erhöhen. Dabei muß man jedoch darauf achten, daß die Dienstaufrufe, die durch Verwendung solcher Maschinen parallel ausgeführt werden können, nicht durch Verwendung lediglich eines Prozesses zwangsweise serialisiert werden. So käme die erhöhte Anzahl von Bedienern nicht zum Tragen.

Würde man in dem Szenario aus Abbildung 8-1 lediglich bei der Maschine *MedMach* die Anzahl der Bediener von eins auf n erhöhen, so hätte diese Maschine zwar eine n-fach höhere Kapazität, da aber lediglich der Prozeß *OutDev* diese Maschine benutzt, und jeder Request-Aufruf den aufrufenden Prozeß blockiert, befände sich nie mehr als ein Auftrag in der Maschine, so daß die

erhöhte Kapazität nicht ausgeschöpft werden könnte. Daher ändert sich im folgenden die Struktur des Blocks *P2P-Link* entsprechend Abbildung 8-7.

Der Prozeß *LRM* verwaltet nun zusätzlich die Prozesse *OutDev*. Dazu erzeugt er sie in seiner Starttransition, um so ihre PIDs durch Auswertung von OFFSPRING zu erhalten. Zusätzlich zu den Funktionen, die er im Fall von Einfach-Links durchführt, muß er in diesem Szenario den neu zu etablierenden Strömen eine Instanz des Prozesses *OutDev* zuweisen.

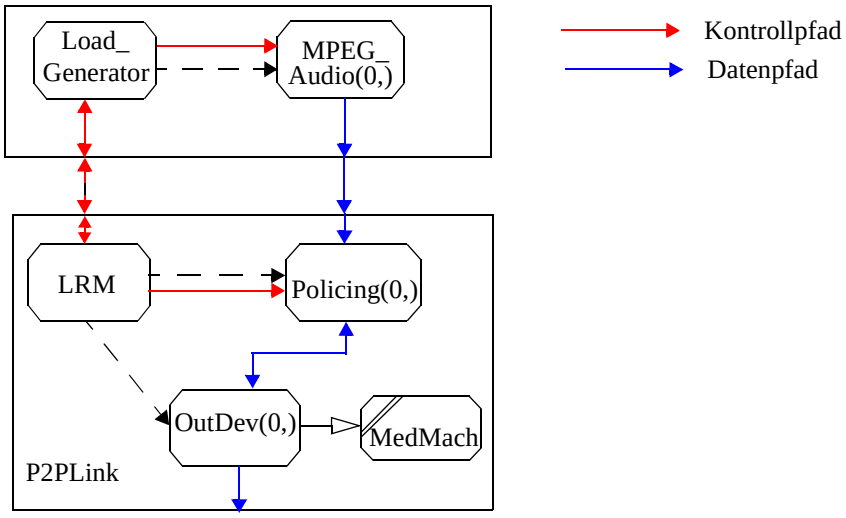


Abbildung 8-7 **Ressource Management für Mehrfach-Links (Blockebene)**

Falls die Anzahl der existierenden Ströme kleiner ist als die Anzahl der vorhandenen Bediener, wird der nächste freie Bediener zugewiesen. Sind alle Bediener belegt, wählt der lokale Ressource-Manager die Instanz aus, die momentan die wenigsten MPEG-Frames pro Zeiteinheit zu verarbeiten hat. Auf diese Weise wird eine annähernd gleichmäßige Lastaufteilung erreicht.

Analog zu den Überlegungen im Falle der Einfach-Links ergibt sich nun die obere Schranke für die Verweilzeit eines Frames bei Beschränkung der maximal zulässigen parallelen Streams zu:

$$VZ_{max} = \frac{MaxSize}{Speed} \cdot (\lceil N \; DIV \; n \rceil + 1) \qquad \text{(Gl. 5)}$$

Dabei bezeichnet *N* die Anzahl der zulässigen parallelen Ströme und *n* ist die Anzahl der vorhandenen Bediener. Wie in Abschnitt 8.2.2 bezeichnet *MaxSize* die maximale Größe eines Link-Layer-Frames und *Speed* die Geschwindigkeit eines einzelnen Bedieners der QSDL-Maschine.

8.3.1 Experimente und Ergebnisse

Auch im Falle des Multi-Links ist zu überprüfen, ob die zugesagten Dienstgüten eingehalten werden und der lokale Ressource-Manager in der Lage ist, den Link vor zu hoher Last zu schützen. Die folgenden Experimente sind für n =3 durchgeführt worden.

Der Prozeß *Load_Generator* versucht diesmal 45 MPEG-Ströme pro Sekunde zu etablieren. Die Lastcharakteristik eines einzelnen MPEG-Stroms hat sich gegenüber den Experimenten aus Abschnitt 8.2.3 nicht verändert. Die Zwischenankunftsabstände aufeinanderfolgender MPEG-Ströme sind als exponentiell verteilt angenommen worden. Die oberen Schranken für die zuläs-

sige Auslastung sind von 50% bis 90% variiert worden, während die Anzahl der gleichzeitig aktiven MPEG-Ströme von 12 auf bis zu 42 Ströme erhöht wurde.

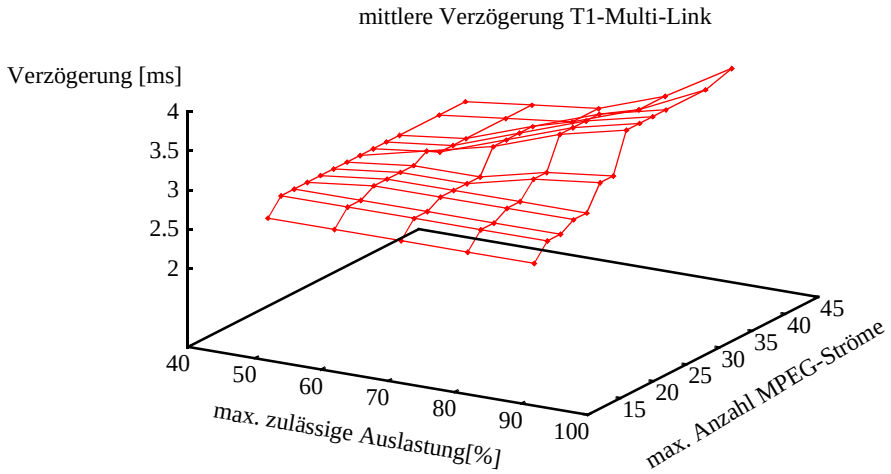


Abbildung 8-8 **Mittlere Verzögerung eines Frames durch einen T1-Multi-Link**

Wie schon im Fall des Einfach-Links konnte sowohl die mittlere (vgl. Abbildung 8-8) als auch die maximale Verzögerung (vgl. Abbildung 8-9) nach oben beschränkt werden. Während die mittlere Verzögerung unter vier Millisekunden bleibt, kann die maximale Verzögerung unter 13 Millisekunden gehalten werden.

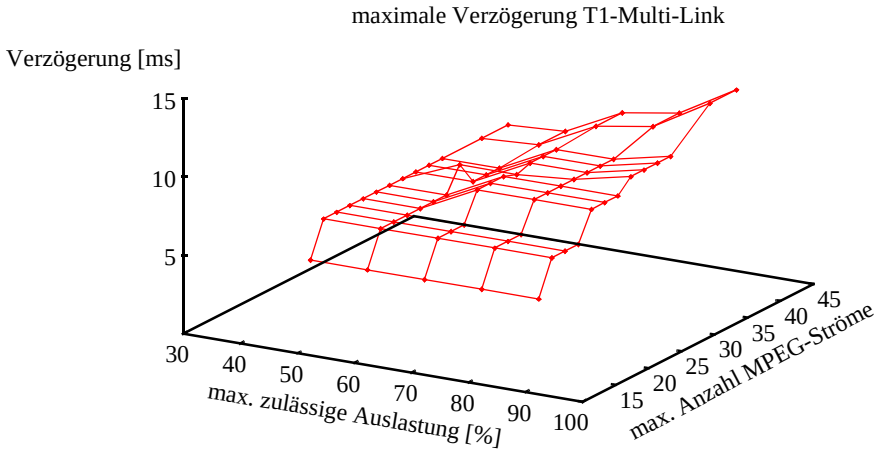


Abbildung 8-9 **Maximale Verzögerung eines Frames durch einen T1-Multi-Link**

Die Verwendung der sehr restriktiven Formel aus (Gl. 5) führt dazu, daß die Anzahl der parallelen Ströme nicht über 21 gesteigert werden kann, da die Auswertung der Formel bei N = 22 zu einer Verzögerung von über 0.026s führt und somit größer wird als der Abstand zwischen zwei aufeinanderfolgender Frames, der sich zu $t = 1/39 = 0,0256s$ ergibt.

Um eine höhere Anzahl paralleler Ströme zu ermöglichen, ist in den Experimenten mit 22 bis 38 parallelen Strömen mit einem optimistischen Faktor von $opt = 0,75$ gerechnet worden. Das

führt dann dazu, daß der lokale Ressource-Manager mehr Verbindungen zuläßt, als es nach den *Worst-Case*-Überlegungen zulässig wäre.

Die Resultate zeigen, daß die tatsächlich erzielten maximalen Verzögerungen mit zehn Millisekunden deutlich unter dem Wert für die Schranke nach (Gl. 5) liegt. Für N = 38 ergibt die Auswertung eine obere Schranke von 0.039 s. Da eine weitere Steigerung der Ströme mit dem Faktor von 0.75 nicht möglich ist, die erzielten Verzögerungen und auch die erreichten Auslastungen (vgl. Abbildung 8-10) aber die Unterstützung weiterer MPEG-Ströme als realistisch erscheinen lassen, ist für die weiteren Experimente dieser Faktor auf 0.50 abgesenkt worden. Diese zweimalige Absenkung des optimistischen Faktors führt zu den erkennbaren Terrassenbildungen in den Ergebnisgraphiken.

Selbst diese starke Abschwächung der oberen Schranke führt bei dem gewählten Lastszenario, exponentiell verteilte Zwischenankunftsabständen bei neuen Verbindungswünschen, nicht dazu, daß die Verzögerung von MPEG-Frames über 13 Millisekunden steigt. Damit liegen sie um zehn Millisekunden unter ihrer Verarbeitungs-Deadline.

Die Auslastung des Multi-Links läßt sich bis auf Werte von 71% steigern. Eine höhere Auslastung wird durch die Verwendung der Verzögerungsschranken im Zusammenspiel mit der Lastverteilung durch den Manager verhindert. In jedem Falle gelingt sowohl die Überlastabwehr als auch die zeitgerechte Auslieferung der MPEG-Frames am Ausgang des Multi-Links. Damit ist die Eignung von verwalteten Links zur Unterstützung isochroner Ströme gegeben.

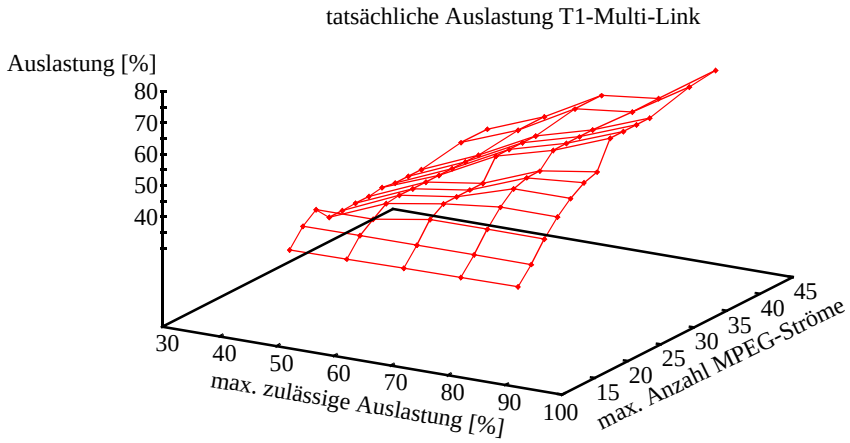


Abbildung 8-10 **Erreichte Auslastung eines T1-Multi-Links**

Diese Eignung setzt aber voraus, daß alle Datenströme, die den Link nutzen wollen, vorher ihre Übertragungswünsche anmelden. Diese Voraussetzung ist in der Realität nicht gegeben. Die bisher existierenden “klassischen” Datenströme werden wie bisher die vorhandenen Kapazitäten nutzen, ohne sich vorher anzumelden, da hierfür die notwendigen Schnittstellen nicht existieren. Zudem sind ihre Verkehrscharakteristiken *burst-artiger,* und die Verwaltung anhand der maximalen Senderate läßt eine effiziente Nutzung der Bandbreiten nicht erwarten.

Daher werden zukünftig neue Ströme mit garantierten Dienstgüten neben bisherigen *Best-Effort*-Strömen in multimedialen Transportsystemen existieren. Um die Dienstgüten mit den oben vorgestellten lokalen Ressource-Managern garantieren zu können, müssen entweder die verwalteten Bandbreiten den Realzeitströmen exklusiv zur Verfügung stehen, oder diese Ströme müssen mit höherer Priorität versehen werden als solche Ströme, die ihren Bedarf nicht anmelden. Im ersten Fall verhindert man, daß die Bandbreiten, die im Moment nicht für multimediale Ströme benötigt

werden, den Best-Effort-Strömen zur Verfügung gestellt werden können. Der zweite Fall setzt voraus, daß in den Zugangsknoten zu den Links mindestens zwei Warteschlangen implementiert werden: Eine für Pakete mit Realzeitanforderungen und eine zweite für Best-Effort-Pakete. Die Auswirkungen dieser beiden Ansätze auf die Paketverzögerung und die Auslastung der Links wird im folgenden untersucht.

8.4 Trennung von Realzeit- und Nicht-Realzeitströmen

Bei der strikten Trennung beider Anwendungsklassen in den beteiligten Übertragungseinrichtungen hängt sowohl die Performance für die Realzeitströme als auch die Performance für die Nicht-Realzeitströme lediglich von der für die jeweilige Klasse vorhandenen Konfiguration ab, da keine gemeinsamen Ressourcen genutzt werden.

In das Szenario aus Abbildung 8-7 wird die Anwendung *Filetransfer* integriert. Dazu wird in dem Block *P2PLink* eine zusätzliche Maschine (*MedMach2*) und die Prozeßinstanz *NrOutDev* deklariert. Diese beiden sind für die Verarbeitung der Datenpakete zuständig, die per Filetransfer übertragen werden sollen. Die Datenpakete werden von den Instanzen des Prozesses *FTP_Data* erzeugt, vgl. Abbildung 8-11. Das Lastverhalten der Anwendung FTP und somit das Verhalten der Prozesse *Load_Generator2* und *FTP_Data* entspricht dem Szenario aus Abschnitt 5.1.2.

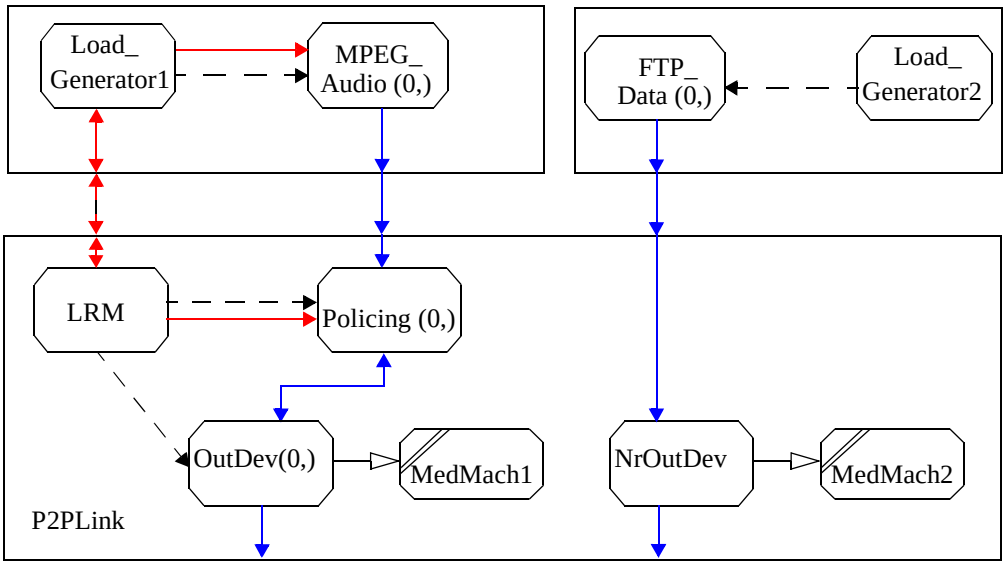
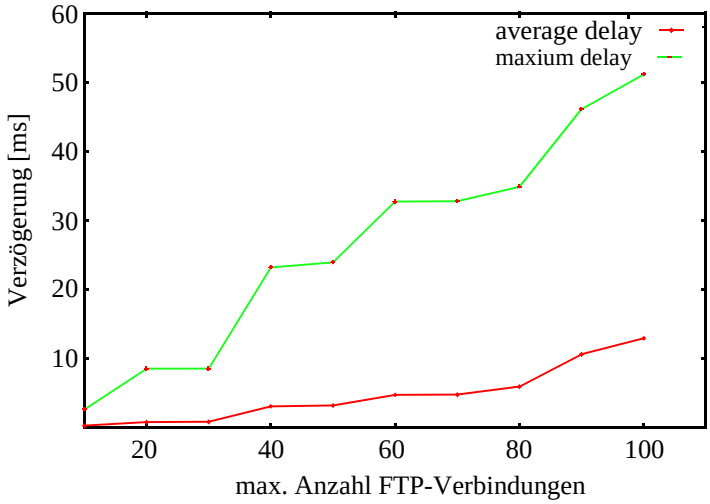


Abbildung 8-11 **Trennung von Realzeit- und Nicht-Realzeitströmen**

Der Prozeß *Load_Generator2* erzeugt eine nach oben beschränkte Anzahl paralleler Prozesse *FTP_Data*, die den Übertragungskanal mit dem für die Anwendung Filetransfer typischen Paketmuster belasten. In dieser Arbeit ist die Paketverzögerung auf einem T1-Link in Abhängigkeit von der Anzahl paralleler FTP-Verbindungen untersucht worden. Der Prozeß *NrOutDev*, der die FTP-Anwendungsdaten als Ganzes erhält, segmentiert diese Daten, so daß die maximal zulässige Paketgröße des Kanals nicht überschritten wird. Diese Aufgabe übernimmt normalerweise die Netzwerkschicht, z.B. IP. Aus Aufwandsgründen ist hier die Einführung eines Prozesses IP verzichtet worden, da dies keine Auswirkungen auf die Kanalauslastung und -verzögerung hat.



Verwendete Parameter:
Datenvolumen der Sitzung
$\log_2$-normalverteilt $\mu = 30.0$ $\sigma = 3.9$
Pausenlänge zwischen Strömen
$\log_2$-normalverteilt $\mu = 5.0$ $\sigma = 1.4$
Datenvolumen eines Stromes
$\log_2$-normalverteilt $\mu = 11.5$ $\sigma = 4.0$
Datenvolumen eines Bursts
Pareto-verteilt $k = 130000$ $\alpha = 1.3$
Pausen zwischen den Bursts
$\lambda_1 = 0.3$ $\lambda_2 = 1.2$ $\lambda_3 = 1.5$
$\lambda_4 = 3.0$ $\lambda_5 = 4.0$
Kanalbandbreite $\mu = 1.536$ MBit/s

Abbildung 8-12 **Paketverzögerung von FTP-Verkehr über T1-Link**

Betrachtet man die Paketverzögerung, siehe Abbildung 8-12, so stellt man fest, daß mit zunehmender Anzahl paralleler Verbindung die mittlere und die maximale Verzögerung ansteigt. Der Anstieg der maximalen Verzögerung ist allerdings deutlich steiler, so daß der Abstand zwischen beiden Werten stetig zunimmt. Während die mittlere Verzögerung über den gesamten Betrachtungsraum hinweg von einer Millisekunde auf etwa 13 Millisekunden steigt, erhöht sich die maximale Verzögerung von vier Millisekunden auf etwa 51 Millisekunden. Die Auslastung des T1-Links steigt dabei etwa linear von 14% auf 70% an, siehe Abbildung 8-13.

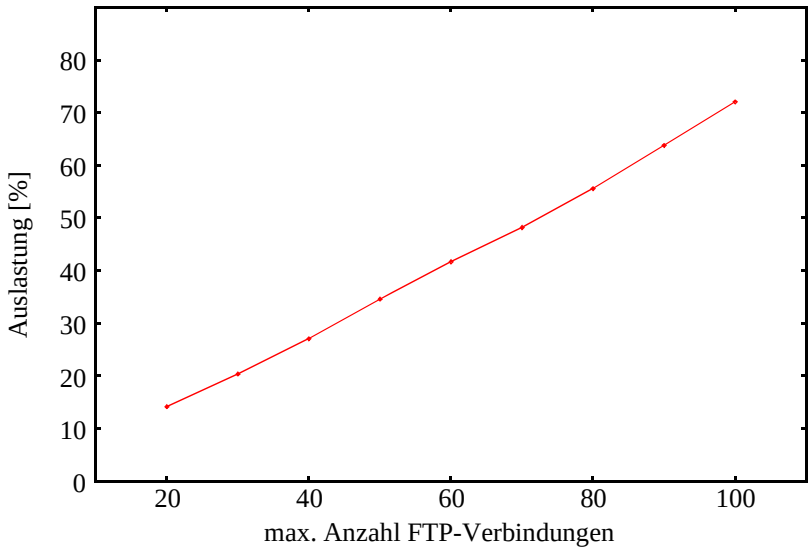


Abbildung 8-13 **Auslastung eines T1-Links durch FTP-Verkehr**

Nachdem die Übertragung sowohl für Best-Effort-Ströme als auch für Realzeitströme über eine Punkt-zu-Punkt-Verbindung separat untersucht worden ist, soll nun abschließend ermittelt werden, bei welchen Lastszenarien sich die gemeinsame Verarbeitung beider Anwendungen unter Verwendung von Prioritäten für Realzeitströme als vorteilhafter herausstellt.

8.5 Gemeinsame Ressourcennutzung durch Realzeit- und Nicht-Realzeitströme

Das Szenario gemeinsamer Ressourcennutzung unterscheidet sich lediglich durch die Nutzung der QSDL-Maschine im Block *P2P-Link*. Existierte bisher für jede Anwendung eine eigene Maschine, die den Zeitverbrauch der Paketübertragung modelliert hat, wird nun nur noch eine Maschine bereitgestellt, die beide Anwendungen bedient, siehe Abbildung 8-14. Diese Maschine bedient ebenfalls mit einer Geschwindigkeit, die der Bandbreite des Links entspricht. Damit die den Anwendungen in Summe zur Verfügung stehende Bandbreite mit der verfügbaren Bandbreite der Einzelszenarien übereinstimmt, wird die QSDL-Maschine *MedMach* mit $n = 4$ Bedienern ausgestattet.

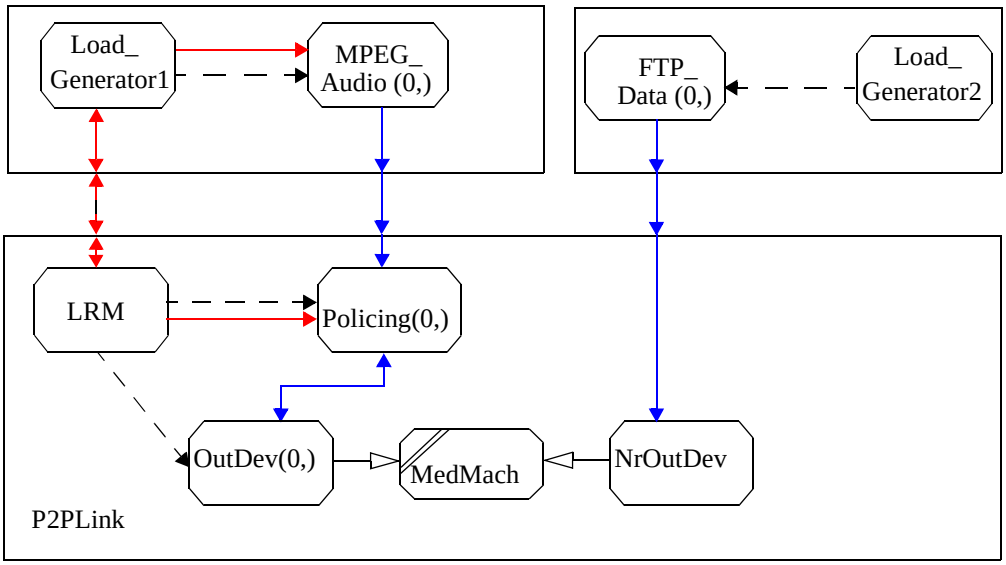


Abbildung 8-14 **Mischung von Realzeit- und Nicht-Realzeitströmen**

Da die MPEG-Audioströme strengere Anforderungen als die FTP-Ströme bzgl. Verzögerung und Durchsatz haben, werden diese priorisiert. Daher ändert sich die Scheduling-Disziplin der QSDL-Maschine *MedMach* von *First Come First Serve* (FCFS) zu *First Come First Serve Prior Non Preemptive* (FCFSPNP), vgl. Abbildung 8-15. Dieser Maschinentyp arbeitet die wartenden *Requests* nach deren Prioritäten ab, allerdings werden in Bedienung befindliche *Requests* bei Ankunft höher priorer *Requests* nicht unterbrochen. Dieses Modell entspricht einer Implementierung separater Warteschlangen und spiegelt so das zeitliche Verhalten dieses Szenarios wieder.

Damit die QSDL-Maschine vom Typ *FCFSPNP* das gewünschte Szenario korrekt nachbilden kann, müssen die von den Prozessen *OutDev* und *NrOutDev* ausgeführten Requests mit Prioritäten versehen werden, wobei den Requests des Prozesses *OutDev* eine höhere Priorität zugeordnet wird. Dazu erhält der Maschinendienst *transmit* einen zusätzlichen Parameter vom Typ *Natural*. Der aktuelle Wert dieses Parameters wird bei Aufruf des Dienstes durch den Prozeß *NrOutDev* auf den Wert *Null* gesetzt, vgl. Abbildung 8-15.

```
/*## MACHINESERVICE transmit(Natural,Natural); ##*/
/*## MACHINE MedMach;
      SERVER  nrserv;
      DISCIPLINE FCFSPNP;
      OFFERS transmit : bandwidth;
      SENSOR util UTILIZATION;
ENDMACHINE MedMach; ##*/

PROCESS NrOutDev;
        STATE ready;
        INPUT Frame(para1, para2);
        TASK tosent := para2;
 lab1:  DECISION tosent <= mtusize;
                (TRUE) :  /*## REQUEST transmit(tosent,0); ##*/
                          OUTPUT Frame(para1, tosent) VIA s6  /*## delay sigdelay ##*/;
                          NEXTSTATE -;
                (FALSE):  /*## REQUEST transmit(mtusize,0); ##*/
                          OUTPUT Frame(para1, mtusize) VIA s6  /*## delay sigdelay ##*/;
                          TASK tosent := tosent - mtusize;
                          JOIN lab1;
        ENDDECISION;
        ENDSTATE;
ENDPROCESS NrOutDev;
```

Serviceamount

Priorität

Abbildung 8-15 **Deklaration Maschinendienst und Maschine und Prozeß *NrOutDev* (Ausschnitt)**

In mehreren Experimentserien sind die Auswirkungen der Zusammenlegung beider Anwendungen auf die Verweilzeit von MPEG-Frames und FTP-Paketen untersucht worden. Dabei zeigt sich, daß die in Abschnitt 8.3 definierten Ressource-Manager auch in diesem Szenario den isochronen MPEG-Strömen die notwendigen Verarbeitungskapazitäten garantieren können. So sinkt sowohl die mittlere als auch die maximale Verzögerung für MPEG-Frames, vgl. Abbildung 8-16.

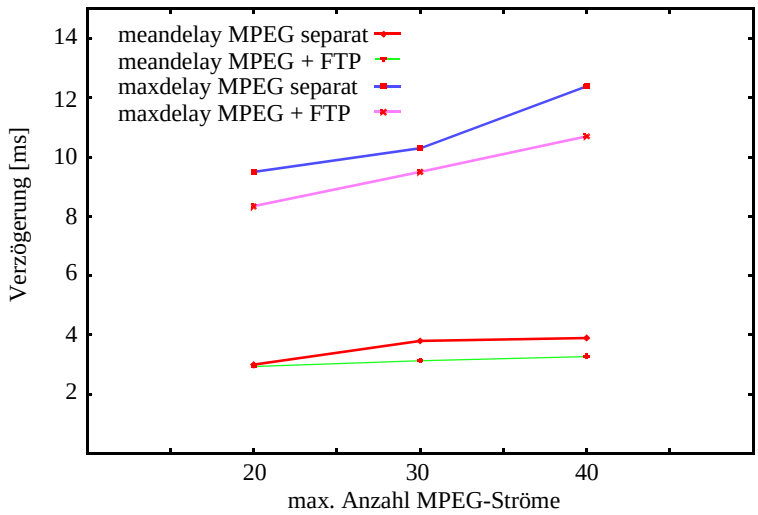


Abbildung 8-16 **Verzögerung MPEG-Pakete**

Hier macht sich der zusätzliche Übertragungskanal bemerkbar, der durch die MPEG-Ströme mitbenutzt werden kann. Die Belastung durch die FTP-Pakete fällt dabei nicht ins Gewicht, da MPEG-Frames mit höherer Priorität behandelt werden. Daher sind die Ergebnisse auch unabhän-

gig von der Anzahl der parallel existierenden FTP-Verbindungen. In Abbildung 8-16 sind die Ergebnisse für 100 parallele FTP-Verbindungen zusammen mit den Ergebnissen des separaten Szenarios dargestellt. So sinkt die mittlere Verzögerung bei max. 20 zulässigen MPEG-Strömen von drei Millisekunden auf 2.9 Millisekunden, während bei 40 zulässigen MPEG-Strömen die Verweilzeit von 3.9 Millisekunden auf 3.3 Millisekunden sinkt. Noch deutlicher macht sich der Vorteil der Zusammenlegung bei der Betrachtung der maximalen Verzögerung bemerkbar. Während diese bei 20 MPEG-Strömen von 9.5 Millisekunden auf 8.3 Millisekunden absinkt, reduziert sich die Verzögerungszeit bei 40 Strömen von 12.4 Millisekunden auf 10.7 Millisekunden.

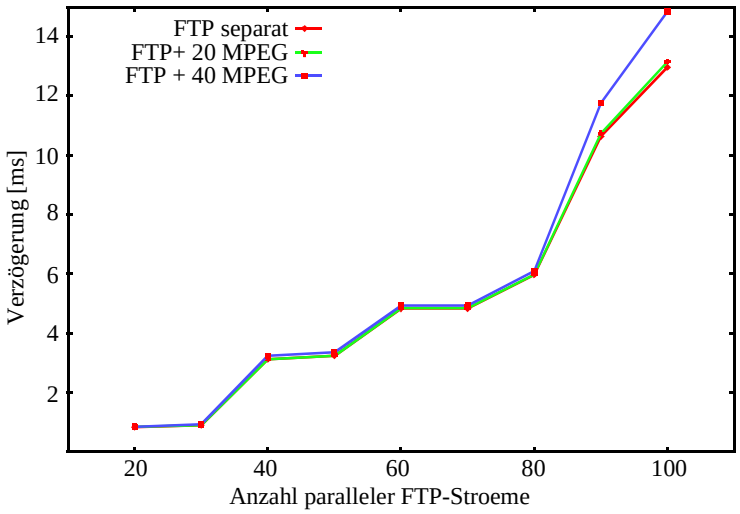


Abbildung 8-17 **Mittlere Verzögerung FTP-Pakete**

Die Bevorzugung der isochronen MPEG-Ströme führt allerdings nicht zu einer deutlichen Verschlechterung der Performance-Ergebnisse für FTP-Ströme. Erst bei einer Anzahl von 80 parallelen FTP-Verbindungen verschlechtert sich die Verzögerung für FTP-Pakete. Die mittlere und auch die maximale Verzögerung erhöhen sich allerdings erst signifikant, wenn parallel dazu etwa 40 MPEG-Ströme parallel aktiv sind, siehe Abbildung 8-17 und Abbildung 8-18.

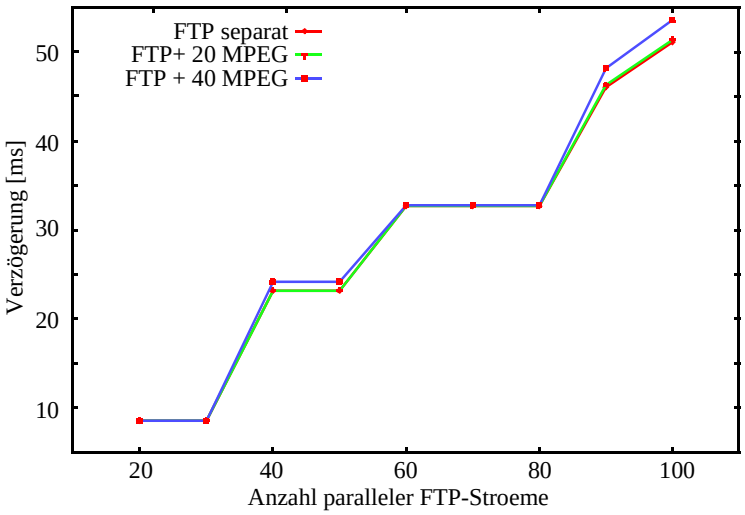


Abbildung 8-18 **Maximale Verzögerung FTP-Pakete**

Bezieht man die zu beobachtende Auslastung des Links in den verschiedenen Szenarien mit in die Bewertung ein, so erkennt man den Grund für die guten Ergebnisse. Die Zusammenlegung der Übertragungskapazitäten führt zu einer effizienteren Nutzung der Ressourcen, siehe Abbildung 8-19.

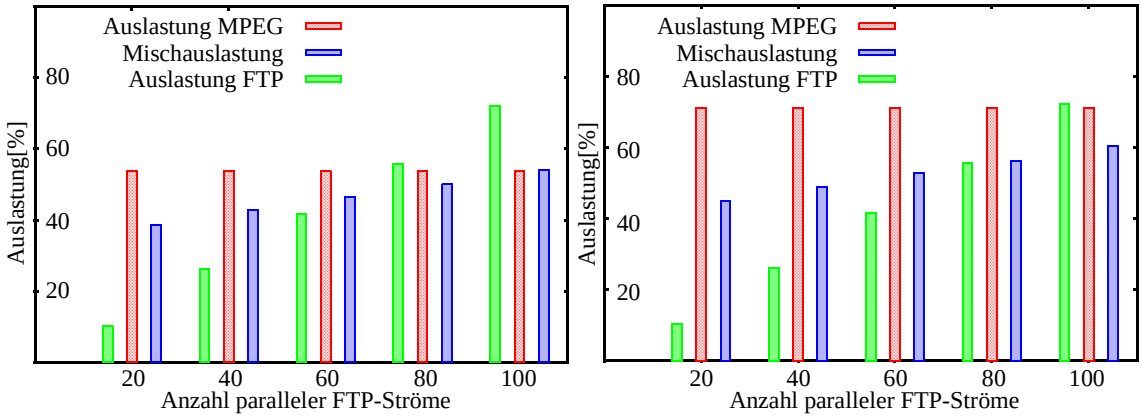


Abbildung 8-19 **Auslastung des Links durch beide Anwendungen: 20 MPEG-Ströme (li) bzw. 40 MPEG-Ströme (re)**

So liegt vor allem im Bereich jenseits von 80 FTP-Verbindungen die Auslastung des Links bei gemeinsamer Nutzung der Bandbreite durch beide Anwendungen unterhalb der Auslastung, die in den separaten Szenarien erreicht wird. Dieser Trend verstärkt sich bei Erhöhung der MPEG-Ströme.

Insgesamt kann man feststellen, daß eine gemeinsame Nutzung der vorhandenen Bandbreite durch Realzeit- und Nicht-Realzeitanwendungen der Trennung dieser Anwendungen vorzuziehen ist. Dabei zeigt sich, daß bei Priorisierung der Realzeitströme deren Anforderungen an die Dienstgüte erfüllt werden können, ohne die Dienstgüten für Best-Effort-Ströme zu stark abzusenken. Die für Realzeitströme entwickelten Ressource-Manager können unverändert übernommen werden.

8.6 Ressource-Manager für aktive Knoten

Neben der Bandbreite auf den Verbindungswegen gilt es, die vorhandenen Verarbeitungskapazitäten auf den aktiven Knoten (Hosts, Router, etc.) zu kontrollieren. Hier sind es im wesentlichen die Prozessoren, deren Zuteilung und Hauptspeicherzugriffe, die einen großen Einfluß auf die Leistungsfähigkeit haben. Aber auch I/O-Systeme wie Platten und interne Busse stellen potentielle Leistungsengpässe dar. Allerdings soll an dieser Stelle festgehalten werden, daß jeglicher Code zur Unterstützung von Realzeitströmen im Hauptspeicher gehalten werden sollte, um die Zugriffe auf Sekundär- oder gar Tertiärspeicher zu vermeiden.

Daher wird in den folgenden Abschnitten das Augenmerk auf lokales Ressource-Management für Prozessoren und Hauptspeicher gelegt. Da die entwickelten Ressource-Manager in der Anwendungsstudie zur Unterstützung des Stream-Protokolls *ST2*$^{+}$ eingesetzt werden sollen, wird dessen Implementierungsarchitektur bereits an dieser Stelle berücksichtigt. Der Kontroll- und der Datenpfad sind zu trennen, da die Datenpakete strenge Realzeitanforderungen besitzen, während die Kontrollpakete keinerlei Verarbeitungsschranken haben. Darüberhinaus können auf

jedem Knoten eine Reihe von parallelen *ST2*$^+$-Verbindungen[d] (*streams*) mit unterschiedlicher Priorität existieren. Da ein *ST2*$^+$-Agent bezüglich einer Verbindung eine Menge unterschiedlicher Zustände einnehmen kann, bzw. eine große Menge verbindungsspezifischer Informationen verwaltet, liegt es nahe, für jeden ST2$^+$-Strom einen SDL-Prozeß zu spezifizieren, der durch den Kontrollprozeß *SCMP* dynamisch erzeugt wird, wenn eine neue Verbindung angefordert wird, vgl. Abbildung 8-20.

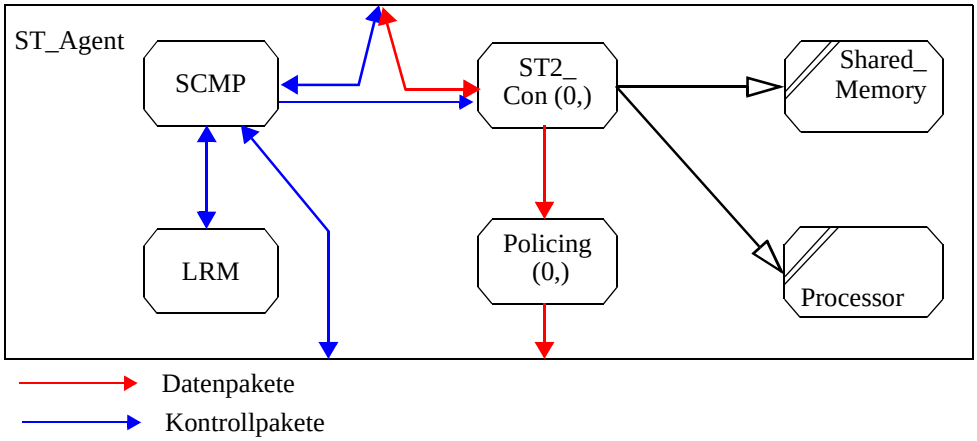


Abbildung 8-20 **Struktur eines ST-Agenten**

Darüberhinaus existiert ein lokaler Ressource-Manager, der die Kapazitäten der vorhandenen aktiven Ressourcen verwaltet. Für jede etablierte Verbindung existiert eine Policing-Einheit, die die Einhaltung der ausgehandelten Verkehrscharakteristik überwacht. Die Policing-Funktion wird jedoch nur im ersten Knoten (*Origin*) eines Stromes benötigt, um die Quelle in ihrem Verhalten zu kontrollieren.

Zur Unterstützung der ST2$^+$-Ströme sind lediglich die Ausführungszeiten der Prozesse *ST2_Con* zu kontrollieren. Sie erhalten gegenüber dem Kontrollprozeß *SCMP* eine höhere Priorität. Alle Prozesse *ST2_Con* laufen quasi-parallel als Threads und erhalten die zur Bearbeitung eines Paketes benötigten Informationen aus einer Datenbasis, die im *Shared-Memory*-Bereich implementiert ist. Der Zugriff auf die Daten wird durch die QSDL-Maschine *Shared_Memory* modelliert, die als FCFS-Maschine mit einem Bediener spezifiziert ist. Somit ist gewährleistet, daß immer nur ein Prozeß zur Zeit Zugriff auf den Bereich hat und es so nicht zu Inkonsistenzen kommt. Alle anderen zeitverbrauchenden Aktivitäten der Instanz werden über Dienstaufrufe an die Maschine *Processor* modelliert. Diese arbeitet mit der Bedienstrategie *PS*, so daß die nebenläufige Ausführung aller aktiven Ströme gewährleistet ist.

8.6.1 Verkehrscharakteristik und Dienstgüte

Die in Abschnitt 8.2.1 benutzte *FlowSpec* wird auch dem lokalen Ressource-Manager in den aktiven Zwischenknoten präsentiert. Konnten jedoch die Angaben in der *FlowSpec* zur Berechnung der oberen Verzögerungsschranken für Kanäle direkt verwendet werden, so bedarf es bei der Bestimmung dieser Größen in den aktiven Knoten einigen Aufwands. Die Angaben der

d. Siehe Abschnitt 7.4.

FlowSpec enthalten die Angaben zur gewünschten Senderate sowie zur Paketgröße und die oberen Schranken für die Ende-zu-Ende-Verzögerung von Paketen, vgl. Abbildung 8-2. Der Kehrwert der gewünschten Senderate gibt die obere Schranke für die Verweilzeit eines Pakets auf dem Knoten vor. Damit diese berechnet werden kann, sind jedoch Angaben erforderlich, welche Aktionen zur Verarbeitung eines *ST2*$^+$-Pakets auf dem Knoten erforderlich sind. Falls eine *ST2*$^+$-Verbindung vollständig aufgebaut ist, kann der Prozeß *ST2_Con* auf Anforderung Daten übertragen. Eine solche Verbindung ist für einen Agenten vollständig aufgebaut, wenn alle Nachfolgeknoten in *Downstream*-Richtung die Verbindung akzeptiert haben.

Bei Empfang des Signals *DataReq* im Zustand *established* muß durch den Prozeß *ST2_Con* die Protokolldateneinheit (PDU) zusammengesetzt und an die unterliegende Instanz weitergereicht werden, vgl. Abbildung 8-21.

```
PROCESS ST2_Con;
   DCL datin MPEG_Frame,
         streamid  Sid,
         pdu DataPdu;

   START;
   NEXTSTATE established;
   STATE established;
      INPUT DataReq(datin);
      TASK pdu!TB := datin!len +12,
            pdu!HCS := datin!len * 2,
            pdu!UID := streamid!UID,
            pdu!OID := streamid!OID,
            pdu!Data := datin;
      OUTPUT Req(pdu);
   NEXTSTATE established;
ENDPROCESS;
```

```
/*## Machineservice getdata(Real), pushdata(Real),
      computepdu(Real), getinfo(Real); ##*/

/*## Machine Processor;
      Server   no_server;
      Discipline  PS;
      Offers getdata : getdataspeed,
             computepdu : computespeed,
             pushdata : pushdataspeed;
Endmachine Processor; ##*/

/*## Machine Shared_Memory;
      Server   1;
      Discipline  FCFS;
      Offers getinfo : getinfospeed,
Endmachine Shared_Memory; ##*/
```

Abbildung 8-21 **Prozeß ST2_Con (Ausschnitt), Maschinen und Maschinendienste**

Dazu müssen die Daten von der überliegenden Instanz übernommen, die Parameter überprüft und der PDU-Header erstellt werden. Bevor die PDU an die unterliegende Instanz weitergereicht werden kann, muß zunächst in der Datenbasis des Agenten nachgesehen werden, welche Einträge bezüglich der Nachfolgeknoten (Anzahl und Adressen) vorhanden sind. Die notwendigen Aktionen müssen dem lokalen Ressource-Manager mit ihren Dienstanforderungen bekannt sein. So sind die Anforderungen bei der Datenübernahme bzw. -übergabe abhängig von Länge der Daten. Diese wird dem Manager beim Verbindungsaufbau durch die ihm übergebene *FlowSpec* mitgeteilt. Daher ist es ausreichend, dem Ressource-Manager die Geschwindigkeit und die Anzahl der Bediener (Prozessoren) der von ihm verwalteten Maschine mitzuteilen.

Die Analyse der übergebenen Dienstprimitive und die Erstellung der Header ist unabhängig von der Länge der übergebenen Daten [33]. Daher wird für jedes Datenpaket ein konstanter Bedienwunsch angenommen. Der konkrete Bedienwunsch kann durch Messungen an einem Prototypen ermittelt werden. Auch dieser Wert muß dem Ressource-Manager zusammen mit der zugehörigen Ausführungsgeschwindigkeit bei dessen Initialisierung mitgeteilt werden. Gleiches gilt für den Aufwand, der je Datenpaket anfällt, um die notwendigen Informationen aus der Datenbasis auszulesen, vgl. Abschnitt 8.6.2. Mit diesen Informationen ist der Ressource-Manager in der Lage, die benötigten Ressourcen für jede Verbindung zu berechnen und für diese zu reservieren. Gleichzeitig ist es ihm möglich, die Auswirkungen auf die bisher existierenden Verbindungen zu ermitteln und somit über Annahme bzw. Ablehnung einer neuen Verbindung zu entscheiden.

Für die QSDL-Maschine *Shared_Memory* können die Ergebnisse aus Abschnitt 8.2 übernommen werden, da es sich bei ihr ebenfalls um eine FCFS-Maschine handelt. Falls mit $speed_{fcfs}$ die Bediengeschwindigkeit und mit $amount_{fcfs}$ der Bedienwunsch bezeichnet wird, so ergibt sich die Bedienzeit $t_{service}$ zu:

$$t_{service} = \frac{amount_{fcfs}}{speed_{fcfs}} \quad \text{(Gl. 6)}$$

Beschränkt man die Anzahl der parallelen Ströme, so ergibt sich die Obergrenze der Verweilzeit zu:

$$VZ_{max} = \frac{amount_{fcfs}}{speed_{fcfs}} \cdot N \quad \text{(Gl. 7)}$$

Für die QSDL-Maschine *Processor*, die als Scheduling-Disziplin *PS* verwendet, stellt sich die Situation anders dar, da es keine Wartezeiten gibt. Falls die Anzahl der vorhandenen Bediener größer oder gleich der Anzahl gleichzeitig aktiver Tasks ist, so ergibt sich die Bedienzeit und somit auch die Verweilzeit analog zur FCFS-Maschine gemäß (Gl. 8):

$$t_{service} = \frac{amount_{ps}}{speed_{ps}} \quad \text{(Gl. 8)}$$

Sind mehr aktive Tasks als Bediener vorhanden, so wird die Gesamtkapazität der Maschine ($k \cdot speed_{ps}$) auf die vorhandenen Tasks aufgeteilt. Bezeichnet k die Anzahl der Bediener und N die maximale Anzahl der gleichzeitig aktiven Tasks, so ergibt sich als obere Schranke der Verweilzeit:

$$VZ_{max} = \frac{amount_{ps}}{k \cdot speed_{ps}} \cdot N \quad \text{(Gl. 9)}$$

Die obere Schranke für die Gesamtverzögerung eines Datenpaketes durch den Knoten ergibt sich somit als Summe der Verweilzeiten nach (Gl. 7) und (Gl. 9). Dieser Wert wird vom Prozeß *LRM* als Wert der Variablen *locmaxdelay* gespeichert. Bei jedem neuen Verbindungswunsch wird durch den Ressource-Manager geprüft, ob die zu erwartende Verzögerung geringer ist, als der Abstand zweier aufeinanderfolgender Datenpakete, siehe Abbildung 8-22. Falls dies nicht der Fall ist, wird dieser Test mit der abgeschwächten Senderate, die in der *FlowSpec* als Wert von *LimitRate* angegeben ist, wiederholt. Kann auch diese Schranke nicht eingehalten werden, muß die neue Verbindung abgewiesen werden. Neben der Verzögerungsschranke kann durch den Manager ebenfalls die maximale Auslastung der Maschinen kontrolliert werden. Dabei ist jedoch zu berücksichtigen, daß die Werte für *Neulast*, *Altlast* und *Maxlast* für die beiden Maschinen *Processor* und *Shared_Memory getrennt* zu betrachten sind, siehe Abbildung 8-22.

Durch die Verwaltung der vorhandenen Ressourcen und der Kontrolle der aktiven Ströme können den zugelassenen Strömen obere Schranken für die Verzögerung zugesagt werden. Im folgenden werden nun konkrete Verzögerungsschranken für die Unterstützung von MPEG-Audioströmen bestimmt und die Obergrenzen für die Zulassung paralleler Ströme in Abhängigkeit der vorhandenen Ressourcen ermittelt.

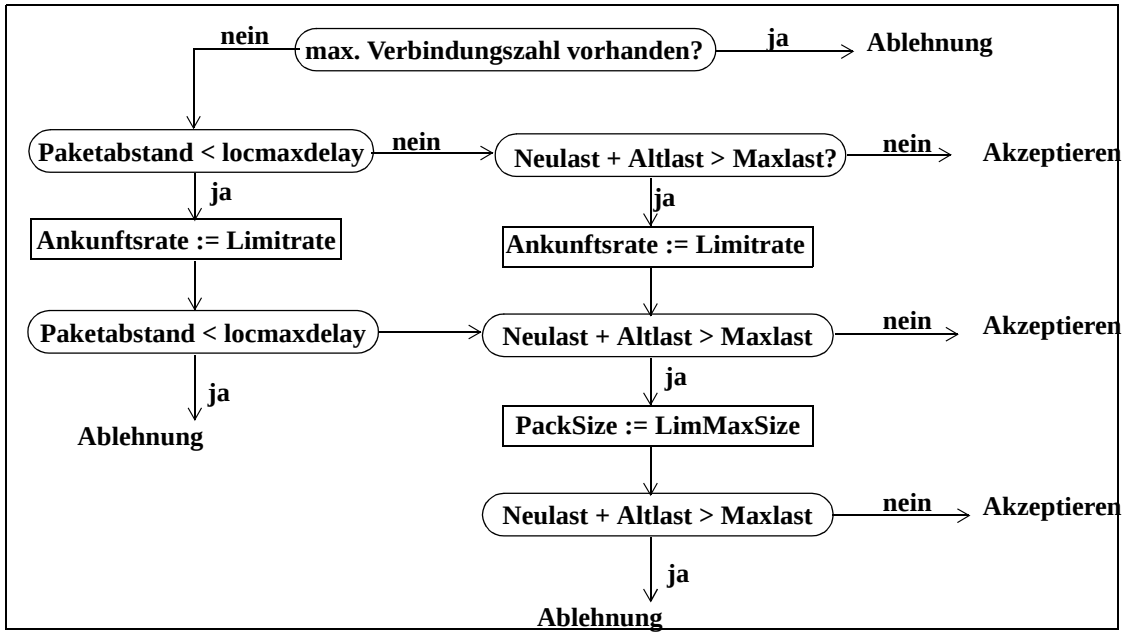


Abbildung 8-22 **Testschema zur Knotenzugangskontrolle**

8.6.2 Experimente und Ergebnisse

Die konkreten Maschinenparameter, aber auch die Art und der Umfang der zeitverbrauchenden Dienstaufrufe orientieren sich an einer Implementierung der Vorgängerversion des *ST2*$^+$-Protokolls auf einer IBM RISC Maschine System 6000 Modell 550 mit 42 MHz Taktfrequenz. Diese Implementierung ist in [33] vermessen worden. In dieser meßbasierten Performance-Studie sind Durchsatz und Verweilzeiten von Datenpaketen in Abhängigkeit der verwendeten Paketgrößen bestimmt worden. Die Meßwerte sind bei einer aktiven ST-II-Verbindung bei gleichzeitiger Priorisierung der ST-II-Verbindung gegenüber anderen Anwendungen erhoben worden, so daß die gemessen Verweilzeiten als Bedienzeiten aufgefaßt werden können, da für ST-II-Datenpakete keine Wartezeiten wegen der Priorisierung aufgetreten sind. Die Vermessung ist ohne Berücksichtigung der Datenübernahme am oberen Dienstzugangspunkt, aber unter Aufruf der Sende-Operation des unterliegenden Mediums durchgeführt worden (Es handelte sich um eine Implementierung mit einer *DownCall*-Schnittstelle.).

Bei dieser Messung wurde eine von der Paketgröße unabhängige Verweilzeit von 98 Millisekunden für die Analyse der übergebenen Parameter, die Bildung eines allgemeinen und eines speziellen *Next-Hop-Headers* sowie für den allgemeinen Overhead gemessen. Die Bildung des speziellen *Next-Hop-Headers* ist in der *ST2*$^+$-Version nicht mehr nötig, da er durch einen für die gesamte Verbindung gültigen *Connection-Header* ersetzt wurde. Die Verweilzeit reduziert sich dadurch auf einen Wert von 64 Millisekunden. Alle anderen Aktionen werden zu einem gemeinsamen QSDL-Request zusammengefaßt. Der zugehörige Maschinendienst *computepdu* wird von der QSDL-Maschine *Processor* angeboten, siehe Abbildung 8-21 rechts. Von dieser Maschine werden ebenfalls die Maschinendienste *getdata* und *pushdata* angeboten, die für die Übernahme und Weitergabe der Daten aufgerufen werden, siehe Abbildung 8-21 und Abbildung 8-23. Die Bedienwünsche dieser beiden Dienste hängen jedoch von der Datenlänge der übernommenen,

bzw. der übergebenen Daten ab. Diese Informationen sind in den Paket-Headern gespeichert (*indat!len*, bzw. *pdu!TB*).

Für den Zugriff auf die Datenbasis sind 21 Millisekunden gemessen worden. Der Zeitbedarf für den Zugriff auf die Datenbasis wird durch den Request *getinfo* modelliert. Der Bedienwunsch ist wie im Falle des Maschinendienstes *computepdu* unabhängig von der Datenlänge und daher als konstant angenommen, siehe Abbildung 8-23.

```
PROCESS ST2_Con;
 DCL datin MPEG_Frame,
          streamid  Sid,
          pdu DataPdu;
 START;
 NEXTSTATE established;
 STATE established;
 INPUT DataReq(datin);
   /*## Request getdata(float(datin!len)) FROM processor; ##*/
   /*## Request computepdu(1.0) FROM Processor; ##*/
   TASK pdu!TB := datin!len +12,
            pdu!HCS := datin!len * 2,
   /*## Request getinfo(1.0) FROM SharedMemory; ##*/
   TASK pdu!UID := streamid!UID,
            pdu!OID := streamid!OID,
            pdu!Data := datin;
   /*## Request pushdata(float(pdu!TB) FROM Processor; ##*/
   OUTPUT Req(pdu);
   NEXTSTATE established;
ENDPROCESS;
```

Abbildung 8-23 **QSDL-Prozeß ST2_Con**

Aus den Meßwerten der Implementierung lassen sich die Maschinengeschwindigkeiten bestimmen. So ergibt sich aus der Verweilzeit (respektive Bedienzeit) für den Zugriff auf die Datenbasis und der Kenntnis, daß dieser Zugriff für jedes Datenpaket genau einmal vorgenommen wird, eine Maschinengeschwindigkeit entsprechend (Gl. 10).

$$speed = \frac{1 Zugriff}{vz_{mess}} = \frac{1}{21ms} = \frac{1000}{21s} = 47,6 \text{ Zugriffe/s} \qquad \text{(Gl. 10)}$$

Analog dazu ergibt sich für die Bediengeschwindigkeit der Maschine *Processor* für den Dienst *computepdu* ein Wert von 15.38 Tasks/s. Da die Werte für die Übergabe von Anwendungsdaten nicht erhoben worden sind, sind für diese die Maschinengeschwindigkeiten verwendet worden, wie sie bereits in einer Performance-Studie des TCP/IP- Protokolls in [60] verwendet worden sind. Dabei beträgt die Geschwindigkeit zur Übernahme der Daten $2,68 \cdot 10^8$ Byte/s und zur Übergabe der Daten $4,19 \cdot 10^8$ Byte/s. Die in [33] gemessenen Verweilzeiten sind allerdings viel zu lang, um z.B. MPEG-Audioströme zu unterstützen, die mit einer Rate von 39 Frames/s Anwendungsdaten erzeugen. Da die gemessenen Verweilzeiten im wesentlichen unabhängig von der verwendeten Paketgröße sind, kann nur festgehalten werden, daß mit einer Maschine, die in ihrer Leistungsfähigkeit nicht deutlich über dem Vermögen der IBM 6000/550 liegt, eine Unterstützung von Realzeitströmen mit der oben genannten Senderate nicht möglich ist.

In dieser Arbeit sind daher Ergebnisse von Benchmarks herangezogen worden, um realistische Werte für die Leistungsfähigkeit von Maschinen zu bekommen. So ergibt sich für die ältere IBM 6000 mit einer Taktrate von 42 MHz ein *SpecInt95*-Wert von 1.82, während eine neuere Siemens Primergy mit einer Taktrate von 500 MHz für diesen Benchmark einen Wert von 22.87 aufweist. Damit ist diese Maschine bzgl. des *SpecInt95*-Benchmarks um den Faktor 12.56 schneller als die IBM 6000 Maschine. In den hier vorgestellten Untersuchungen sind daher nicht die aus den Messungen ableitbaren Geschwindigkeiten angenommen worden, sondern Geschwindigkeiten, die um einen Faktor 13 höher liegen. Daraus resultiert eine Maschinengeschwindigkeit von 200 Tasks/s für den Maschinendienst *computepdu* und eine Geschwindigkeit von 620 Tasks/s für den Maschinendienst *getinfo*, vgl. Abbildung 8-24.

GETINFOSPEED	620.0	VARDUR	1.0
PUSHDATASPEED	419000000.0	MEANDUR	150.0
COMPUTESPEED	200.0	LAMBDA	5.0
GETDATASPEED	268000000.0	OPT	0.333
MAXUTILPROC	0.80	NOSERVERS	3
MAXDATASIZE	520	NOSTREAMS	12
MAXUTILMEM	0.80		

Abbildung 8-24 **Quantitative Parameter für Last und Maschinen**

In einer ersten Studie ist das QSDL-System mit genau einem MPEG-Strom von 39 Frames je Sekunde mit einer Paketgröße von 520 Byte belastet worden. Die im Modell beobachtbare Verweilzeit eines MPEG-Frames ergab sowohl für den Mittelwert als auch für den Maximalwert 6.61 ms. Dieser Wert stimmt exakt mit den Meßwerten überein, wenn man diese entsprechend durch den Faktor 13 dividiert, um den die Maschinen schneller gemacht worden sind. Dieses Szenario stellt den für den unterstützten Strom bestmöglichen Fall dar, bei dem davon ausgegangen wird, daß ein maximaler Parallelitätsgrad vorhanden ist. Um die Auswirkungen parallel existierender Verbindungen auf die Performance zu untersuchen, ist das QSDL-System in einer weiteren Experimentserie mit mehreren MPEG-Strömen belastet worden. Dabei ist der lokale Ressource-Manager so eingestellt worden, daß er beginnend mit zwei parallelen Strömen bis zu zehn parallele Ströme zuläßt, sofern die oberen Grenzen für die maximal zulässige Auslastung (80%) und die Verzögerung (25.6 ms) nicht überschritten werden, siehe Abbildung 8-25.

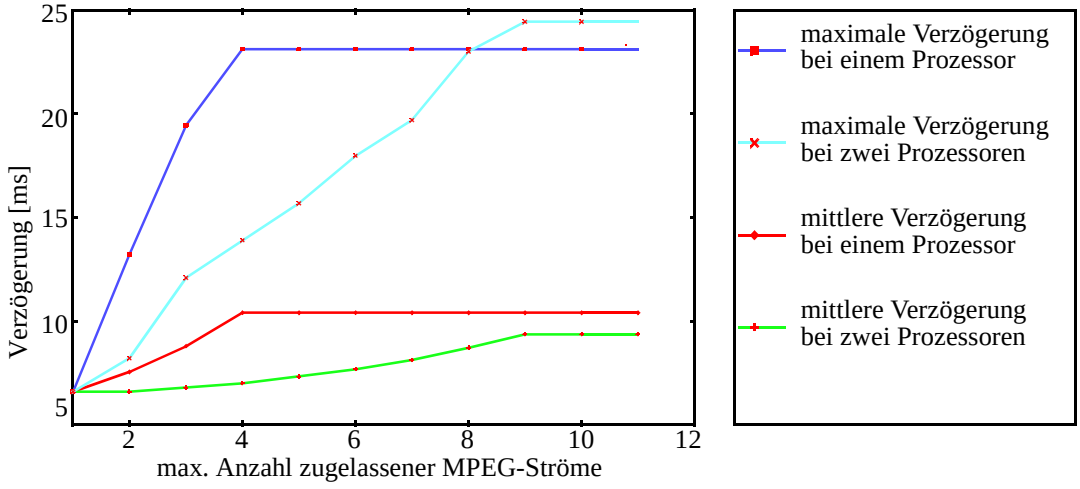


Abbildung 8-25 **Verzögerung von MPEG-Frames durch einen ST2$^+$-Knoten**

Falls für die Implementierung des *ST2*$^+$-Protokolls lediglich ein Prozessor zur Verfügung steht, so wird bereits bei vier aktiven Verbindungen die Kapazitätsgrenze erreicht. Der lokale Ressource-Manager läßt keine weiteren Verbindungen zu. Dabei stellt sich für die maximale Verzögerung von Datenpaketen ein Wert von ca. 23 Millisekunden ein, während die durchschnittliche Verzögerung der Pakete einen Wert von 10.4 Millisekunden erreicht. Dabei stellt sich die Maschine *Processor* als leistungslimitierender Faktor der untersuchten Konfiguration heraus, siehe Abbildung 8-26.

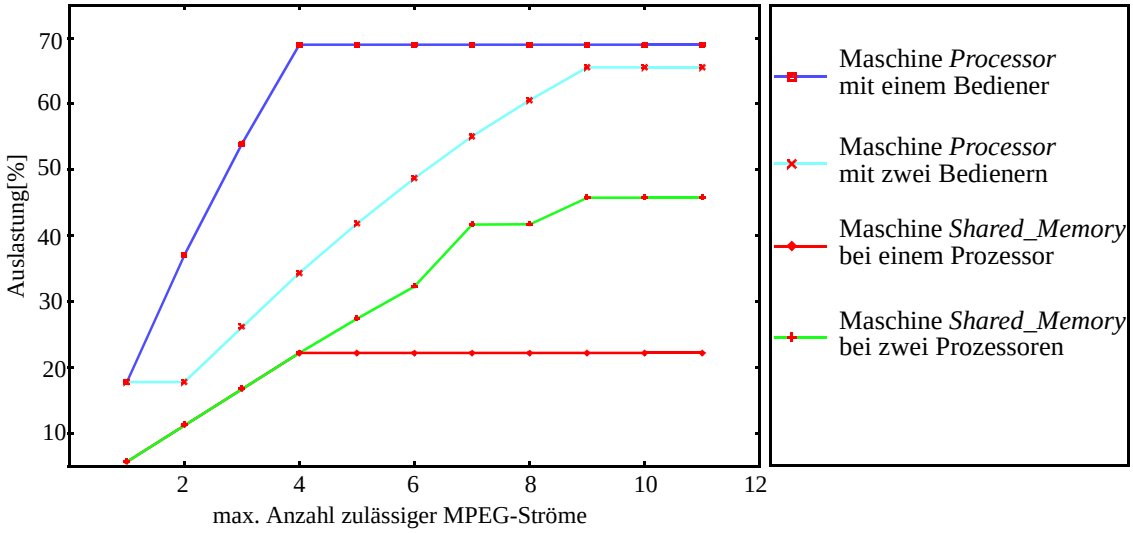


Abbildung 8-26 **Maschinenauslastungen eines ST2$^+$-Knotens**

Während die FCFS-Maschine *Shared_Memory* eine Auslastung von etwa 20% aufweist, ergibt sich für die Maschine *Processor* ein Wert von ca. 70%. Eine Erhöhung der Anzahl aktiver Ströme würde nicht nur zur Überschreitung der Auslastungsgrenze führen, sondern auch zur Überschreitung der maximalen Verzögerung von 25.6 Millisekunden.

Stattet man hingegen die Maschine Prozessor mit zwei Bedienern aus, so läßt sich die Anzahl paralleler Ströme auf neun erhöhen. Die Verweilzeiten erhöhen sich im Maximalwert auf 24.4 Millisekunden, während die mittlere Verzögerung auf etwa 9.3 Millisekunden absinkt. Durch die zusätzliche Bedienkapazität in der Maschine *Processor* sinkt die Auslastung dieser Maschine deutlich ab. Sie erreicht selbst bei neun aktiven Strömen nur einen Wert von etwa 65%, siehe Abbildung 8-26. Die Maschine *Shared_Memory* wird zwar durch die vermehrte Anzahl paralleler MPEG-Ströme deutlich höher ausgelastet, allerdings erreicht die Belastung mit Werten von etwa 45% keine kritischen Bereiche.

In beiden Fällen ist deutlich zu erkennen, daß der lokale Ressource-Manager die vorhandenen Ressourcen derart verwaltet, daß eine sie überlastende Anzahl von Strömen nicht zugelassen wird. Sowohl die Auslastungs- als auch die Verzögerungskurve zeigen einen idealen Verlauf, bei denen die Werte zunächst mit der Steigerung der Last linear ansteigen, während sie bei Erreichen der Kapazitätsgrenzen auf dem hohen Niveau konstant gehalten werden.

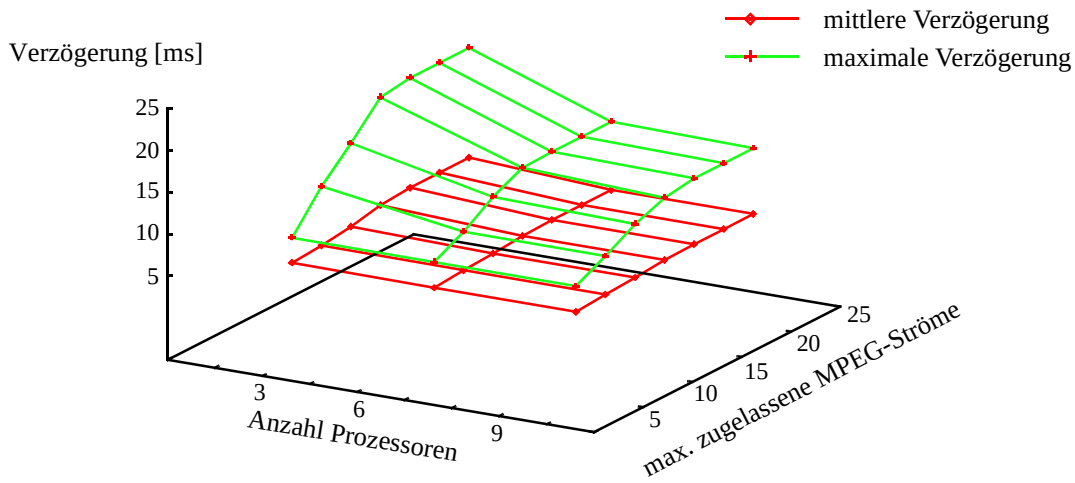


Abbildung 8-27 **Verzögerungsverhalten bei Mehrprozessorsystemen**

Erhöht man die Anzahl der Prozessoren weiter, so erkennt man eine weitere Erhöhung des Durchsatzes, die allerdings bei zwölf parallelen Strömen ihr absolutes Maximum erreicht. Eine weitere Erhöhung der Kapazität ist dann nur noch möglich, wenn auch der Zugriff auf die Datenbasis parallelisiert werden kann, da sich bei mehr als drei Prozessoren der Engpaß auf die Datenbasis verlagert. Dazu wären dann zwei Datenbasen notwendig.

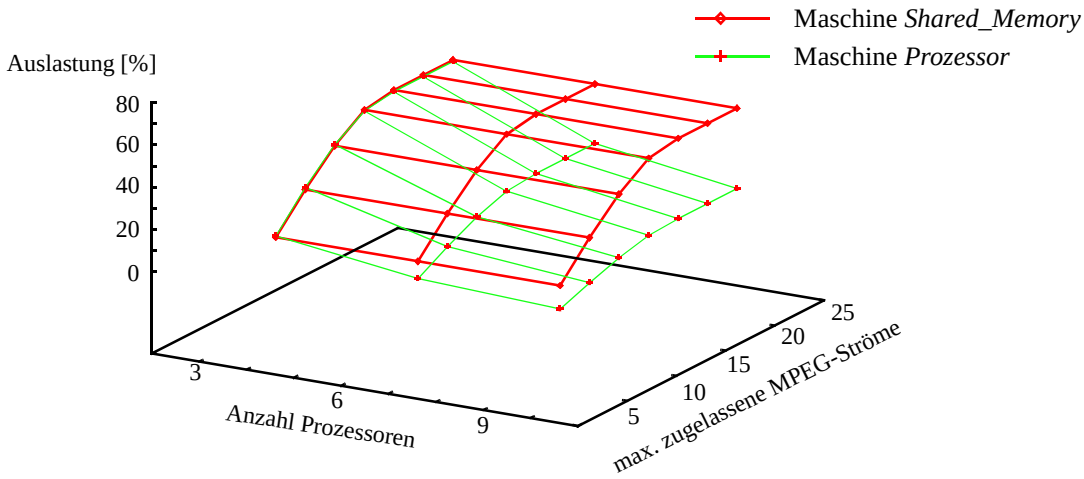


Abbildung 8-28 **Ressourcen-Auslastung bei Mehrprozessorsystemen**

8.7 Zusammenfassung Kapitel 8

Die beiden Frameworks der IETF, die zur Unterstützung von Multimedia-Diensten auf Reservierung von Ressourcen basieren, setzen entsprechende Reservierungs- und Verwaltungsmechanismen voraus. In diesem Kapitel sind Ressource-Manager entwickelt und vorgestellt worden, die durch eine geeignete Kontrolle der Auslastung sowie der lokalen Verzögerung durch Beschrän-

kung der parallel zugelassenen Verbindungen, die Verzögerung von Dateneinheiten sowohl durch Links als auch durch aktive Knoten wie Router und Endsysteme nach oben beschränken können. Die verwendeten Mechanismen schützen die durch sie verwalteten Ressourcen vor Überlastung und tragen so zu einer zeitgerechten Verarbeitung der Daten bei. Dabei haben die Untersuchungen in diesem Abschnitt gezeigt, daß es mit relativ geringem Aufwand möglich ist, die Ressourcen auf einem Knoten so zu verwalten, daß eine Unterstützung von Realzeitströmen möglich ist.

Dabei hat sich die QUEST-Methodik als sehr gut einsetzbar erwiesen, die Auswirkungen einer Implementierungsarchitektur auf das Leistungsverhalten des Knotens und so letztlich auf die Verzögerung und den Durchsatz von Multimedia-Strömen zu untersuchen.

Im nun folgenden letzten Kapitel wird gezeigt, wie die bisher vorgestellten Methoden und Bausteine eingesetzt werden können, die Leistungsfähigkeit eines vollständigen Transportsystems für Multimedia-Ströme entwurfsbegleitend zu untersuchen.

Kapitel 9

Analyse multimedialer Anwendungen über ST2$^+$

9.1 Einführung

Der Fokus wird in diesem Kapitel von der bisher dominierenden mikroskopischen Betrachtung lokaler Auswirkungen von Architektur- und Implementierungsentscheidungen auf die makroskopische Betrachtung der globalen Auswirkungen auf die unterstützten Anwendungen verlagert. Die Betrachtung großer Topologien liegt zwar nicht im Zentrum der QUEST-Methodik, allerdings können erforderliche makroskopische Untersuchungen mit QUEST durchgeführt werden.

In der folgenden Studie werden die Auswirkungen von parallel existierenden Audioströmen auf die Dienstgüte aus Anwendersicht für einen Verteildienst untersucht, der sich auf das *ST2*$^+$-Protokoll abstützt. Die vom Protokoll vorausgesetzten Routing-Funktionen sowie lokale Ressource-Manager, die die vorhandenen Ressourcen auf den aktiven Zwischenknoten und den Verbindungsleitungen verwalten und entsprechend zuteilen, werden dabei entwickelt und analysiert. Hierbei wird vor allem untersucht, inwieweit die entwickelten Funktionen geeignet sind, die für Realzeitanwendungen wichtigen anwenderorientierten Leistungsmaße wie **Ende-zu-Ende-Verzögerung**, **Verzögerungsschwankung** (Jitter) oder erzielbare **Durchsätze** zu garantieren. An dieser Stelle sei darauf hingewiesen, daß für die Untersuchung großer Netze kommerziell vermarktete Werkzeuge existieren [27], [84], [99]. Allerdings ist auch hier wieder der Hinweis angebracht, daß die den Werkzeugen zugrundeliegende Methodik die separate Erstellung und Pflege von Performance-Modellen erfordert. Sie verlieren dadurch den Vorteil der automatischen Konsistenz von Entwurfsbeschreibung und Performance-Modell und erschweren alleine durch den erhöhten Aufwand zur Modellerstellung die Integration von Performance-Untersuchungen in den Entwurfsprozeß.

9.2 Anwendungsstudie Audio-Verteildienst

Der betrachtete Verteildienst besteht aus einem Audio-Server und einer Reihe von räumlich verteilten Audio-Clients, die mit MPEG-Audiodaten vom Server versorgt werden. Die lokalen Netze, in der sich sowohl die Clients als auch der Server befinden, sind durch ein Router-Netz verbunden. Die aktiven Elemente sowie die Netz-Topologie sind in Abbildung 9-1 dargestellt.

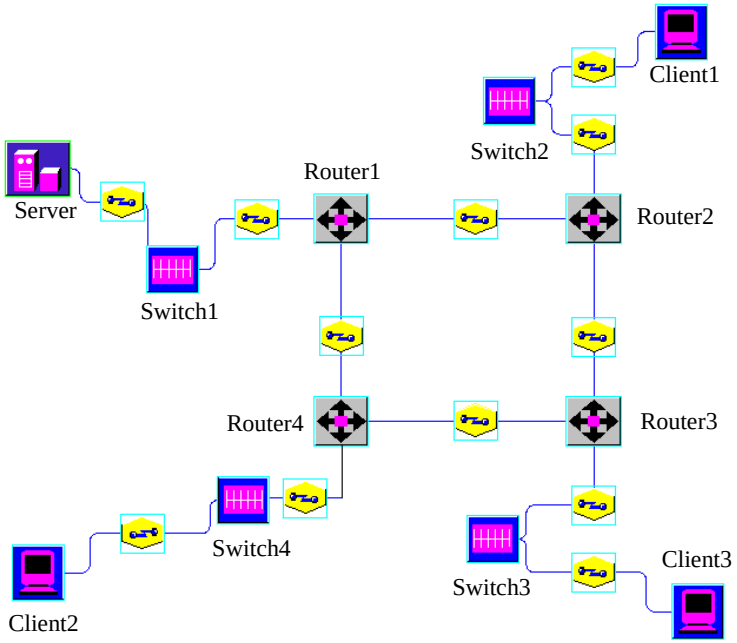


Abbildung 9-1 **Topologie des Anwendungsszenarios**

Das zentrale Router-Netz besteht aus vier Routern, die ringförmig durch Punkt-zu-Punkt-Leitungen verbunden sind. Die lokalen Netze sind Ethernets, die durch Verwendung von Switching-Technologien aufgebaut sind. Somit verhindert man Kollisionen zwischen zwei Endsystemen, die im Standard-Ethernet die Angabe von oberen Schranken für die Verzögerung unmöglich machen. Durch eine entsprechende Verschaltung in den *Switches* kann dafür gesorgt werden, daß sich die Verbindungen zwischen zwei Knoten wie eine direkte Punkt-zu-Punkt-Verbindung verhalten [109]. Bei ausreichender Backplane- und Speicherkapazität in den verwendeten Switches kann daher die Verzögerung durch einen „Link“ bestehend aus zwei Kabeln und einem Switch durch die Verwendung eines Einzel- oder Mehrfach-Links approximiert werden, wie sie in Abschnitt 8.2 und Abschnitt 8.3 beschrieben sind.

In den Routern sind die Funktionen des Internet-Protokolls (*IP*) durch die Funktionen des Stream-Protokolls (*ST2*$^+$) ersetzt. Dies ermöglicht nicht nur die Verwendung von Multicast-Mechanismen, die es erlauben Verarbeitungskapazitäten und Netzbandbreiten effizienter zu nutzen, sondern auch die gezielte Bevorzugung der Realzeitdaten gegenüber anderen Anwendungs- und Kontrolldaten.

Im folgenden wird die SDL-Spezifikation vorgestellt, die auf Grundlage der informellen Angaben aus dem IETF-Dokument *RFC1819* [34] erstellt worden ist. Die erstellte SDL-Spezifikation kann dabei nicht nur als eindeutige Protokollbeschreibung dienen, sondern ist auch der Ausgangspunkt für die Leistungsanalyse mit Hilfe der QUEST-Methode.

9.3 SDL-Systembeschreibung

Das SDL-System des Verteildienstes besteht aus insgesamt acht SDL-Blöcken. Dabei enthalten vier von ihnen alle notwendigen Prozesse für einen Router. Für jeden Client und für den Server existieren ebenfalls je ein SDL-Block siehe Abbildung 9-2.

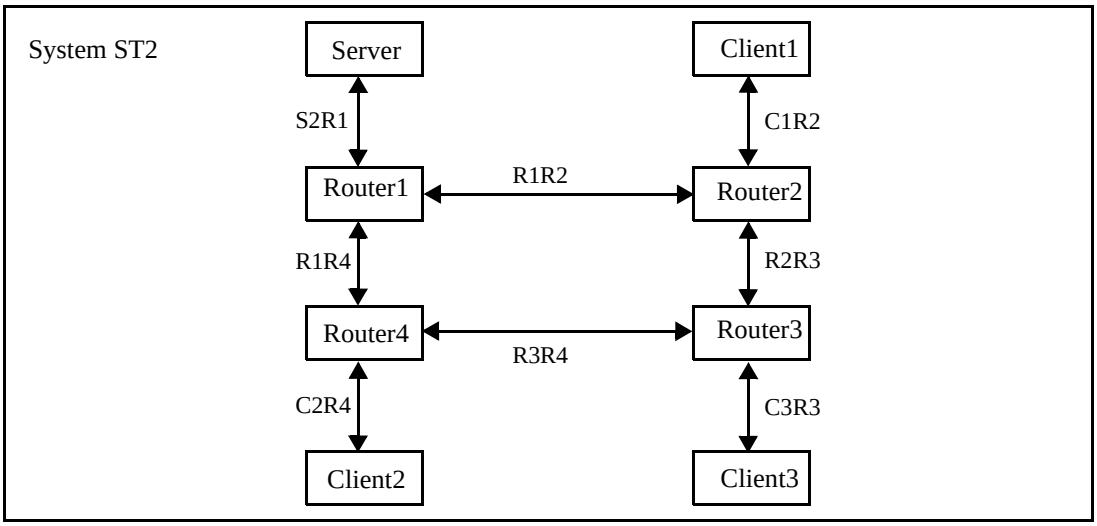


Abbildung 9-2 **SDL-System des Anwendungsszenarios (Blockebene)**

Die Blöcke *Router1* bis *Router4* sind durch bi-direktionale Kanäle verbunden. Diese transportieren ST2$^+$*-Frames*, in denen sich sowohl Anwendungsdaten als auch Kontrolldaten befinden können. Der Block *Server* ist durch einen bi-direktionalen Kanal mit dem Block *Router1* verbunden, der die Verbindung über den *Switch1* (siehe Abbildung 9-1) modelliert. Die Blöcke *Client1* bis *Client3* sind in gleicher Weise mit den Blöcken *Router2* bis *Router4* verbunden.

9.3.1 SDL-Block Server

Der SDL-Block Server enthält als wichtigste Komponente den Prozeß *SCMP* (Stream Control Message Protocol), der die Kontrollaufgaben innerhalb des Stream-Protokolls übernimmt, siehe Abbildung 9-3.

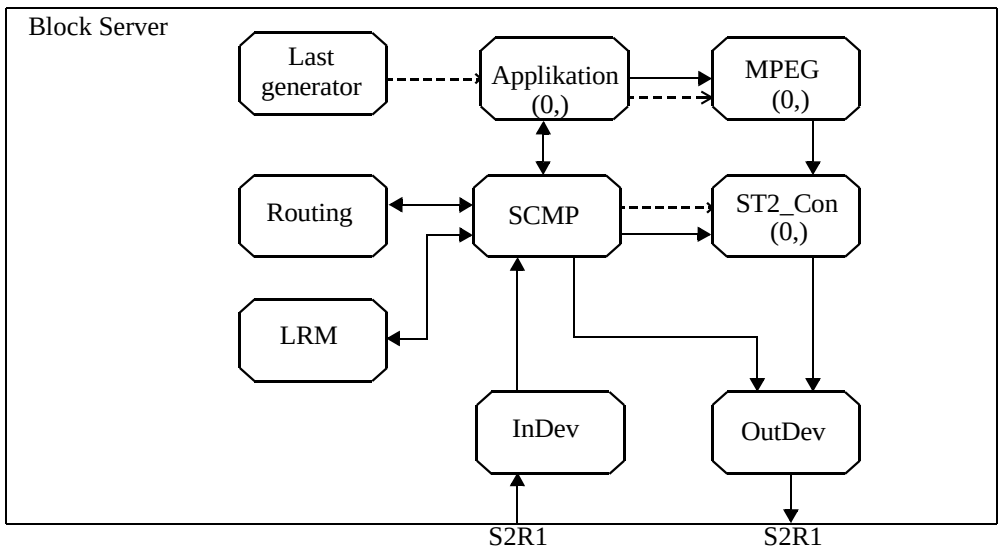


Abbildung 9-3 **Prozeßstruktur Block Server**

Darüberhinaus existiert für jeden aktiven *ST2*$^+$-Strom, der den Knoten durchläuft, genau eine Instanz des Prozesses *ST2_Con*. Dieser Prozeß ist für die Verarbeitung der Nutzdaten in den Knoten verantwortlich.

Weiterhin enthält der Block *Server* sowohl einen Prozeß *LRM* als auch einen Prozeß *Routing*, deren Existenz von dem Stream-Protokoll vorausgesetzt wird. Die Prozesse *Applikation*, *MPEG* und *Lastgenerator* bilden die Umgebung des Stream-Protokolls. Sie bedienen sich des Dienstes, der durch die Protokollinstanzen zusammen mit deren unterliegendem Dienst erbracht werden. Sie sind aus Strukturierungsgründen mit in den Block *Server* integriert worden, um so die Anzahl der Blöcke auf Systemebene zu begrenzen. Aus diesem Grund sind auch die beiden Prozesse *InDev* und *Outdev* innerhalb des Blocks spezifiziert worden. Sie bilden die Schnittstelle zum Netz und sind für die Datenübergabe an der Netzschnittstelle zuständig.

In den folgenden Abschnitten werden die wichtigsten Funktionen der einzelnen Prozesse innerhalb des Blocks *Server* beschrieben.

Der Prozeß LRM im Server

Der Prozeß *LRM* (Local Resource Manager) umfaßt alle Funktionen, die bereits in Abschnitt 8.2 und Abschnitt 8.6 beschrieben wurden. Der hier entwickelte Ressource-Manager für den Server entscheidet zuerst anhand der ihm übergebenen *FlowSpec*[a], ob auf dem lokalen Knoten genügend Kapazitäten vorhanden sind, um den neu aufzubauenden Strom zu unterstützen, ohne die QoS-Schranken der bereits etablierten Ströme zu verletzen. Falls selbst die Absenkungen der Senderate auf den Wert von *LimRate* und der maximalen Paketgröße auf den Wert von *LimMaxSize* nicht ausreichen, um die benötigten Ressourcen zu reservieren, wird dem Kontrollprozeß die Ablehnung des Aufbauwunsches für diesen Strom mitgeteilt. Dazu wird in jedes Feld der Reservierungsliste (siehe Abbildung 9-4 Mitte) der Wert *False* eingetragen und die ursprüngliche *FlowSpec* an den Kontrollprozeß zurückgegeben. Falls die Kapazitäten auf dem lokalen Knoten ausreichen, überprüft der Prozeß *LRM*, ob für die Ausgangsverbindung zum *Router1* genügend freie Bandbreite vorhanden ist. Falls ja, wird für die Ressourcen auf dem Knoten (Prozessor und Speicher) die aus dem neuen Strom resultierende Belastung errechnet und zur aktuellen Belastung des Knotens addiert. Gleichzeitig wird der Zähler für die aktiven Verbindungen um den Wert Eins erhöht.

Als Antwort sendet der Ressource-Manager eine Liste, in der das Ergebnis der Prüfung sowie die akzeptierten QoS-Werte eingetragen sind, siehe Abbildung 9-4 Mitte. Für diejenigen Ströme, für die die freien Kapazitäten ausreichen, wird dann der Aufbauwunsch weitergeleitet, für die Ströme mit unzureichender freier Kapazität wird eine entsprechende *Refuse*-Nachricht versendet. Beim Verbindungsabbau werden die reservierten Ressourcen wieder freigegeben. Dabei wird die aktuelle Belastung der lokalen Ressourcen und der Ausgangsleitung für den abzubauenden Strom berechnet und die Gesamtbelastung entsprechend reduziert. Gleichzeitig wird die Anzahl der aktiven Verbindungen um eins reduziert.

Der Prozeß Routing im Server

Der Prozeß *Routing* wird für jeden *ST2*$^+$-Strom beim Verbindungsaufbau benötigt, da er über die zur Weiterleitung der Daten- und Kontroll-Frames benötigten Informationen verfügt. Dazu ver-

a. Zum Aufbau der FlowSpec siehe Abbildung 8-2 auf Seite 147.

waltet der Prozeß eine Routing-Tabelle, in der für jede Zieladresse eingetragen ist, über welchen Nachbarknoten dieses Ziel zu ereichen ist.

Da in der hier untersuchten Topologie der *Server* alle Frames an den *Router1* über den Prozeß *OutDev* versendet, ist der Prozeß *Routing* im Server sehr einfach ausgelegt. Er antwortet auf jede Routing-Anfrage mit der Antwort *„alle Ziele sind über den Router1 erreichbar*“. Für den Fall, daß weitere Knoten in das Subnetz integriert würden, müßte die Routing-Tabelle entsprechend erweitert werden. Um den Prozeß *Routing* nicht nur für diesen Spezialfall verwenden zu können, enthält der Prozeß in diesem Block die gleichen Funktionen wie in den Blöcken *Router1* bis *Router4*. Lediglich die Routing-Tabelle (siehe Abbildung 9-6 auf Seite 179) enthält für jeden Wert von *Target Net* den Eintrag *Nexthop = R1*.

Der Prozeß SCMP im Server

Der Prozeß *SCMP* enthält den Protokollautomaten des S*tream Control Message Protocols*, wie er durch die Spezifikation [34] informell beschrieben ist. Zu den zentralen Aufgaben gehören der Auf- und Abbau sowie die Verwaltung von isochronen Realzeitströmen. Zur Erfüllung dieser Aufgaben existiert auf jedem Knoten eine Datenbasis, die alle notwendigen Informationen über die existierenden Ströme enthält, siehe Abbildung 9-4 rechts.

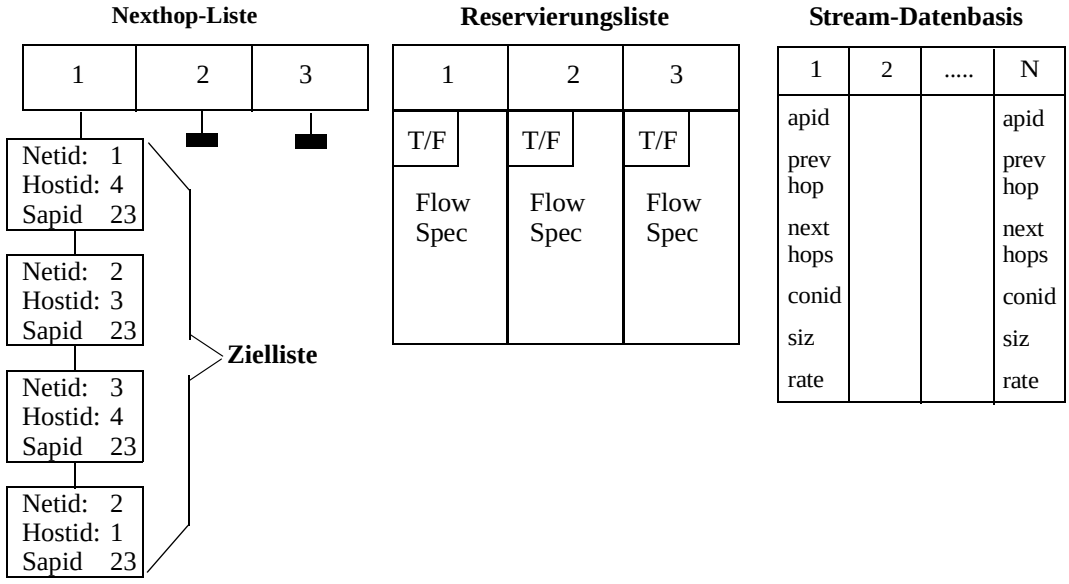


Abbildung 9-4 **Kontrollinstrumente Ressource-Management und Routing im Server**

In der Datenbasis werden die Vorgänger- und Nachfolgerknoten, die Prozeßidentifikationen der zugeordneten Anwendung und der Verbindungsinstanz sowie die aktuell gültigen QoS-Werte der Verbindung gespeichert. Die Instanz, die auf dem Audio-Server existiert, stellt die Quelle für die *ST2*$^{+}$-Ströme dar (*Origin*). Sie erhält von der Anwendung durch das Schnittstellenereignis *Open-Req* den Wunsch zum Aufbau eines Stromes. Mit diesem Ereignis wird dem Prozeß *SCMP* eine Empfängerliste, eine *FlowSpec* sowie optionale Informationen über Besonderheiten des Stroms übergeben. Zu diesen Besonderheiten zählen z.B. *Recovery*-Optionen, die Aufzeichnung des gewählten Weges oder die Festlegung, nach welchen Regeln Clients einem existierenden Strom beitreten dürfen.

Der Kontrollprozeß (SCMP) hat daraufhin eine Reihe von Aufgaben zu erfüllen:

1. Erstellung einer eindeutigen Kennung (*Stream Identifier, SID*) für den Strom, bestehend aus der Adresse des Rechners und einer zufällig[b] gezogenen ganzen Zahl aus dem Bereich $0..2^{16}-1$.
2. Bestimmung aller Nachfolgeknoten mit Hilfe der Routing-Funktion.
3. Weiterleitung der *FlowSpec* an den lokalen Ressource-Manager (*LRM*) zur Reservierung der benötigten Ressourcen.
4. Falls der Strom etabliert werden kann, wird eine lokale Datenbasis erzeugt, in der die Informationen über den neuen Strom verwaltet werden, siehe Abbildung 9-4, rechts.
5. Er informiert alle vorhandenen Nachfolgeknoten über den Aufbauwunsch durch Versendung einer *Connect*-Nachricht. In dieser werden die Stromkennung, die modifizierte *FlowSpec,* sowie eine Teilmenge der ursprünglichen Zielliste mitversendet.

Für jede *Connect*-Nachricht wird ein Timer gesetzt. Dieser wird gelöscht, wenn der Empfänger den Empfang der *Connect*-Nachricht quittiert. Empfänger ist in diesem Fall die Kontrollinstanz auf dem Knoten *Router1*. Für jeden Verbindungswunsch empfängt die Protokollinstanz schließlich entweder eine *Accept*- oder eine *Refuse*-Nachricht. Der Empfang dieser Nachrichten wird der Anwendung durch das Dienstprimitiv *OpenAcceptInd* oder *OpenRefuseInd* angezeigt. Im ersten Fall wird der zuständigen Instanz *ST2_Con* mitgeteilt, daß die Verbindung erfolgreich etabliert wurde und sie ab sofort Anwendungsdaten entgegennehmen und weiterleiten kann. Im Falle einer Ablehnung wird der Prozeß *LRM* davon informiert und die für diese Verbindung vorreservierten Kapazitäten wieder freigegeben. Darüberhinaus erhält die zuständige Verbindungsinstanz das Signal *Fin*, woraufhin sie sich ebenfalls beendet.

Der Verbindungsabbau wird durch das Dienstprimitiv *CloseReq* eingeleitet. Der Prozeß *SCMP* veranlaßt daraufhin durch Versendung des Signals *QosRelease* an den Prozeß *LRM* die Freigabe der für den abzubauenden Strom reservierten Kapazitäten. Als Parameter enthält das Signal die Senderate, die maximal verwendete Paketgröße sowie die benötigten Ausgangsleitungen. Gleichzeitig wird die Verbindungsinstanz *ST2_Con* vom Ende der Verbindung informiert, damit sie sich beenden kann.

Der Verbindungsabbauwunsch wird per *Disconnect*-Nachricht an alle Nachfolgeknoten weitergereicht. Genau wie die *Connect*-Nachricht enthält jede *Disconnect*-Nachricht eine Liste aller zu informierenden Zielrechner, und der Empfang dieser Nachricht ist von der Kontrollinstanz der Nachfolgeknoten zu bestätigen. Als letzte Aktion wird die Datenbasis für diese Verbindung gelöscht. Damit ist die Verbindung für diesen Knoten nicht mehr existent.

Der Prozeß ST2_Con im Server

Dieser Prozeß ist für den Datentransfer innerhalb des Stream-Protokolls verantwortlich. Wenn während des Verbindungsaufbaus die notwendigen Ressourcen zur Verfügung stehen, wird eine Instanz dieses Prozesses für die Abwicklung dieser Verbindung erzeugt. Direkt nach der Erzeugung erhält der Prozeß die für die Verbindung notwendigen Informationen, die in die Header der Daten-Frames eingetragen werden müssen. Zu diesen zählen die eindeutige Identifizierungsnummer (*Stream IDentifier SID*), die IP-Adresse des Knotens, aber auch die Identifikation der benötigten Ausgangsleitung.

b. Falls eine bereits verwendete Kennung generiert werden sollte, wird eine weitere Zahl gezogen.

Direkt nach der Erzeugung ist der Prozeß noch nicht bereit, Anwenderdaten entgegenzunehmen, da der Strom noch nicht vollständig aufgebaut ist. Erst wenn mindestens einer der Zielknoten mit einer *Accept*-Nachricht den erfolgreichen Verbindungsaufbau zurückmeldet, wird dies vom Prozeß *SCMP* an die für den Strom zuständige Instanz *ST2_Con* gemeldet. Diese wechselt daraufhin in den Zustand *established* und ist ab sofort bereit, Anwenderdaten entgegenzunehmen. Zusammen mit der Information über das Eintreffen einer *Accept*-Nachricht wird der Instanz *ST2_Con* auf dem Server die Identifikation ihrer Partnerinstanz (*PID* der zuständigen Instanz) auf dem *Router1* mitgeteilt. Da auf jedem aktiven Knoten für jeden etablierten Strom eine Instanz *ST2_Con* existiert und alle Instanzen *Daten-Frames* vom Prozeß *InDev* annehmen können, wird die *PID* dazu verwendet, die zuständige Instanz zu adressieren.

Im Zustand *established* nimmt der Prozeß *ST2_Con* MPEG-Frames als Anwenderdaten entgegen, stellt diese zu Daten-Frames zusammen und sendet diese dann an den Prozeß *OutDev*, der sie über das Netz weitersendet. Wenn der Strom wieder aufgelöst wird, erhält der Prozeß das Signal *Fin* vom Kontrollprozeß *SCMP*, woraufhin er die SDL-*Stop*-Aktion ausführt und sich beendet.

Die Prozesse Applikation und Lastgenerator im Server

Beide Prozesse sind nicht Bestandteil des Protokolls. Sie enthalten in der SDL-Spezifikation nur sehr rudimentäre Angaben, da sie als Lastprozesse für die Performance-Analyse eingeführt worden sind. Der Prozeß *Lastgenerator* erzeugt zyklisch Instanzen des Prozesses *Applikation*. Diese wiederum initiieren einen Verbindungsaufbau mit einer zufälligen Anzahl von Clients. Nach erfolgreichem Verbindungsaufbau erzeugen sie als Lastquelle eine Instanz des Prozesses *MPEG*. Danach pausieren sie, bevor sie schließlich den Verbindungsabbau initiieren. Im SDL-System übernehmen beide Prozesse die Aufgabe, die korrekte Funktionsweise des spezifizierten Protokolls zu überprüfen. Dazu werden durch die beiden Prozesse zunächst Verbindungen etabliert, anschließend einige Nutzdaten übertragen und die aufgebauten Verbindungen wieder geschlossen. Bei dieser Vorgehensweise handelt es sich um eine Technik, den Entwurf zu überprüfen.

Der Prozeß MPEG im Server

Die Prozesse dieses Typs fungieren als Lastquellen in dem System. Die Werte sowohl für die Senderaten als auch für die Paketgrößen, die in der *FlowSpec* zur Aushandlung der Dienstgüte verwendet werden, sind so angelegt, daß sie mit den Angaben des MPEG-Standards für Audio-Übertragung übereinstimmen, vgl. Abschnitt 5.2.2. Der Prozeß wird nach Anzeige eines erfolgreichen Verbindungsaufbaus durch den Prozeß *Applikation* erzeugt. Er sendet mit der während des Verbindungsaufbaus ausgehandelten Senderate MPEG-Frames. Die Größe der versendeten Frames entspricht dem Wert *ActMaxSize*, der mit der *FlowSpec* übergeben worden ist. Wenn die Verbindung durch den Prozeß *Applikation* wieder abgebaut wird, erhält der Prozeß *MPEG* das Signal *Fin*, woraufhin er die Sendung von Anwendungsdaten einstellt und sich beendet.

Die Prozesse InDev und OutDev im Server

Der Prozeß *Indev* dient dazu, die eingehenden Kontroll-Frames von der Netzschnittstelle anzunehmen und an die Kontrollinstanz (*SCMP*) weiterzuleiten. Der Prozeß *Outdev* bildet die Sendeeinheit der Schnittstelle und übergibt Kontroll-Frames von der Kontrollinstanz und Daten-Frames von den Verbindungsinstanzen zur Übertragung an den Router.

9.3.2 Die SDL-Blöcke Router1 bis Router4

Die SDL-Blöcke *Router1* bis *Router4* sind identisch aufgebaut. Sie unterscheiden sich lediglich durch die Kanäle, über die sie erreichbar sind. Abbildung 9-5 zeigt am Beispiel des Blocks *Router1* die Struktur eines Routers.

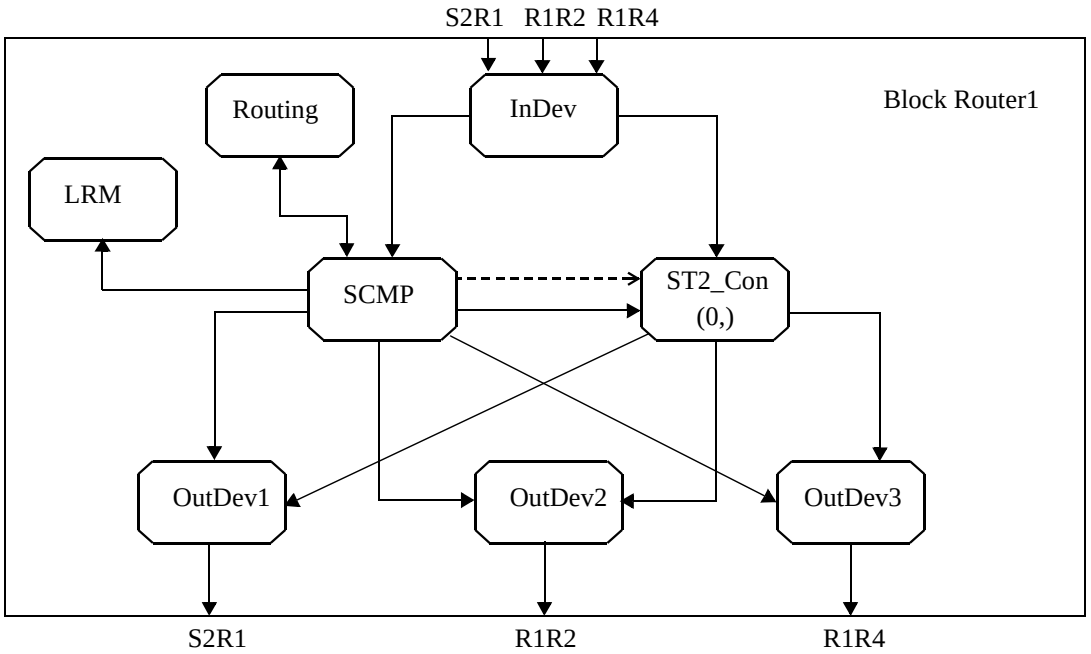


Abbildung 9-5 **Prozeßstruktur Router (Block *Router1)***

Auf jedem Router sind eine Instanz des Kontrollprotokolls (*SCMP*) und für jede aktive Verbindung eine Instanz des Prozesses *ST2_Con* vorhanden. Ebenso wie im Server existieren ein Prozeß *LRM* und eine Instanz des Prozesses *Routing*. Während im Server lediglich eine Instanz des Prozesses *OutDev* existiert, sind dies im *Router1* drei Instanzen. Die Anzahl der Instanzen innerhalb eines Blockes Router hängt von der Anzahl der Ausgangsleitungen ab, die von diesem Router ausgehen, da für jede Ausgangsleitung genau eine Instanz existiert. Vom SDL-Prozeß *InDev* existiert auf jedem Knoten genau eine Instanz.

Da auf einem Router keine Anwendungen implementiert sind, existieren die Prozesse *Applikation*, *Lastgenerator* und *MPEG* dort nicht.

Der Prozeß LRM im Router

Der Prozeß *LRM* in den Routern arbeitet genauso wie der Prozeß im Block *Server*. Der einzige Unterschied besteht darin, daß er nicht nur eine Ausgangsverbindung bezüglich freier Kapazitäten überprüfen muß. Er erhält beim Verbindungsaufbau zusammen mit der *FlowSpec* eine Liste über alle benötigten Ausgangskanäle. Diese überprüft er der Reihe nach, wenn die Prüfung der lokalen Kapazitäten einen Verbindungsaufbau zuläßt. Für alle Ausgangsleitungen mit ausreichender Kapazität werden in die Reservierungsliste der Wert *True* und die akzeptierten Dienstgüteparameter in die *FlowSpec* eingetragen, vgl. Abbildung 9-4 auf Seite 175. Für jede Leitung mit ungenügender Kapazität wird der Wert *False* zusammen mit der die Ausgangs-*FlowSpec*

zurückgegeben. Im Falle der Verbindungsauslösung arbeitet die Instanz genauso wie die im Block *Server.*

Der Prozeß Routing im Router

Zu der wesentlichen Aufgabe in den Routern gehört die Bestimmung der Nachfolgeknoten für die Datenströme. *Das ST2*$^+$-Protokoll legt für alle Ströme beim Verbindungsaufbau eine feste Route fest. Da dieses Protokoll Multicast-Mechanismen unterstützt, muß in jedem Router geprüft werden, ob der Datenstrom in diesem Knoten geteilt werden muß. *ST2*$^+$ kann die Standard-Routing-Mechanismen von *IP* verwenden. Daher ist ein statisches *Minimum Hop Routing* mit und ohne Berücksichtigung der vorhandenen Bandbreiten ebenso anwendbar wie adaptive Verfahren, die die existierenden Auslastungen berücksichtigen. Allerdings sei an dieser Stelle nochmals darauf verwiesen, daß alle *ST2*$^+$-Datenpakete denselben Weg von der Quelle zum Ziel verwenden, so daß adaptive Mechanismen ihre große Stärke nicht ausspielen können, weil nicht für jedes neue Datenpaket die aktuelle Situation im Netz berücksichtigt wird. Statt dessen wird die Situation zum Zeitpunkt des Verbindungsaufbaus als Grundlage für die Wahl der Route herangezogen.

Im hier vorgestellten Szenario wird ein statisches Routing verwendet, bei dem die unterschiedlichen Bandbreiten der Verbindungsstrecken zwischen den Routern berücksichtigt werden. Dabei wird ein vereinfachtes Adressierungsschema verwendet, bei dem jeder physikalische Rechner durch die Identifikation seines lokalen Netzes und innerhalb des Netzes durch eine Rechneridentifikation eindeutig festgelegt ist. Dieses Vorgehen ist auch aus dem IP-Adressierungsschema bekannt. Es sei nun angenommen, daß die vorhandene Bandbreite der Verbindung zwischen *Router2* und *Router3* die schwächste ist. Daher werden die Ströme zwischen *Router1* und *Router3* über den *Router4* und die Ströme zwischen *Router2* und *Router4* über den *Router1* geleitet. In den Prozessen *Routing* der Blöcke *Router1* bis *Router4* sind die folgenden Routing-Tabellen hinterlegt:

Router1

Target Net	Nexthop
1	hid
2	R2
3	R4
4	R4

Router2

Target Net	Nexthop
1	R1
2	hid
3	R3
4	R1

Router3

Target Net	Nexthop
1	R4
2	R2
3	hid
4	R4

Router4

Target Net	Nexthop
1	R1
2	R1
3	R3
4	hid

Abbildung 9-6 **Routing-Tabellen der Router**

Der Eintrag *hid* bedeutet, daß das Zielnetz erreicht ist und die entsprechende Rechneridentifikation (*hid*) aus dem Adreßfeld ausgelesen werden muß.

Der Prozeß *Routing* erhält mit jeder Anfrage *GetNextHops* eine sog. Zielliste vom Prozeß *SCMP*, in der die Adressen aller Endsysteme enthalten sind, zu denen eine Verbindung aufgebaut werden soll. Der Prozeß *Routing* überprüft mit Hilfe der Routing-Tabelle, zu welchen Nachbarknoten die Daten als nächstes zu versenden sind. Weiterhin wird die ursprüngliche Empfängerliste so umsortiert, daß mit jedem Nachfolgeknoten (*nexthop*) diejenigen Zielknoten verbunden sind, die über den jeweiligen Zwischenknoten erreichbar sind. Die Listenstrukturen sind in Abbildung 9-7 dargestellt. Die sog. *Nexthop*-Liste erhält der Kontrollprozeß mit der Nachricht *PutNextHops* zurück.

Der Prozeß SCMP im Router

Auf den Routern werden die Prozesse SCMP als *ST2*$^{+}$-Agenten ausgeführt, die lediglich Weiterleitungsfunktionen haben. In den Zwischenknoten reagieren die Kontrollprozesse beim Verbindungsaufbau auf die *Connect*-Nachricht, die sie von anderen Knoten empfangen. Der Empfang der Nachricht wird an den Vorgängerknoten per *Ack*-Nachricht bestätigt. Danach werden die folgenden Aktionen durchgeführt:

1. Bestimmung aller Nachfolgeknoten unter Zuhilfenahme der Routing-Funktion.
2. Anforderung der Reservierung der benötigten Ressourcen über den LRM.
3. Weiterleitung der *Connect*-Nachricht an alle durch die Routing-Funktion ermittelten Nachfolgeknoten.

Für die unter 1. genannte Aktion wird die in der *Connect*-Nachricht versendete Zielliste (*target list*), die in Abbildung 9-7 links dargestellt ist, entnommen und an den Prozeß *Routing* übergeben. Dieser bestimmt daraus die Ausgangsleitungen und sortiert sie zur sog. *Nexthop-Liste* um, siehe Abbildung 9-7 Mitte.

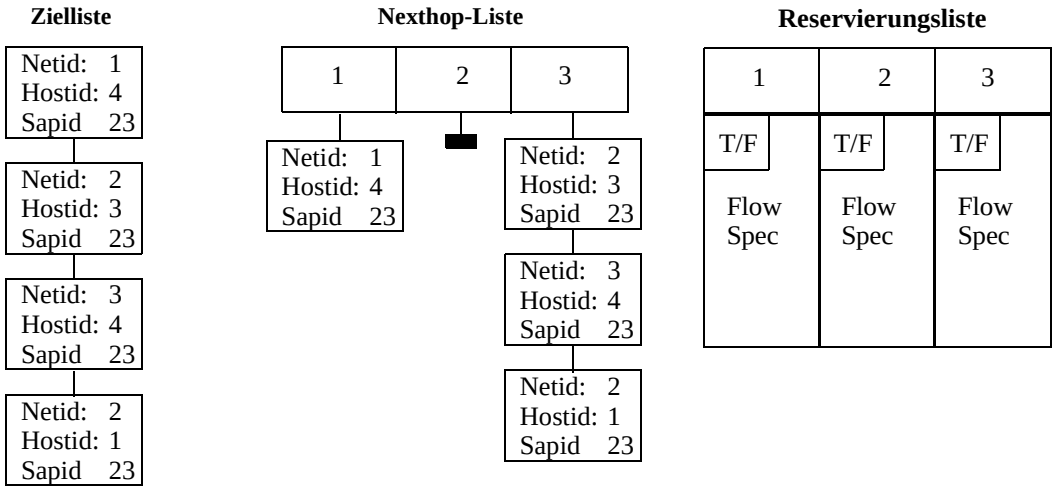


Abbildung 9-7 **Kontrollstrukturen Ressource-Management und Routing im Router**

Für die unter 2. genannte Aktion werden die benötigten Ausgangsleitungen zusammen mit der *FlowSpec* dem Prozeß *LRM* übergeben. Dieser antwortet mit der Reservierungsliste, siehe Abbildung 9-7 rechts. Anschließend wird über alle Ausgangsleitungen, für die die Reservierungsliste den Wert *True* zeigt, der Verbindungsaufbauwunsch per *Connect*-Nachricht weitergereicht. Für die Ausgangsleitungen, für die nicht genügend freie Kapazitäten vorhanden sind, erhält der Vorgängerknoten eine *Refuse*-Nachricht, in der alle Ziele aufgeführt sind, die wegen der fehlenden Leitungskapazitäten nicht erreichbar sind.

Beim Verbindungsabbau verhält sich die Instanz *SCMP* genauso wie die auf dem Server. Sie übergibt dem Prozeß *LRM* mit Hilfe des Signals *QosRelease* die Senderate und die verwendete maximale Paketgröße sowie die von der Verbindung genutzten Ausgangsleitungen. Gleichzeitig wird die Verbindungsinstanz *ST2_Con* informiert und der Verbindungseintrag in der Datenbasis gelöscht.

Der Prozeß ST2_Con im Router

Die Funktionsweise der Prozesse *ST2_Con* in den Routern ist beinahe identisch mit denen der Prozesse im Block *Server.* Der einzige Unterschied besteht darin, daß eine Instanz auf einem Router keine Anwendungsdaten per *DatReq* von der Lastquelle bekommt, sondern die Daten bereits als Protokolldateneinheit (*DataPDU*) von der Eingangsinstanz *InDev* erhält. Die Instanz ändert die Prozeßidentifikation (*PID*) für die Empfängerinstanz auf den Nachfolgeknoten und versendet die Protokolldateneinheit über alle festgelegten Ausgangsleitungen an die Nachfolgeknoten. Da es in den Routern passieren kann, daß die Multicast-Ströme geteilt werden, kann der Empfang eines Datenpaketes die Sendung mehrerer Pakete veranlassen. Dazu wird jeder Instanz des Prozesses eine Liste von der Kontrollinstanz *SCMP* zur Verfügung gestellt, in der für jeden Nachfolgeknoten die PID der Partnerinstanz enthalten ist.

Die Prozesse InDev und OutDev im Router

Der Prozeß *Indev* dient wie im Server dazu, die eingehenden Frames von der Netzschnittstelle anzunehmen und an die Kontrollinstanz (*SCMP*) bzw. an die Verbindungsinstanzen (*ST2_Con*) weiterzuleiten. Da es auf jedem Knoten mehrere gleichzeitig aktive Verbindungsinstanzen geben kann, entnimmt der Prozeß *InDev* in den Routern aus dem *Header* der Datenpakete die Prozeßidentifikation des Empfangsprozesses. Die Prozesse *Outdev* bilden die Sendeeinheit der Schnittstelle und übergeben Frames von der Kontrollinstanz und den Verbindungsinstanzen zur Datenübertragung an die angeschlossenen Rechner.

9.3.3 Die SDL-Blöcke Client1 bis Client3

Die Blöcke *Client1* bis *Client3* sind identisch strukturiert und enthalten wie der Block *Server* je einen Prozeß *Outdev,* einen Prozeß *Indev,* die vier Prozesse für das Stream-Protokoll sowie die beiden Prozesse *AppGen* und *App,* die lediglich als Datensenken fungieren. Die Struktur und die Kommunikationsbeziehungen sind am Beispiel des Blocks *Client1* in Abbildung 9-8 dargestellt.

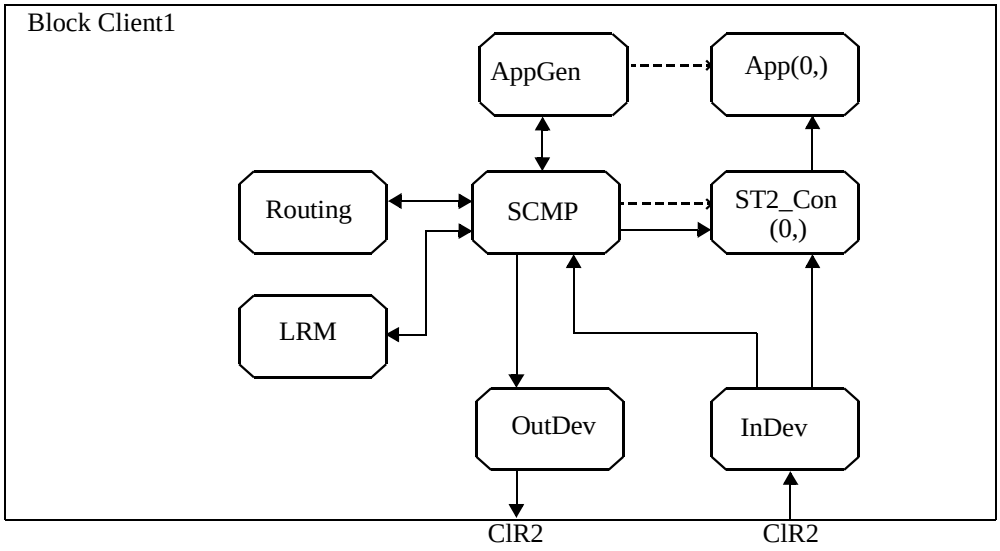


Abbildung 9-8 **Prozeßstruktur Clients (Block *Client1)***

Wie schon im Block *Server* sind die Prozesse, die die Nutzer des durch das Stream-Protokoll erbrachten Dienstes aus Strukturierungsgründen in den Block integriert worden. Daher kann man die SDL-Blockgrenzen in diesem System als physikalische Grenzen der betrachteten Rechner ansehen.

Der Prozeß LRM im Client

Da alle Clients als Empfänger der Anwendungsdaten auftreten, muß der Prozeß *LRM* in den Blöcken *Client1* bis *Client3* lediglich die lokalen Ressourcen auf dem Endsystem verwalten. Daher überprüft er wie die Instanzen auf den Routern zunächst die Verfügbarkeit freier Kapazitäten auf dem Knoten. Sind diese verfügbar, wird die Verbindung akzeptiert und die Reservierung entsprechend vorgenommen. Beim Verbindungsabbau übernimmt der Prozeß von der Kontrollinstanz lediglich die Senderate sowie die maximale Paketgröße. Er reduziert daraufhin die Anzahl der aktiven Verbindungen und die aktuellen Belastungen der Ressourcen um die Belastung, die durch den abzubauenden Strom entstanden ist.

Der Prozeß Routing im Client

Der Prozeß *Routing* wird in den Clients eigentlich gar nicht benötigt, da keine Datenpakete versendet werden müssen und die Kontrollpakete alle an den mit dem Client verbundenen Router versendet werden. Da das Stream-Protokoll aber Doppelrollen vorsieht, ist der Prozeß *Routing* in die Clients integriert worden, um das System gegebenfalls leicht anpassen zu können. Doppelrolle in diesem Kontext bedeutet, daß ein Knoten bezüglich eines Streams sowohl als Client (*Target*) als auch als Zwischenknoten (*Router*) gleichzeitig auftreten kann. In diesem Fall wird die Routing-Funktion sehr wohl benötigt.

Der Prozeß SCMP im Client

Der Kontrollprozeß in den Clients, siehe Abbildung 9-8, übernimmt die Rolle eines sog. *Targets* nach *ST2*$^+$-Terminologie. Ein Rechner, der nur als *Target* fungiert, reagiert auf *Connect*-Nachrichten, die er von seinem direkten Vorgängerknoten erhält. Auch der Empfang dieser Nachricht ist an den Absender zu bestätigen. Danach führt er die folgenden Aktionen durch:

1. Anforderung und Reservierung der benötigten Kapazitäten auf dem lokalen Knoten.
2. Benachrichtigung der lokalen Anwendung, daß ein Stromaufbauwunsch vorliegt. Diese Benachrichtigung wird durch das Schnittstellenereignis *OpenInd* an den Prozeß *AppGen* vorgenommen.
3. Je nach Reaktion der Anwendung (*OpenAccept* oder *OpenRefuse*) wird eine *Accept*- oder eine *Refuse*-Nachricht an den Vorgängerknoten gesendet. Die *Accept*-Nachricht enthält die *FlowSpec*, wie sie von der Zielanwendung akzeptiert wurde.

Beim Verbindungsabbau, der durch eine *Disconnect*-Nachricht angezeigt wird, versorgt der Prozeß *SCMP* den lokalen Ressource-Manager mit der aktuellen Senderate und der aktuell verwendeten maximalen Paketgröße des Stroms. Darüberhinaus wird die Verbindungsinstanz informiert und der lokalen Anwendung durch das Schnittstellenereignis *CloseInd* der erfolgte Verbindungsabbau mitgeteilt.

Der Prozeß ST2_Con im Client

Der Prozeß ST2_Con liefert die per Daten-Frame empfangenen Anwendungsdaten an die zugehörige Instanz des Prozesses *App* aus. Beim Empfang des Signals *Fin* beendet er sich.

Die Prozesse OutDev und InDev im Client

Der Prozeß *Outdev* bildet die Netzschnittstelle und liefert Kontroll-Frames, die er vom Prozeß *SCMP* empfängt, über den Kanal *CLR* an den Prozeß *InDev* des zugeordneten Routers. Analog empfängt der Prozeß *InDev* Daten- und Kontroll-Frames über die Netzschnittstelle und liefert sie an die entsprechende *ST2_Con*-Instanz bzw. an die Kontrollinstanz *SCMP.*

Die Prozesse AppGen und App

Die beiden Prozesse bilden die Anwendungsinstanzen auf jedem Client-Rechner. Der Prozeß *AppGen* nimmt die Dienstprimitive *CloseInd* und *OpenInd* an. Im Falle eines *OpenInd* antwortet er entweder mit dem Dienstprimitiv *OpenAccept* oder *OpenRefuse*. Die Entscheidungsgrundlage ist im SDL-System nicht spezifiziert, sondern wird durch das *Decision-Any*-Konstrukt ausgedrückt. Das Dienstprimitiv *CloseInd* informiert den Prozeß *AppGen* von der Beendigung eines $ST2^+$-Stroms. Falls der Prozeß *AppGen* dem Verbindungsaufbauwunsch mit *OpenAccept* zustimmt, erzeugt er gleichzeitig eine Instanz des Prozesses *App*. Dieser Prozeß dient als Datensenke für die Anwendungsdaten des Stroms. Er beendet sich, wenn er das Signal *Fin* erhält. Dieses wird ihm vom Prozeß SCMP gesendet, wenn die ihm zugeordnete Verbindung wieder aufgelöst wird.

Damit ist das SDL-System des Audio-Server-Szenarios vollständig beschrieben. Im folgenden werden nun die Ergänzungen erläutert, die notwendig sind, um die gesuchten Performance-Maße zu ermitteln.

9.4 Das QSDL-Modell

Zur Erstellung der Performance-Spezifikation sind zunächst diejenigen SDL-Aktionen zu identifizieren, deren Bearbeitungszeit im implementierten System in Form von Maschinendienstaufrufen berücksichtigt werden sollen. Zu diesen Aktionen zählen die Übertragung der Daten- und Kontrollpakete über die Verbindungsleitungen zwischen den Knoten. Weiterhin ergeben sich Bearbeitungszeiten bei der Übergabe und Übernahme von Daten zwischen den Protokoll- und Anwendungsinstanzen. Schließlich müssen die Anwendungsdaten in den Knoten zu Protokolldateneinheiten verpackt werden. Dabei müssen die notwendigen Header-Informationen entweder berechnet oder aus der Datenbasis ausgelesen werden. Für diese zeitverbrauchenden SDL-Aktionen werden in einem ersten Schritt QSDL-Maschinendienste auf der Systemebene definiert, durch die die Zeitverbräuche modelliert werden, siehe Abbildung 9-9.

```
/*## MACHINESERVICE transmitlow(Natural,Natural), transmitmed(Natural,Natural),
                    transmithigh(Natural,Natural), getdata(Real,Natural), getinfo(Real,Natural),
                    pushdata(Real,Natural), computepdu(Real,Natural) ; ##*/
```

Abbildung 9-9 **Maschinendienste des Systems Audioserver**

Alle Maschinendienste haben zwei Aufrufparameter. Der erste Parameter gibt den Bedienwunsch an und der zweite Parameter die gewünschte Prioritätsklasse. Alle Aktionen, die von den Kontrollinstanzen ausgeführt werden, haben eine niedrigere Priorität als solche Aktionen, die von den Verbindungsinstanzen ausgeführt werden. Ebenso werden Daten-Frames bei der Netzübertragung gegenüber Kontroll-Frames bevorzugt, da lediglich die Anwendungsdaten Realzeitanforderungen haben und sie daher bevorzugt behandelt werden müssen. Darüberhinaus basieren die Berechnungen und Reservierungsstrategien in den entworfenen lokalen Ressource-Managern darauf, daß alle Daten, die mit höchster Priorität bearbeitet werden, ihre Bedienwünsche anmelden müssen. Die systemweit definierten Maschinendienste können nun von den QSDL-Maschinen in den einzelnen Blöcken angeboten werden.

9.4.1 Ergänzung der Performance-Information im Block Server

Der Block Server enthält drei QSDL-Maschinen, siehe Abbildung 9-10.

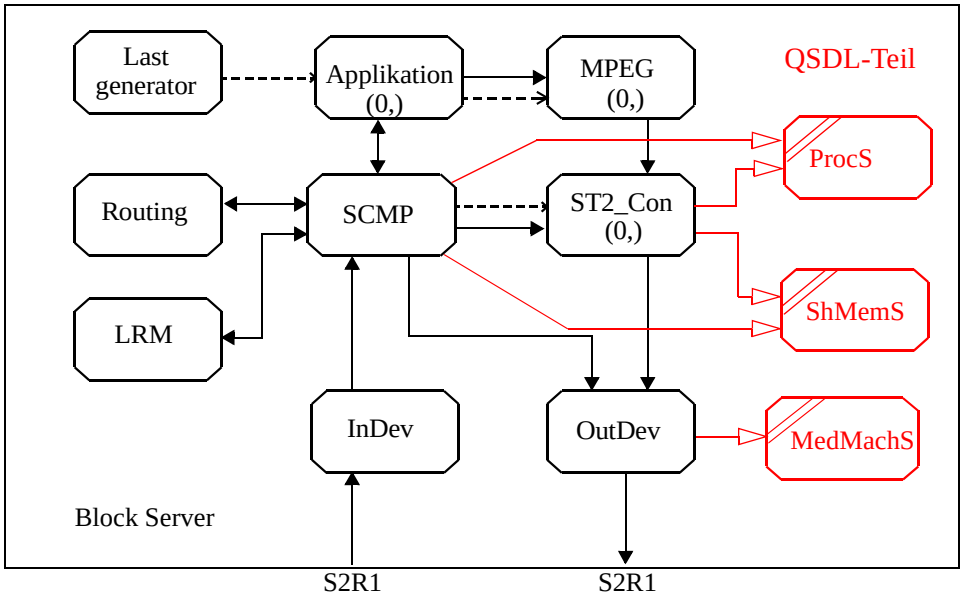


Abbildung 9-10 **Maschinen und Links im Block Server**

Die Maschine *ProcS* modelliert die Zeitverbräuche, die für die Datenübernahme, die Datenübergabe sowie die Erstellung der Protokolldateneinheiten benötigt werden. Sie bietet die Maschinendienste *getdata*, *pushdata* und *computepdu* an, die vom Prozeß *SCMP* für die Bearbeitung der Kontrolldaten und von den Prozessen *ST2_Con* für die Bearbeitung der Nutzdaten aufgerufen werden, siehe Abbildung 9-11 links.

Die Maschine *ShMemS* modelliert die Zeit, die benötigt wird, um auf die gemeinsame Datenbasis zuzugreifen. Sie bietet den Maschinendienst *getinfo* an, der bei der Zusammenstellung der Header-Informationen aufgerufen wird, siehe Abbildung 9-11 Mitte.

Die Maschine *MedMachS* modelliert die Zeit, die benötigt wird, die zur Übertragung bereitstehenden Pakete an der Netzschnittstelle zu übergeben. Sie bietet den Maschinendienst *transmit-low* an, der vom Prozeß *OutDev* aufgerufen wird wenn er einen Daten-Frame oder einen Kontroll-Frame zu übertragen hat.

```
/*##
MACHINE ProcS;
SERVER: NoServ;
DISCIPLINE: PS;
OFFERS getdata: getdataspeed,
       pushdata: pushdataspeed,
       computepdu computespeed;
ENDMACHINE ProcS;
##*/
```

```
/*##
MACHINE ShMemS;
SERVER: NoDatabasis;
DISCIPLINE: FCFSPNP;
OFFERS getinfo: getinfospeed;
ENDMACHINE ShMemS;
##*/
```

```
/*##
MACHINE MedMachS;
SERVER: 1;
DISCIPLINE: FCFSPNP;
OFFERS
  transmitlow: bandwidthSR;
ENDMACHINE MedMachS;
##*/
```

Abbildung 9-11 **Maschinendeklaration Block Server**

Bei Kontroll-Frames erhält der Request einen Prioritätswert von Null, während bei Daten-Frames der Wert Eins benutzt wird. Da die Maschine *MedMachS* mit der Bedienstrategie *FCFSPNP*, also mit nichtunterbrechender Priorität arbeitet, werden Daten-Frames bevorzugt bearbeitet.

9.4.2 Ergänzung der Performance-Information in den Blöcken Router1 bis Router4

Die Zuordnung von Maschinen zu den Prozessen ist in den Blöcken *Router1* bis *Router3* identisch mit der Zuordnung im Block Server, siehe Abbildung 9-12.

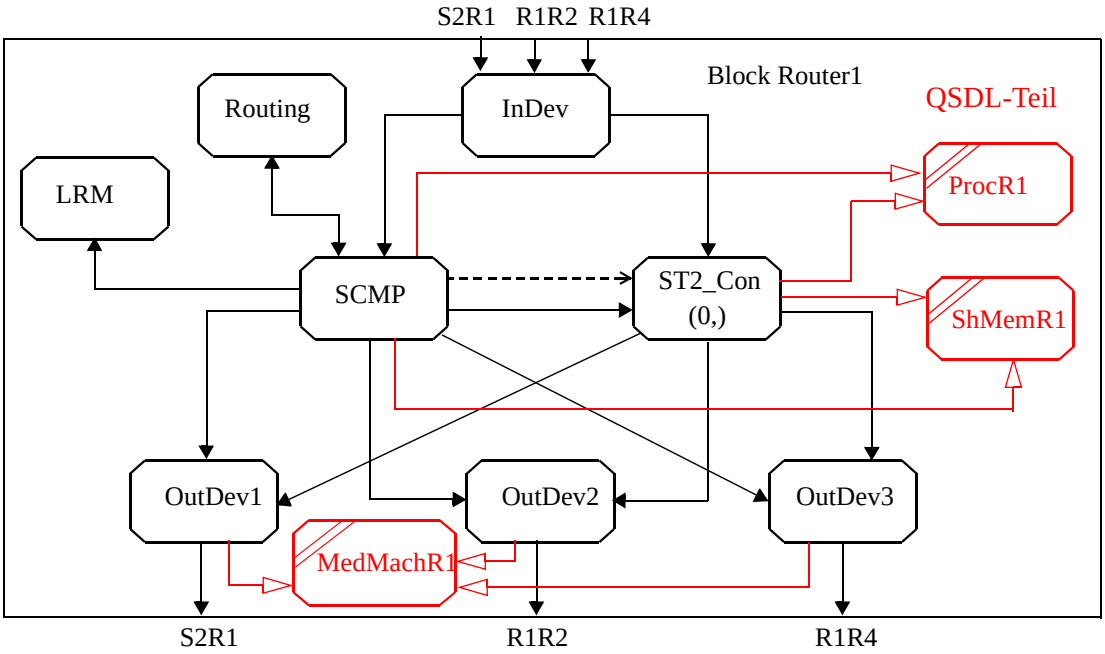


Abbildung 9-12 **Maschinen und Links im Block Router1**

Auch hier bieten die Maschinen *ProcR1* und *ShMemR1* die Dienste für die Kontroll- und die Verbindungsinstanz an, während die Maschine *MedMachR1* für die Modellierung des Zeitverbrauchs für die Frame-Übertragung genutzt wird. Ein Unterschied ergibt sich, wenn man die Maschinendeklaration in den Routern mit der im Block *Server* vergleicht, siehe Abbildung 9-13.

Während die Maschine *MedMachS* in Block *Server* insgesamt nur einen Bediener hat, besitzt die Maschine *MedMachR1* für jeden Prozeß *OutDev* einen Bediener. Da jeder der Prozesse *OutDev1* bis *OutDev3* zu jedem Zeitpunkt maximal einen aktiven Request besitzt, ist die Deklaration mit einer Maschine und drei Bedienern identisch zu einer Deklaration von drei Maschinen mit jeweils einem Bediener. Dies gilt allerdings nur dann, wenn die Kanäle identische Bandbreiten haben, wobei diese durch die Maschinengeschwindigkeit ausgedrückt wird.

```
/*##
MACHINE MedMachS;
SERVER 1;
DISCIPLINE FCFSPNP;
OFFERS transmitlow: bandwidthSR;
ENDMACHINE MedMachS;
##*/
```

```
/*##
MACHINE MedMachR1;
SERVER 3;
DISCIPLINE FCFSPNP;
OFFERS transmitlow: bandwidthSR,
       transmithigh bandwidthR1R2;
ENDMACHINE MedMachR1;
##*/
```

Abbildung 9-13 **Maschinendeklaration MedMach im Block Server und im Block Router1**

Falls die Ausgangskanäle unterschiedliche Bandbreiten aufweisen, bieten sich zwei Lösungen an. Entweder man deklariert separate Maschinen, die dann den Maschinendienst *transmit* mit unterschiedlichen Geschwindigkeiten anbieten, oder man deklariert unterschiedliche Maschinendienste *transmit*, die dann von einer Maschine mit unterschiedlichen Geschwindigkeiten angeboten werden. Der zweite Lösungsansatz ist im vorliegenden Szenario in den Routern gewählt worden, da sich die Bandbreiten in den Links zu den Clients bzw. zum Server von denen zwischen den Routern unterscheiden.

So bietet die Maschine *MedMachR1* den Maschinendienst *transmit* mit einer Geschwindigkeit von *bandwidthSR* an, vgl. Abbildung 9-13. Diese Geschwindigkeit entspricht der Bandbreite des Links zwischen dem *Server* und dem *Router1*. Der Maschinendienst *transmit2* wird mit einer Geschwindigkeit von *bandwidthR1R2* angeboten. Diese Geschwindigkeit entspricht der Bandbreite zwischen dem *Router1* und dem *Router2*, die auch der Link zwischen *Router1* und *Router4* aufweist.

Die Dienstaufrufe in den Prozessen *OutDev1* bis *OutDev3* geschehen analog zu den Aufrufen im Block *Server*.

9.4.3 Ergänzung der Performance-Information in den Blöcken Client1 bis Client3

Die Blöcke *Client1* bis *Client3* enthalten wie der Block *Server* drei Maschinen, die als Einbedienersysteme deklariert sind, siehe Abbildung 9-14.

Während die Maschinen *ProcC1* und *ShMemC1* sowohl bei der Bearbeitung von Anwendungsdaten als auch bei der Bearbeitung von Kontrolldaten Requests bearbeiten, wird die Maschine *MedMachC1* lediglich bei der Übertragung von Kontrolldaten zurück zum jeweiligen Server benötigt, da Anwendungsdaten nur an der oberen Dienstschnittstelle übergeben werden.

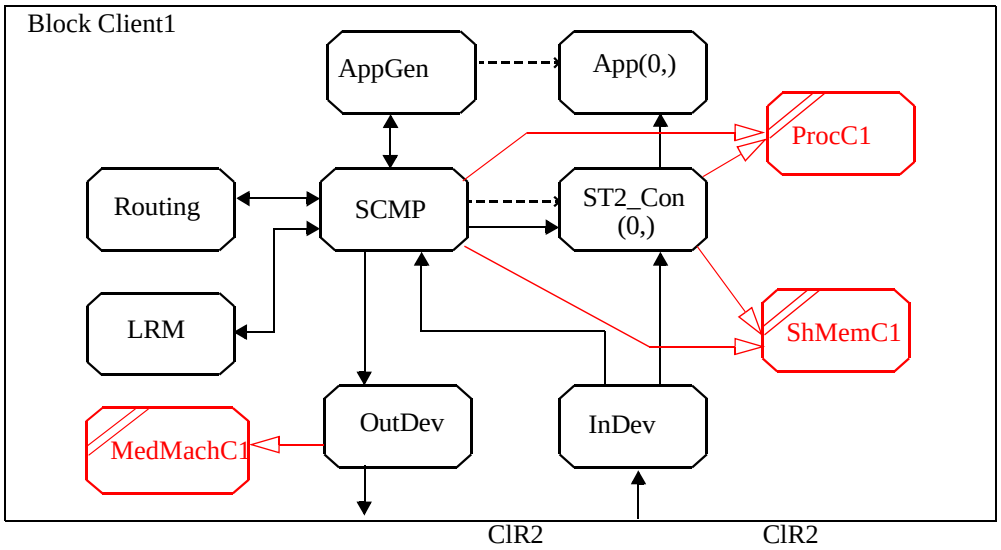


Abbildung 9-14 **Maschinen und Links im BlockClient1**

9.4.4 Ergänzung der Performance-Information in den Anwendungen des Blocks Server

Das zeitliche Verhalten der Anwendungsprozesse *LastGenerator*, *Applikation* und *MPEG* legt die Last fest, die das Transportsystem bewältigen muß. Dabei sind der mittlere Abstand zwischen neuen Verbindungen, die Länge einer Verbindung sowie das Datenaufkommen pro Zeit je Verbindung für die Gesamtbelastung des Systems relevant. Diese Größen werden zum Teil durch die Anwendungsprozesse und zum Teil durch die lokalen Ressource-Manager und somit durch die Belastung des Transportsystems während des Verbindungsaufbaus festgelegt.

Der Prozeß Lastgenerator

Dieser Prozeß legt den mittleren Abstand zwischen zwei Verbindungsaufbauwünschen sowie die Länge einer *ST2*$^+$-Verbindung fest. Dazu ist der ursprüngliche SDL-Prozeß nur geringfügig zu erweitern, siehe Abbildung 9-15.

Für den Prozeß sind die Variablen *streamlength* und *streampause* zu deklarieren, die für die Generierung von Zufallsströmen benötigt werden. Diese werden mit der Starttransition initialisiert. Der Abstand zwischen zwei Verbindungsaufbauwünschen folgt einer Exponentialverteilung, während die Länge der etablierten Ströme einer Normalverteilung folgen. Diese Annahmen sind aus den Untersuchungen in Kapitel 8 übernommen worden. Sollten sich bei Messungen realer *ST2*$^+$-Ströme andere Verteilungstypen ergeben, so können diese einfach in das Modell integriert werden.

```
PROCESS LastGenerator;
   DCL
      next Real,
      len Real,
      streamlength Normal,
      streampause Negexp;

   START;
   TASK streampause := INIT(streampause, lambda),
        streamlength := INIT(streamlength,meandur,vardur),
        next := SAMPLE(streampause),
        len := SAMPLE(streamlength);
   NEXTSTATE ready;
ENDPROCESS;

STATE ready /*## AWAKE next ##*/;
INPUT NONE;
CREATE Applikation(len);
TASK next := SAMPLE(streampause),
     len := SAMPLE(streamlength);
NEXTSTATE ready;
```

Abbildung 9-15 **Prozeß Lastgenerator (QSDL/PR)**

Der Prozeß ermittelt je einen Wert aus beiden Zufallsströmen und wechselt in den Zustand *ready*. Dieser ist ein zeitbehafteter Zustand, da der Prozeß solange warten muß, bis die awake-Zeitspanne abgelaufen ist. In der Spontantransition wird festgelegt, daß eine Instanz des Prozesses *Applikation* zu erzeugen ist. Bei der Erzeugung bekommt die Instanz den Wert für die Länge der zu erzeugenden Verbindung mit. Der Prozeß zieht neue Werte aus den Zufallsströmen und pausiert bis zur Erzeugung der nächsten Instanz. Es entsteht also ein Laststrom, indem mit exponentiell verteilten Zwischenankunftsabständen versucht wird, neue Verbindungen zu etablieren.

Der Prozeß Applikation

Die Instanzen des Prozesses *Applikation* werden dynamisch durch den Prozeß *Lastgenerator* erzeugt. Eine Instanz dieses Prozesses ist für genau einen *ST2*$^+$-Strom verantwortlich. Bei ihrer Erzeugung wird bereits durch den formalen Parameter *contime* festgelegt, wie lange die Verbindung dauern soll, die von ihr aufgebaut wird, siehe Abbildung 9-16. Die Instanz wählt in ihrer ersten Spontantransistion aus der Menge der ihr bekannten potentiellen Empfänger eine zufällige Anzahl von Adressaten aus.

```
PROCESS Applikation;
  FPAR contime Real;
  START;
  NEXTSTATE ready,
  STATE ready;
  INPUT NONE;
        :
    OpenReq(mytargetlist,myspec);
    NEXTSTATE waitacks;


ENDPROCESS;

STATE waitacks;
INPUT OpenAcceptInd(targetlist,myspec,mysource,mysession);
    DECISION reack = noack;
      (FALSE) : NEXTSTATE -;
      (TRUE)  : CREATE MPEG;
                OUTPUT QosPara(rate, siz, mysource);
                NEXTSTATE established;
    ENDDECISION;

STATE established /*## AWAKE contime ##*/;
INPUT NONE;
    OUTPUT Fin TO OFFSPRING;
    OUTPUT CloseReq(mytargetlist, mysession);
    STOP;
```

Abbildung 9-16 **Ausschnitt aus dem Prozeß Applikation (QSDL/PR)**

Die Adressatenliste übergibt sie dann zusammen mit der *FlowSpec* als Parameter des Dienstprimitivs *OpenReq* an den Prozeß *SCMP* des Servers. Die Instanz wechselt daraufhin in den Zustand *waitacks*. Nun wartet sie, bis die Antworten auf diesen Verbindungsaufbauwunsch wieder bei ihr Eintreffen. Falls alle Reaktionen eingetroffen sind, kennt diese Instanz alle Empfänger, die die Verbindung akzeptiert haben und die QoS-Werte, die während des Verbindungsaufbaus ausgehandelt worden sind. Sie erzeugt nun eine Instanz des Prozesses *MPEG* und übergibt dieser Instanz die Werte für die Senderate, die zu verwendende MPEG-Frame-Größe und die Prozeßidentifikation der Instanz *ST2_Con*. Sie wechselt in den Zustand *established*. Dieser Zustand ist ebenfalls zeitbehaftet. Die Instanz verweilt solange in diesem Zustand, wie es der aktuelle Wert des formalen Parameters *contime* festlegt. Nach Ablauf dieser Zeitspanne sendet die Instanz der Quelle das Signal *Fin* und leitet selbst per *CloseReq* den Verbindungsabbau ein.

Der Prozeß MPEG

Der Prozeß MPEG fungiert als Lastquelle, da er die Anwendungsdaten für die *ST2*$^+$-Ströme erzeugt. Er bekommt die notwendigen quantitativen Parameter, die während des Verbindungsaufbaus von den Instanzen des Kontrollprotokolls ausgehandelt worden sind, von der Instanz *Applikation*. Während seiner Existenz führt er immer wieder dieselbe Transition aus:

Er stellt im Zustand *activ*[c] einen MPEG-Frame zusammen, sendet diesen per *DatReq* an die für ihn zuständige Instanz des Prozesses *ST2_Con* und pausiert wieder, siehe Abbildung 9-17.

```
PROCESS MPEG;
  START;
  NEXTSTATE ready;
  STATE ready;
    INPUT QosPara(myrate,mysize,mypid);
      TASK pause := 1 / myrate;
      NEXTSTATE activ  /*## AWAKE pause ##*/;

ENDPROCESS;
```

```
STATE activ /*## AWAKE pause ##*/;
    INPUT NONE;
      TASK myframe!tmpstmp := NOW,
           myframe!len := mysize,
      OUTPUT DatReq(myframe) TO mypid;
      NEXTSTATE -;
INPUT Fin;
    STOP;
```

Abbildung 9-17 **Prozeß MPEG (QSDL/PR)**

Die Pause zwischen zwei aufeinanderfolgenden Frames ist festgelegt als Kehrwert der ausgehandelten Senderate. Falls er das Signal *Fin* empfängt, stellt der Prozeß die Sendung von MPEG-Frames ein und beendet sich.

9.4.5 Monitoring-Ergänzungen in den Anwendungen der Clients

Die Prozesse *AppGen* und *App* in den Blöcken *Client1* bis *Client3* modellieren das Verhalten der Anwendungen in dem Audioverteilsystem. Während der Prozeß *AppGen* die Kontrolldaten entgegennimmt fungiert jede Instanz des Prozesses *App* als Datensenke für genau einen *ST2*$^+$-Strom.

c. Der Zustandsname *active* ist nicht zulässig, da mit diesem SDL-Schlüsselwort der Zustand von Timern abgefragt werden kann.

Der Prozeß AppGen

Er nimmt die Dienstprimitive *OpenInd* und *CloseInd* entgegen. Bei Empfang des Signals *OpenInd* kann er wahlweise per *OpenRefuse* oder per *OpenAccept* reagieren. Im letzteren Fall erzeugt er eine Instanz des Prozesse *App*. In diesem Prozeß wäre in einem realen System die Client-seitige Anwendung implementiert. Da in dieser Studie lediglich die Leistungsfähigkeit des Transportsystems bewertet werden soll, sind diese rudimentären Angaben zum Verhalten der Anwendung völlig ausreichend.

Der Prozeß App

In den Instanzen dieses Prozesses werden die QSDL-Sensoren definiert, die die Dienstgüte beobachten, wie sie aus Anwendersicht erfahren wird. Dazu erhält jede Instanz drei Sensoren, siehe Abbildung 9-18. Sie werden mit dem Empfang jedes Signals *DataInd* aktualisiert. Die Prozesse ermitteln den Durchsatz, der für diesen Strom erzielt wird, sowie die Übertragungszeit und die Übertragungszeitschwankungen (*Jitter*).

Die Größe des Jitter-Wertes gibt einen Hinweis auf die Größe des benötigten Playout-Puffers. Dieser Puffer wird benötigt, um für isochrone Daten die Laufzeitschwankungen auszugleichen. Die Definition des Jitters ist allerdings in der Literatur nicht einheitlich. Während M. Zitterbart in [116] Jitter als „Laufzeitschwankungen sequentiell aufeinanderfolgender Dateneinheiten“ definiert, bezeichnet A. Danthine in [31] Jitter als „Differenz zwischen der minimalen und maximalen Laufzeit von Dateneinheiten während der Existenz einer Verbindung“.

Die Sensoren im QSDL-System sind in der Lage, beide Werte zu ermitteln. Während der QSDL-Sensor *jitter* die Laufzeitschwankung gemäß der Definition nach Zitterbart ermittelt, kann nach Beendigung der Verbindung der Jitter-Wert gemäß der Definition nach Danthine über den Sensor *del* ermittelt werden. Dieser Sensor ermittelt die Verzögerungen der übertragenen Dateneinheiten. Da dieser Sensor vom Type *BASETALLY* ist, steht sowohl der Wert des Maximums als auch der des Minimums zur Verfügung, so daß lediglich eine Differenzbildung vorgenommen werden muß.

```
PROCESS App;
      DCL data MPEG_Frame,
          diff    Duration,
          last    Duration,
          jitt Duration,
          lastdiff Duration,
          pay Natural;
/*## SENSOR del BASETALLY; ##*/
/*## SENSOR thru BASECOUNTER; ##*/
/*## SENSOR jitter BASETALLY; ##*/
      START;
      TASK lastdiff := 0.0;
      NEXTSTATE ready;
ENDPROCESS App;

STATE ready;
   INPUT DataInd(data);
    TASK diff := NOW - data!tmpstmp,
                pay := data!len;
    /*## UPDATE (thru,pay); ##*/
    /*## UPDATE (del,diff); ##*/
    DECISION lastdiff = 0.0;
    (TRUE) : TASK lastdiff := diff;
             NEXTSTATE -;
    (FALSE) : TASK jitt := CALL abs(difflast,diff),
                           difflast := diff;
              /*## UPDATE(jitter,jitt);
              NEXTSTATE -;
    ENDDECISION;
  INPUT Fin;
  STOP;
```

Abbildung 9-18 **Prozeß App (QSDL/PR)**

Im folgenden Abschnitt werden nun die anwenderorientierten Leistungsmaße ermittelt und damit die Performance-Analyse des Audioverteilsystems abgeschlossen.

9.5 Experimente und Ergebnisse

Die erreichbare Dienstgüte sowohl aus Anwendersicht als auch aus Sicht des Systembetreibers ist von der Verarbeitungskapazität und der dem System übergebenen Last abhängig. Die Kapazität in den Knoten ergibt sich durch die Geschwindigkeit der dienstanbietenden Maschinen sowie durch die Anzahl der vorhandenen Bediener. Gleiches gilt für die vorhandene Übertragungskapazität der Verbindungsleitungen.

Die vom System zu bewältigende Last hängt zum einen von der Anzahl der gleichzeitig aktiven Nutzer und von der Last eines einzelnen Benutzers ab. Daher ergeben sich eine Vielzahl von Parametern, deren Änderung einen Einfluß auf die erzielbare Dienstgüte haben.

Das Lastverhalten einer einzelnen *ST2*$^+$-Sitzung hängt zum einen von den während des Verbindungsaufbaus ausgehandelten Parametern ab. Darüberhinaus legt die Dauer einer solchen Verbindung die Länge der Systembelastung fest. Zusammen mit den Zwischenankunftsabständen[d] neuer *ST2*$^+$-Ströme und der maximal zulässigen Anzahl dieser Ströme kann das Lastaufkommen vollständig beschrieben werden. Die verwendeten Parameter sind in Tabelle 9-1 dargestellt.

Beschreibung	Verteilung	Parameter
Erzeugungsrate für neue ST2$^+$-Ströme	Exponentialverteilung	λ = 0.5 fest
Länge eines ST2$^+$-Stromes	Normalverteilung	μ = 50.0 fest; σ = 2.0 fest
Max. Anzahl paralleler ST2$^+$-Ströme	Deterministisch	P = 8 ... 20 variabel
Max. Größe verwendeter Datenpakete	Deterministisch	P = 520 ... 460 variabel

Tabelle 9-1 **Lastparameter Audioverteildienst**

Neue Aufbauwünsche für *ST2*$^+$-Ströme werden durch den Prozeß *Lastgenerator* gemäß einer Exponentialverteilung mit einer Rate von λ = 0.5 pro Sekunde erzeugt. Die Länge eines jeden Stroms folgt einer Normalverteilung mit Mittelwert von 50.0 s und einer Standardabweichung von 2.0 s. Die Parameter sind dabei so gewählt, daß ein System ohne Zugangskontrolle überfordert wäre, da die Verarbeitungskapazitäten für eine solche Last auf keinem Knoten vorhanden sind. Allerdings wird jeder Knoten durch die Ressourcenverwaltung des lokalen Ressource-Managers vor Überlastung geschützt, vgl. Abschnitt 8.6.

Die Anzahl der parallel zugelassenen Ströme wird in den Experimentserien variiert. Je nach Leistungsfähigkeit der Maschinen in den Knoten werden zwischen 8 und 20 Ströme zugelassen. Der Prozeß *LRM* verwendet den Wert des Parameters um zu entscheiden, ob weitere Verbindungen zugelassen werden können. Ebenfalls variabel ist die verwendete Paketgröße der Anwendungsdaten. Diese kann zwischen 520 Bytes und 416 Bytes schwanken. Der konkrete Wert wird beim Verbindungsaufbau ausgehandelt und ist dann je Strom festgelegt.

Neben den Lastparametern sind die in Tabelle 9-2 dargestellten Bandbreiten, Verarbeitungsgeschwindigkeiten und Auslastungsgrenzen festgelegt worden. Die Bandbreiten zwischen den Endsystemen und den Routern sind mit 1250000 Bytes/s entsprechend 10 MBit/s festgelegt worden. Entsprechend ergeben sich die Bandbreiten zwischen den Routern zu 100 MBit/s und die Bandbreite zwischen *Router2* und *Router3* zu 80 MBit/s.

d. Der Kehrwert des mittleren Ankunftabstandes entspricht der Erzeugungsrate neuer Ströme.

Die Bearbeitungsgeschwindigkeiten zur Übernahme bzw. -gabe der Anwendungsdaten sind wie in Abschnitt 8.6 gewählt worden. Damit können Nutzdaten mit einer Geschwindigkeit von $2.68 * 10^8$ Bytes/s übernommen werden, während für die Übergabe eine Geschwindigkeit von $4.19 * 10^8$ Bytes/s erreichbar ist [60]. Die Bearbeitungsgeschwindigkeiten für die Informationsübernahme aus dem gemeinsamen Speicher ist mit 620 Zugriffen/s ebenso aus Abschnitt 8.6 übernommen worden wie die Bediengeschwindigkeit mit 200 PDUs/s zur Erstellung der PDUs.

Beschreibung	Parameter	Einheit
Bandbreite Server-Router	1250000	Bytes/s
Bandbreite Clients-Router	1250000	Bytes/s
Bandbreite Router1-Router2	12500000	Bytes/s
Bandbreite Router1-Router4	12500000	Bytes/s
Bandbreite Router3-Router4	12500000	Bytes/s
Bandbreite Router2-Router3	10000000	Bytes/s
Datenlesegeschwindigkeit	$2.68 * 10^8$	Bytes/s
Datenschreibegeschwindigkeit	$4.19 * 10^8$	Bytes/s
Zugriffsgeschwindigkeit SharedMemory	620	Zugriffe/s
Geschwindigkeit bei PDU-Erstellung	200	PDUs/s
Anzahl Prozessoren	4 - 8	-
Anzahl Datenbasen	3	-
Max. zulässige Auslastung pro Link	80	%
Max. zulässige Auslastung pro Prozessor	90	%
Max. zulässige Auslastung SharedMemory	90	%

Tabelle 9-2 **Performance-Parameter Audioverteildienst**

Auch die oberen Schranken für die noch zulässigen Auslastungen der aktiven Ressourcen haben einen Einfluß auf das Performance-Verhalten des Audioverteildienstes, da bei jedem neuen Verbindungsaufbau geprüft wird, ob diese Schranke mit dem neuen Strom überschritten würde. Somit ist die Anzahl der aktiven Verbindungen von den Werten dieser Schranken abhängig. In der vorliegenden Studie ist die maximal zulässige Auslastung für die Kanäle mit 80% festgelegt worden, während die Auslastungen der Ressourcen auf den Knoten einen Wert von 90% erreichen dürfen.

Das Gesamtsystem ist mit den unterschiedlichen Parametern jeweils für etwa 1000 Sekunden Modellzeit simuliert worden. Damit sind je nach Anzahl der zulässigen maximalen Anzahl von *ST2*$^+$-Strömen pro Simulationslauf zwischen 160 und 400 Verbindungen auf- und wieder abgebaut worden. Da im Mittel etwa 2000 Meßwerte pro Verbindung angefallen sind, ist ein ausreichender Stichprobenumfang erreicht, und die Konfidenzintervalle sind so klein, daß sie in den Graphiken von Abbildung 9-19 bis Abbildung 9-22 nicht eingezeichnet worden sind. So ergibt z.B. die Breite des 95%-Konfidenzintervalls der mittleren Verzögerung eines Frames bei vier Prozessoren und acht zugelassenen Strömen einen Wert von 0.013 Millisekunden bei einem Mittelwert von 34.02 Millisekunden. Die Größe aller anderen Konfidenzintervalle bezogen auf den Mittelwert verhält sich entsprechend.

In Abbildung 9-19 ist die mittlere sowie die maximale Verzögerung eines MPEG-Frames auf dem Weg vom Server zu den Clients im Netz1 und im Netz2 dargestellt. Diese Pakete erfahren eine Verzögerung durch vier Knoten und drei Links. Hingegen werden die Pakete in das Netz3, deren Verzögerungswerte in Abbildung 9-20 dargestellt sind, durch fünf Knoten und vier Links verzögert.

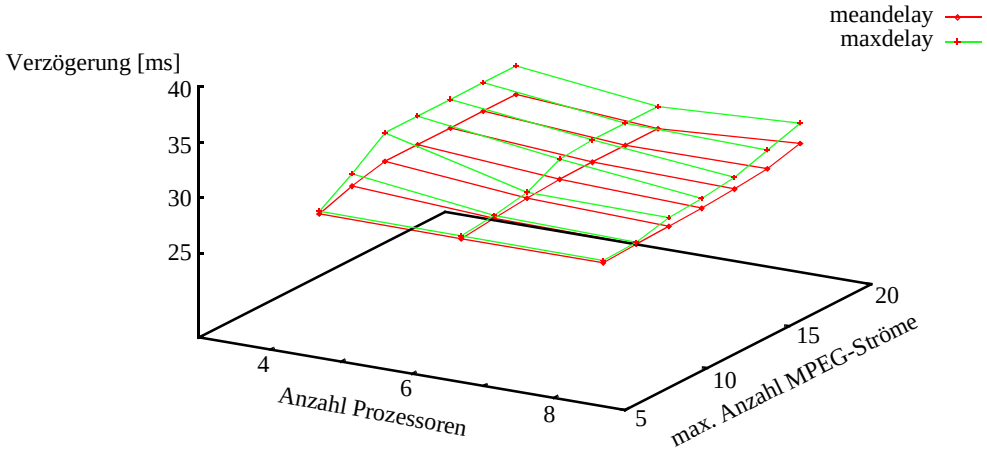


Abbildung 9-19 **Verzögerung MPEG-Frames durch 4 Knoten und 3 Kanäle**

In dieser ersten Experimentserie ist die Anzahl der Datenbasen im Shared-Memory-Bereich auf den Wert drei festgelegt worden. Die Anzahl der Prozessoren ist von vier über sechs auf acht erhöht worden. Als dritter Parameter ist die Anzahl der zulässigen MPEG-Ströme von acht auf bis zu zwanzig erhöht worden.

Dabei steigt sowohl die mittlere als auch die maximale Verzögerung bei Erhöhung der oberen Schranke für die Anzahl parallel existierender Ströme, siehe Tabelle 9-3.

# Proz.	# Ströme	Mittlere Verzögerung	Maximale Verzögerung	# Ströme	Mittlere Verzögerung	Maximale Verzögerung
4	8	27,42 ms	27,65 ms	12	29,12 ms	31,67 ms
6	8	27,35 ms	27,60 ms	16	28,22 ms	30,20 ms
8	8	27,35 ms	27,56 ms	20	29,06 ms	30,87 ms

Tabelle 9-3 **Verzögerung von MPEG-Frames durch 4 Knoten und 3 Kanäle**

So steigt die mittlere Verzögerung bei vier vorhandenen Prozessoren von 27.42 ms auf 29.12 ms, wenn die Anzahl zugelassener Ströme von acht auf zwölf erhöht wird, vgl. 1. Zeile in Tabelle 9-3. Die maximale Verzögerung erhöht sich dabei von 27.65 Millisekunden auf 31.67 Millisekunden. Bei acht vorhandenen Prozessoren steigt die mittlere Verzögerung von 27.35 Millisekunden auf 29.06 Millisekunden, wenn man die Anzahl der Ströme von acht auf 16 erhöht, während sich die maximale Verzögerung gleichzeitig von 27.56 Millisekunden auf 30.87 Millisekunden erhöht, vgl. 3. Zeile in Tabelle 9-3.

Der Vergleich der Ergebnisse bei einer oberen Schranke von zwölf zugelassenen Strömen[e] zeigt den geringen Einfluß, den die zusätzlich bereitgestellten Prozessoren auf die mittlere Verzögerung haben, vgl. Tabelle 9-4.

# Proz.	Mittlere Verzögerung	Absolute Verbesserung	Relative Verbesserung	Maximale Verzögerung	Absolute Verbesserung	Relative Verbesserung
4	29,12 ms	-		31,67 ms	-	
6	27,99 ms	1,12 ms	3,8 %	28,51 ms	3,16 ms	9,9 %
8	27,63 ms	0,36 ms	1,2 %	28,06ms	0,45 ms	1,5 %

Tabelle 9-4 **Verzögerung von MPEG-Frames durch 4 Knoten und 3 Kanäle bei konstant 12 zugelassenen Strömen**

Während bei der Erhöhung von vier auf sechs Prozessoren die mittlere Verzögerung von 29.12 Millisekunden auf 27.99 Millisekunden sinkt, sinkt sie bei einer weiteren Erhöhung auf acht auf lediglich 27.63 Millisekunden ab. Die Erhöhung der Prozessoren von vier auf sechs reduziert die mittlere Verzögerung um 3,8%, während eine Aufstockung um weitere zwei Prozessoren lediglich zu einer Reduktion um 1,2% führt.

Bei Betrachtung der maximalen Verzögerung ergibt sich ein anderes Bild. Die Erhöhung von vier auf sechs Prozessoren führt hier zu einer deutlichen Verbesserung von 3,16 Millisekunden, was einer Steigerung von 9,9% entspricht. Die maximale Verzögerung kann dann bei Einsatz weiterer Prozessoren um 0,45 Millisekunden und somit um 1,5% erhöht werden.

Die zusätzlichen Prozessoren sind aber in jedem Fall erforderlich, um die Erhöhung der Anzahl parallel zugelassener Ströme zu erreichen. Der Durchsatz ist konstant und erreicht unabhängig von der Anzahl der Prozessoren und der Anzahl paralleler Ströme einen Wert von 20,28 KBytes/s. Dieser Wert entspricht dem durch die Quelle erzeugten Ankunftstrom je MPEG-Verbindung und zeigt, daß in allen Knoten das Ressource Management den gewünschten Erfolg hat und den $ST2^+$ Strömen der ausgehandelte Durchsatz ermöglicht wird.

Ein vergleichbares Bild ergibt sich, wenn man die Verzögerungsergebnisse für den Fall einer Verzögerung durch fünf Knoten und vier Links betrachtet, siehe Abbildung 9-20.

e. Bei vier Prozessoren sind nicht mehr als zwölf Strome zugelassen. Als Werte für mehr als zwölf Ströme sind die Ergebnisse für zwölf Ströme übernommen worden, da das Graphik-Tool GNU-Plot diese Eingaben erfordert.

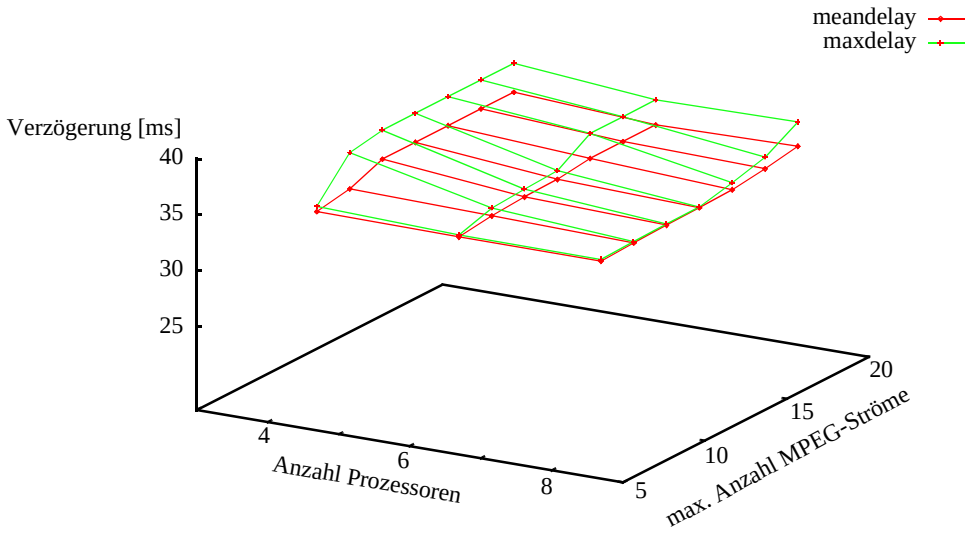


Abbildung 9-20 **Verzögerung MPEG-Frames durch 5 Knoten und 4 Kanäle**

Hier steigt die mittlere Verzögerung von 34,02 Millisekunden auf 35,82 Millisekunden, wenn man die Anzahl der zugelassenen Verbindungen bei vier Prozessoren von acht auf zwölf erhöht, siehe 1. Zeile Tabelle 9-5. Die maximale Verzögerung erhöht sich dabei von 34,58 Millisekunden auf 38,41 Millisekunden. Bei acht Prozessoren erhöhen sich die mittlere Verzögerung von 34,01 Millisekunden auf 35,31 Millisekunden und die maximale Verzögerung von 34,15 Millisekunden auf 37,47 Millisekunden, wenn man die Anzahl zugelassener Ströme von acht auf 20 erhöht, siehe Tabelle 9-5, Zeile 3.

# Proz.	# Ströme	Mittlere Verzögerung	Maximale Verzögerung	# Ströme	Mittlere Verzögerung	Maximale Verzögerung
4	8	34,02 ms	34,58 ms	12	35,82 ms	38,41 ms
6	8	34,01 ms	34,20 ms	16	35,05 ms	37,28 ms
8	8	34,01 ms	34,15 ms	20	35,31 ms	37,47 ms

Tabelle 9-5 **Verzögerung von MPEG-Frames durch 5 Knoten und 4 Kanäle**

Auch hier kann der geringe Einfluß, den die zusätzlich bereitgestellten Prozessoren auf die mittlere Verzögerung haben, nachgewiesen werden, siehe Tabelle 9-6.

# Proz.	Mittlere Verzögerung	Absolute Verbesserung	Relative Verbesserung	Maximale Verzögerung	Absolute Verbesserung	Relative Verbesserung
4	35,82 ms	-		38,41 ms	-	
6	34,59 ms	1,23 ms	3,4 %	35,30 ms	3,11 ms	8,1 %
8	34,23 ms	0,36 ms	1,04 %	34,32 ms	0,98 ms	2,7 %

Tabelle 9-6 **Verzögerung von MPEG-Frames durch 4 Knoten und 3 Kanäle bei konstant 12 zugelassenen Strömen**

Die Erhöhung von vier auf sechs Prozessoren führt zu einer Verbesserung der mittleren Verzögerung von 1,23 Millisekunden oder 3,4% und die weitere Erhöhung auf acht Prozessoren zu einer weiteren Verbesserung um 0,36 Millisekunden oder 1,04%. Deutlicher wird auch hier die maximale Verzögerung um 3,11 Millisekunden oder 8,1% bzw. 0,98 Millisekunden oder 2,7% verbessert.

Ein sehr überraschendes Ergebnis liefert die Analyse für die Laufzeitschwankungen (Jitter). Da die Laufzeitschwankungen mit der Anzahl der Verzögerungselemente nur unmerklich variieren, sind die Ergebnisse in Abbildung 9-21 und in Abbildung 9-22 für alle Clients gültig.

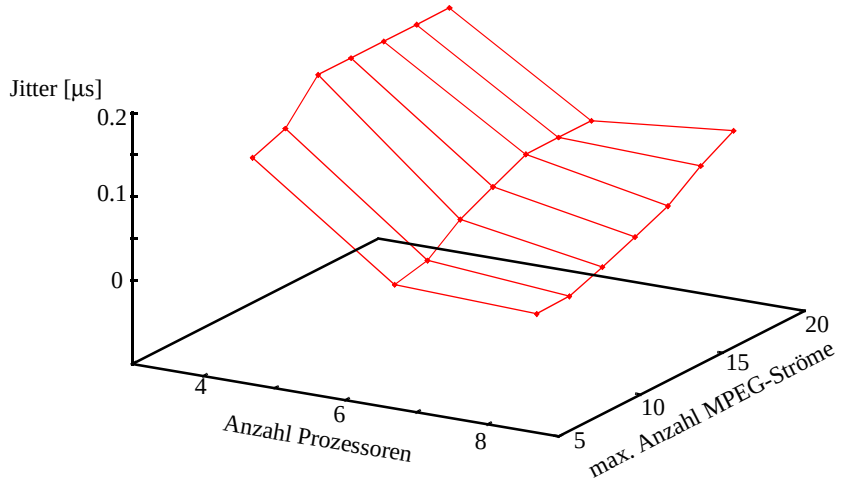


Abbildung 9-21 **Mittlere Laufzeitschwankungen (Jitter) von MPEG-Frames**

Die mittleren wie auch die maximalen Laufzeitschwanken liegen im Bereich von wenigen Microsekunden. Das bedeutet, daß auf die Einrichtung von Playout-Puffern vollständig verzichtet werden kann, da die Audio-Frames mit nahezu der gleichen Geschwindigkeit bei den Clients eintreffen, mit der sie vom Server eingespielt werden.

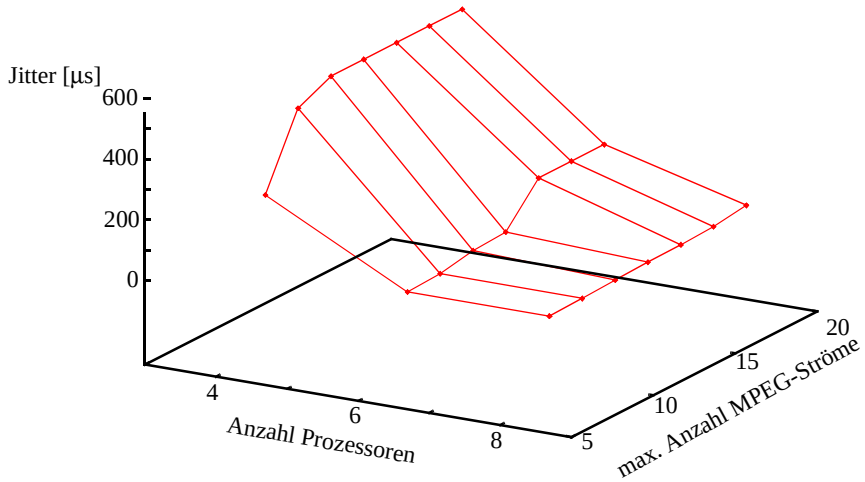


Abbildung 9-22 **Maximale Laufzeitschwankungen (Jitter) von MPEG-Frames**

Die Erhöhung der Prozessoren wirkt sich prozentual sehr deutlich auf die Reduktion der Laufzeitschwankungen aus. Da die Laufzeitschwankungen aber bereits bei vier Prozessoren unterhalb einer Millisekunde liegen, sind diese Verbesserungen durch das menschliche Gehör nicht wahrnehmbar.

So ergibt sich bei der Erhöhung der Prozessoren von vier auf sechs eine Reduktion des maximalen Jitters von 95%, und die Ergänzung zweier weiterer Prozessoren führt noch einmal zu einer Verbesserung von 66% auf einen Wert von lediglich 8,83 µs, siehe Tabelle 9-7.

# Proz.	Mittlerer Jitter	Relative Verbesserung	Maximaler Jitter	Relative Verbesserung
4	0,190 µs		520 µs	
6	0,046 µs	75,8%	26 µs	95 %
8	0,018 µs	60,8 %	8,8 µs	66 %

Tabelle 9-7 **Laufzeitschwankungen von MPEG-Frames**

Da die geringen Laufzeitschwankungen sehr überraschend sind, wurde die Plausibilität der Ergebnisse durch die Auswertung weiterer Sensoren geprüft. So konnten die Werte der Maschinenauslastungen durch die Anwendung analytischer Formeln verifiziert werden. Das zeigte, daß die komplexen Bedienstrategien FCFSPNP und PS bei mehreren Bedienern korrekt implementiert wurden und die Sensoren korrekte Werte liefern.

Da die Bedienwünsche eines *ST2*$^+$-Stromes für die Lebenszeit einer Verbindung konstant sind und auch der Abstand zweier aufeinanderfolgender MPEG-Frames als Kehrwert der Senderate konstant ist, wird die gegenseitige Beeinflussung von parallel aktiven MPEG-Strömen durch ihre Erzeugungszeitpunkte festgelegt. Die Verbindungsaufbauwünsche werden durch den Prozeß Lastgenerator gesteuert und folgen einer Exponentialverteilung, siehe Tabelle 9-1.

Die Bearbeitung der Verbindungsaufbauwünsche geschieht mit Hilfe des Kontrollprotokolls durch die Instanzen der Prozesse *SCMP* in den Knoten. Dazu konkurrieren diese Prozesse aber um dieselben Maschinen, die auch die etablierten *ST2*$^+$-Ströme unterstützen. Da die Bearbeitung der Requests durch die Prozesse *ST2_Con* priorisiert ist, werden die Verbindungsaufbauwünsche nur in den Pausen der Datenströme bearbeitet. Das führt letztlich dazu, daß neue Ströme überwiegend in die Lücken der bereits etablierten Ströme eingefügt werden und so die geringen Laufzeitschwankungen entstehen.

Unterstellt man die Korrektheit dieser These, müßten die Werte sowohl für die maximale Verzögerung der MPEG-Frames, als auch vor allem die Werte für die Laufzeitschwankungen steigen, wenn man die Variabilität der bisher konstanten Bedienwünsche je *ST2*$^+$-Strom steigert. Durch diese Steigerung kommt es vermehrt zu Wartesituationen, was die maximale Verzögerung erhöht, aber auch zu unterschiedlichen Wartesituationen innerhalb eines Stromes, was den Wert für den Jitter erhöht. Daher sind für einige Parameterkombinationen Kontrollsimulationen durchgeführt worden. Da die wesentliche Verzögerung innerhalb eines Knotens durch den Maschinendienst *compute_pdu* zustande kommt, wurde der konstante Bedienwunsch durch einen exponentialverteilten Bedienwunsch mit Rate $\mu = 1.0$ ersetzt. Dadurch ist der Variationskoeffizient von Null auf Eins erhöht worden.

Die in Tabelle 9-8 dargestellten Ergebnisse unterstreichen die Korrektheit der These, daß die geringen Laufzeitschwankungen durch die Etablierung neuer Ströme in den Lücken der bereits existierenden herrühren.

# Proz.	# Ströme	Min Verzögerung	Mittlere verzögerung	Max Verzögerung	Min Jitter	Mittlerer Jitter	Max Jitter
4	12	17,33 ms	29,42 ms	70,99 ms	2,3 µs	7,58 ms	49,42 ms
8	16	17,54 ms	27,83 ms	76,48 ms	4 ms	7,7 ms	42,2 ms

Tabelle 9-8 **Verzögerung und Laufzeitschwankungen bei variablen Bedienwünschen**

Während die mittleren Verzögerungen nahezu unverändert bleiben, erhöht sich nicht nur die maximale Verzögerung deutlich, sondern auch vor allem der Jitter-Wert. Die mittleren als auch die maximalen Laufzeitschwankungen erhöhen sich etwa um den Faktor 10^4 und erreichen dieselbe Größenordnung wie die Laufzeiten selbst.

Damit kann festgestellt werden, daß die entworfenen lokalen Ressource-Manager nicht nur eine wirksame Zugangskontrolle ermöglichen, sondern auch als lokale Verkehrsglätter fungieren, da sie den Start neuer Ströme gezielt in die Pausen der bereits etablierten Ströme verlegen.

Für eine vollständigere Sicht auf das entworfene Szenario könnten nun weitere Lastquellen an anderen Stellen in das System eingeführt werden, um die Auswirkungen weniger stark entzerrter Ströme auf das Gesamtsystemverhalten zu bewerten. Solche Experimente stellen aber keine prinzipiell neue Analysetechnik dar und sind daher im Rahmen dieser Arbeit nicht mehr durchgeführt worden.

Resümee zu den Experimenten

Die für die Leistungsbewertung in der SDL-Spezifikation zu ergänzenden Informationen betragen ca. 8,7% (in Lines of Code). Während das funktionale Modell, also das SDL-System aus 10875 Zeilen SDL/PR-Code besteht, umfaßt das quantitative Modell, also das QSDL-System 11822 Zeilen QSDL/PR-Code. Die gesamte Simulationszeit für alle durchgeführten Simulationen betrug etwa 48 Stunden auf einer SPARC Ultra3000. Allerdings zeigte sich bei der Größe und Komplexität des Gesamtsystems, daß auch die Werkzeuge zur Analyse des funktionalen Modells (SDT-Validator, MSC-Editor) an ihre Grenzen gestoßen sind. Die Suche nach semantischen Fehlern in einer solch umfangreichen SDL-Spezifikation hat sich als extrem schwierig herausgestellt.

Bei der quantitativen Analyse des Systems hat sich gezeigt, daß die QSDL-Sensoren nicht nur zur Erfassung der gewünschten Leistungsmaße, sondern auch zur Fehleranalyse eingesetzt werden können. So zeigte sich bei ersten Analysen, daß die von den Clients beobachtbare Verzögerung mit der Simulationsdauer stetig zunahm. Dies ist ein Hinweis darauf, daß sich an mindestens einer Station im System eine stetig zunehmende Warteschlange aufbaut. Da dieses Phänomen in jedem Client unabhängig von seiner Lokation zu beobachten war, mußte das Problem in dem Teil des Systems gesucht werden, der von allen Strömen benutzt wird. Die Auswertung der Maschinenauslastungen zeigte dann, daß die Maschine *ProcS* im Server zu 100% ausgelastet war. In der Spezifikation wurde die Anzahl der Bediener als Konstante mit dem Wert Eins definiert, statt das externe Synonym *NoServers* zu verwenden. Da der lokale Ressource Manager die Kapazität der Maschine anhand dieses Synonyms verwaltet hat und der Wert des Synonyms für den analysierten Simulationslauf vier war, hat der Manager eine Maschine mit vier Prozessoren verwaltet, obwohl die Maschine eine Einprozessormaschine war. Nach Behebung dieses Fehlers war eine ordnungsgemäße Ausführung des Systems möglich.

9.6 Zusammenfassung Kapitel 9

In diesem Kapitel ist ein großes Anwendungsszenario entwickelt und analysiert worden, das im Kern das von der IETF als Draft verabschiedete *ST2*$^{+}$-Protokoll enthält. Es wurde sowohl für das Kontroll- als auch für das Transferprotokoll aus den informellen Angaben des RFC 1819 eine formale SDL-Spezifikation entwickelt.

Diese Spezifikation ist als Grundlage verwendet worden, um die anwenderorientierten Leistungsmaße Ende-zu-Ende-Verzögerung, Laufzeitschwankungen (Jitter) und Durchsatz in einem Audio-Verteildienst zu untersuchen. Zusätzlich sind die benötigten Hilfsfunktionen für das Routing und das Ressource-Management entwickelt und ebenfalls in SDL spezifiziert worden.

Die im Rahmen dieser Arbeit entwickelten Lastbausteine für Realzeitströme sind ohne großen Aufwand in das Gesamtsystem integriert worden. Hier zeigt sich die ausgezeichnete Modularität der QUEST-Methode.

Die für eine Performance-Analyse notwendigen Ergänzungen wie Maschinendienste, Maschinen, Links und Requests sind ebenso in das SDL-System integriert worden wie die notwendigen Sensoren in den Anwendungsprozessen. Diese Sensoren erfassen die anwenderorientierten Leistungsmaße. Der zusätzliche Aufwand ist mit etwa 9% verhältnismäßig gering, so daß er parallel zum Systementwurf tragbar ist.

Die Analyse des Verteildienstes hat gezeigt, daß die Überlastabwehr und die ausreichende Unterstützung isochroner Audioströme durch den lokalen Ressource-Manager gewährleistet wird. Weiterhin führt die Kombination aus Ressource-Management und Priorisierung der Dateneinheiten an allen Maschinen zu einer Verkehrsglättung und so letztlich zu extrem niedrigen Laufzeitschwankungen.

Kapitel 10

Zusammenfassung

Neben der funktionalen Korrektheit spielt die Leistungsfähigkeit von komplexen, verteilten Software-Systemen eine zunehmend wichtigere Rolle. Systeme, die ihre Deadlines nicht einhalten, werden oft auch funktional als nicht korrekt angesehen. Daher ist die Integration von Performance-Analysen in den Software Engineering Prozeß notwendig. Dabei ist zu beachten, daß die Methoden, Werkzeuge und Vorgehensweisen mit den gängigen Techniken des Software Engineerings verträglich sind. Für den Entwurf von Protokollen hat sich die SDL-Methode als Quasi-Standard in der Telekommunikationsindustrie etabliert. Die Zusammenführung von Performance-Analysen mit der SDL-Methode war deshalb eine große Herausforderung zu Beginn der Forschungsarbeiten zu dieser Schrift. Die erzielten Ergebnisse lassen sich wie folgt zusammenfassen:

1. Im SDL-basierten Entwurf von verteilten Kommunikationsprotokollen werden nur logisch funktionale Aspekte betrachtet. Daher fehlen in SDL Sprachelemente zur ausreichenden Formulierung von Performance-Aspekten. Als erster Schritt zur Integration von Performance-Analysen wurde gemeinsam mit Marc Diefenbruch [38] eine Beschreibungssprache entwickelt, mit deren Hilfe die notwendigen Informationen in den SDL-basierten Entwurf verteilter reaktiver Systeme integriert werden können. Im Gegensatz zu den bisher in der Literatur vorgestellten Ansätzen ermöglicht der QSDL-Ansatz die Betrachtung von Konkurrenzsituationen an begrenzten Ressourcen. Darüberhinaus wird durch die Trennung von Last und Maschine sowie die Definition eines Bindungsmechanismus ein flexibles und allgemeingültiges Beschreibungsinstrument für quantitative Modelle bereitgestellt. Durch die Anwendung der Sprache und das Werkzeug QUEST ist die Sprache QSDL im Rahmen dieser Arbeit um das mutex-Konzept erweitert worden. Dieses Konstrukt ist ursprünglich nicht im Sprachumfang enthalten gewesen. Die Notwendigkeit dieses ergänzenden Synchronisationsmechanismus ist bei der Entwicklung der Performance-Modelle für die unterschiedlichen Implementierungsstrategien in Abschnitt 4.2 dargestellt worden.
2. Die Definition einer Sprache sowie die Implementierung eines Analysewerkzeugs ist eine notwendige Voraussetzung, um Performance-Analysen in den Software Engineering Prozeß zu integrieren. Für die Anwendung in der praktischen Entwicklung ist eine

durchgängige Methode bzw. ein Vorgehensmodell notwendig, das dem System-Engineer zeigt, wie er Sprache und Werkzeug in seiner täglichen Arbeit einsetzen kann. In dieser Arbeit ist für die Sprache QSDL eine Methodik entwickelt und demonstriert worden, die eine Unterstützung des Implementierungs-Designs durch Leistungsbewertungstechniken ermöglicht. Dazu sind für die gängigen Implementierungsansätze und für die am weitesten verbreiteten Schnittstellen Performance-Bausteine erstellt worden, die es einem Entwickler erlauben, Design- und Architekturentscheidungen bezüglich ihres Leistungsverhaltens **vor** einer vollständigen Implementierung modellgestützt zu bewerten.

So sind Bausteine für das *Einprozeß-Server-Modell*, das *Mehrprozeß-Server-Modell* und das *Activity-Thread-Modell* jeweils für die Implementierungsvarianten mit *Prozessen*, mit *Threads* oder mit *Prozeduren* erstellt worden. Ergänzend dazu sind Modellbausteine für *puffernde* und für *aktionsorientierte* Schnittstellen erstellt worden.

Für die implizit vorhandenen Parallelitäten in Kommunikationsprotokollen sind Richtlinien für Entwurfstechniken erarbeitet und Performance-Bausteine für die vorgeschlagenen Techniken entworfen worden. So sind solche Bausteine für *schichtenbezogene*, *nachrichtenbezogene*, *funktionsbezogene*, *richtungsbezogene* und *verbindungsbezogene* Parallelität entwickelt und vorgestellt worden.

Somit steht den Protokoll-Designern eine reiche Auswahl von Performance-Bausteinen für die quantitative Bewertung von Protokollen zur Verfügung. Darüberhinaus sind Analysetechniken für das Werkzeug QUEST beschrieben worden, mit deren Hilfe sowohl das kurzzeitige als auch das langfristige zeitliche Verhalten von Kommunikationsprotokollen ermittelt werden können. So ist der Einsatz des Werkzeugs QUEST zur Analyse von *Ursache-Wirkungsbeziehungen* ebenso demonstriert worden, wie der Einsatz zur *Engpaßanalyse*.

Darüberhinaus wurde die *transiente* und *stationäre* Durchsatzanalyse eines Kommunikationsprotokolls beschrieben, bei der beide Analysen mit Hilfe **eines** Modells und in **einem** Analyselauf durchgeführt werden konnten.

Die bei der Entwicklung der Methodik gemachten Erfahrungen bzw. Vorschläge zum Einsatz der entwickelten Performance-Komponenten zu einem Gesamtsystem sollen hier kurz zusammengefaßt werden.

(a) Bereits beim Entwurf des SDL-Systems, das die Zerlegung eines komplexen Gesamtsystems in funktionale Einheiten beschreibt, ist darauf zu achten, daß nicht schon an dieser Stelle Festlegungen getroffen werden, die das Performance-Verhalten beeinflussen können. Alle Aktionen, die potentiell parallel ablaufen sollen, müssen in separaten SDL-Prozessen spezifiziert werden, da andernfalls durch die SDL-Semantik eine Serialisierung dieser Aktionen vorgenommen wird.

(b) Wenn das funktionale Design abgeschlossen[a] ist, d.h. das System ist vollständig spezifiziert und es befinden sich keine logischen Fehler mehr in der Spezifikation, können erste Performance-Informationen ergänzt werden. Im ersten Schritt ist die Hardware-Architektur des Zielsystems zu berücksichtigen. Dazu sind die Art und die Anzahl der zur Verfügung stehenden aktiven Ressourcen wie Prozessoren oder Kanäle durch entsprechende QSDL-Maschinen zu beschreiben. Dabei werden die

a. Das zu untersuchende System muß nicht vollständig bis in das kleinste Detail ausspezifiziert sein. Es können auch erste Grobentwürfe analysiert werden. Allerdings muß das System soweit festgelegt sein, daß die SDL-Prozesse Transitionen ausführen und dabei Signale versenden, damit das Gesamtsystem ablaufen kann.

Funktionsweise und die Kapazität der realen Komponenten durch Schedulingstrategien und Bediengeschwindigkeiten von QSDL-Maschinen und Maschinendienste repräsentiert. Die funktionale Zerlegung des Gesamtsystems muß nun auf die physikalische Struktur der unterliegenden Hardware abgebildet werden. Dabei wird jeder SDL-Prozeß, der in die Performance-Analyse mit eingebunden werden soll, über QSDL-Links und -Pipes an die QSDL-Maschinen gebunden. Damit liegt die Implementierungsarchitektur fest.

(c) Schließlich müssen noch innerhalb der SDL-Prozesse die Aktionen identifiziert werden, deren Performance-Verhalten in das Modell mit aufgenommen werden sollen. Dabei werden die durch das Implementierungsmodell festgelegten Verarbeitungsaufwände der Aktionen als Bedienwünsche innerhalb der QSDL-Requests beschrieben.

(d) Die durch das zu untersuchende System zu bewältigende Arbeitslast kann durch separate Lastprozesse spezifiziert werden. Dabei ist darauf zu achten, daß die Schnittstellen zwischen Lastprozessen und System gegebenenfalls anzupassen sind, d.h. Signalnamen, Signalparameter und Signalrouten sind so zu gestalten, daß die SDL-Signale zwischen beiden Teilsystemen transferiert werden können. Gleichzeitig werden mit der Spezifikation der Lastprozesse die Simulationsszenarien festgelegt und das Verhalten dieser Prozesse entsprechend abgestimmt.

Falls zur Unterstützung des Systems weitere unterliegende Dienste notwendig sind, so sind Performance-Bausteine für diese Dienste zu entwickeln, bzw. vorhandene anzupassen und entsprechend des Vorgehens bei den Lastprozessen in ein Gesamt-Performance-Modell zu integrieren. Auch hier muß im wesentlichen dafür gesorgt werden, daß die Schnittstellen zwischen den Teilsystemen konsistent sind.

(e) Nun kann das Gesamtsystem analysiert und die gewünschten Leistungsmaße über Sensoren ermittelt werden. Falls die erreichbare Performance des Systems nicht ausreicht, kann entweder über leistungsstärkere Ressourcen oder durch die Bereitstellung weiterer Ressourcen die Performance verbessert werden. Bei sehr großen Abweichungen von den gewünschten Zielgrößen muß gegebenenfalls erneut in den Design-Prozeß zurückgekehrt werden, um dann mit einem neuen Entwurf erneut in den Analysezyklus einzusteigen

3. Die Tragfähigkeit und Anwendbarkeit der entwickelten Methodik ist für den Entwurfsprozeß eines Multimedia-Transportsystems demonstriert worden. Dabei wurden zunächst notwendige vorbereitende Arbeiten durchgeführt, die zur Erstellung eines homogenen Performance-Modells notwendig sind.

   (a) Um zu einer umfassenden Bewertung eines Transportsystems zu gelangen, müssen die Belastungen definiert werden, mit denen ein solches System konfrontiert wird. Daher sind sowohl für diskrete Ströme wie *FTP, Telnet* oder *WWW* als auch für kontinuierliche Ströme wie *MPEG-Video*, *MPEG-Audio* oder *digitalisierte Sprache* QSDL-Lastbausteine entwickelt worden. Diese Lastbausteine konnten in den diversen Studien im Rahmen dieser Arbeit immer wieder verwendet werden, was die Modularität des entwickelten Ansatzes demonstriert.

   Darüberhinaus konnte durch die Analyse rückgekoppelter Sprachquellen der Einfluß der zeitlichen Verzögerung der Rückkopplung auf das zeitliche Verhalten von zwischenmenschlicher Kommunikation gezeigt werden. Dadurch war es möglich, verbesserte Modelle für diese Kommunikation zu erstellen.

(b) Neben der Definition des Lastverhaltens ist eine quantitative Festlegung der Dienste vorzunehmen, auf die sich das zu untersuchende Transportsystem abstützt. Daher sind für wichtige Protokolle der ISO-OSI-Schicht-2 (link layer) SDL-basierte Performance-Modelle entwickelt und analysiert worden. Somit konnten günstige und ungünstige Betriebsparameter für diese Protokolle bestimmt werden.

Für die Schwierigkeiten, hardwarenahe Protokolle mit Hilfe von SDL bzw. QSDL zu beschreiben, sind Lösungen erarbeitet und vorgestellt worden. Der Vergleich der Ergebnisse der QSDL-Modelle mit den Ergebnissen von klassischen Performance-Modellen zeigt, daß die mit Hilfe der QUEST-Methode erzielbaren Resultate mit denen übereinstimmen, die durch die Analyse eigenständiger und nur für die Performance-Analyse entwickelter Modelle erzielt werden. Somit läßt sich festhalten, daß die Ergebnisqualität nicht schlechter ist, als die der klassischen „handgemachten" Modelle. Durch die Verwendung SDL-naher Beschreibungselemente mit anschließender automatisierter werkzeuggestützter Generierung von Simulationsmodellen sind weniger spezielle Kenntnisse im Bereich Performance-Modellierung notwendig, so daß die Durchführung dieser Studien auch durch Nicht-Performance-Experten durchgeführt werden können.

4. Die informellen Angaben des IETF-Drafts RFC1819 über den Kontroll- und den Datentransferteil des *Internet Stream Protocols* sind in eine SDL-Spezifikation umgesetzt worden. Die durch das Protokoll vorausgesetzten Ressource-Management-Funktionen sind für *Links* und *aktive Knoten* im Rahmen dieser Arbeit entwickelt und bewertet worden. Die Eignung dieser Funktionen für den Einsatz wurde in einem komplexen Gesamtsystem mit Hilfe des QUEST-Ansatzes untersucht und nachgewiesen. Dabei hat sich gezeigt, daß die entwickelten Mechanismen nicht nur zu einer bedarfsgerechten Verwaltung der vorhandenen Ressourcen, sondern auch zu einer lokalen Verkehrsglättung führen, die die Einrichtung großer Playout-Puffer unnötig macht.

   Weiterhin hat sich herausgestellt, daß auch die makroskopischen Betrachtungen globaler Auswirkungen in komplexen Gesamtsystemen mit Hilfe der QUEST-Methode möglich sind. Die vorhandenen Analysemöglichkeiten erlauben eine performance-bedingte Fehlersuche und ergänzen so in idealer Weise die Methoden und Werkzeuge im Software-Engineering Prozeß für verteilte reaktive Systeme.

Somit schließt diese Arbeit die bisher bestehende Lücke im SDL-basierten Entwurfsprozeß verteilter reaktiver Systeme und ermöglicht die modellgestützte Betrachtung von Performance-Aspekten in den frühen Phasen des Entwurfsprozesses.

Im Gegensatz zu den bisher in der Literatur vorgestellten Ansätzen zur Kombination von Techniken zur Leistungsbewertung mit der SDL-Methode ist die Eignung der QUEST-Methode an komplexen Realweltproblemen demonstriert worden.

Kapitel 11

Ausblick

Im Rahmen dieser Arbeit ist eine Methodik zur Integration einer modellgestützten Leistungsbewertung in den SDL-basierten Entwurfsprozeß von Multimedia-Protokollen entwickelt und an Beispielen demonstriert worden. Obwohl die für diese Arbeit gesteckten Ziele erreicht worden sind, lassen sich eine Reihe von fortsetzenden Arbeiten formulieren.

- Die entwickelten Konzepte und die Sprache QSDL ist als Erweiterung der Sprache SDL entwickelt worden. Dabei sind eine Reihe von zwischenzeitlich in SDL eingeführte Elemente wie z.B. die objektorientierten Erweiterungen bisher nicht berücksichtigt. Der Annotationsmechanismus von QUEST und die Sprache QSDL sind als Vorschläge für die Z.108 *SDL and Performance* eingereicht worden. In diesem Rahmen könnte die Sprache QSDL auf Vollständigkeit geprüft werden. So hat sich zum Beispiel bei der Entwicklung der QUEST-Methodik in Kapitel 4 gezeigt, daß über die bisher in QSDL existierenden Mechanismen weitere Elemente zur Prozeßsynchronisation notwendig sind, um die Performance-Auswirkungen von Implementierungstechniken zu beschreiben. Darüberhinaus sind z.B. die Konzepte der QSDL-Maschinen auf SDL-Blocktypen zu erweitern, um so für den gesamten Sprachumfang von SDL'96 Entsprechungen in QSDL zu erhalten. In diesem Rahmen könnte auch direkt über die Integration von Zeit- und Performance-Aspekten in SDL 2000 nachgedacht werden.
- Die bisher entwickelten QSDL-basierten Performance-Modelle sind mit Hilfe von „handgemachten" simulativen Performance-Modellen validiert worden. Dabei hat sich gezeigt, daß die QUEST-Methode geeignet ist, dieselben Aussagen zu erzielen, die auch mit einer klassischen Performance-Studie erzielbar sind. In weiteren Arbeiten könnte anhand eines realen Implementierungsprojektes versucht werden, die Performance-Aussagen, die mit Hilfe des QUEST-Ansatzes erzielt werden, später anhand des implementierten System zu verifizieren. Dabei kann dann vor allem untersucht werden, wie und mit welchem Aufwand die Eingangsparameter für die Bedienwünsche von Requests ermittelt werden können. Auf diese Art und Weise läßt sich auch eine Art Datenbank ernstellen, in der für die immer wieder vorkommenden Aktionen innerhalb einer Protokollimplementierung typische Eingangsparameter für QSDL-Modelle ermittelt werden. Zu diesen Aktionen zählen die Übernahme bzw. die Übergabe von Daten, die verwaltung von Timern oder die Manipulierung von Daten innerhalb von SDL-Tasks. Somit

ergeht eine Aufforderung an alle Systementwickler, die mit SDL reaktive verteilte Systeme entwickeln, die QUEST-Methode im Alltag anzuwenden und so zu einer Weiterentwicklung beizutragen.

- Die QUEST-Methode ist im Kontext von Kommunikationsprotokollen entwickelt und angewendet worden. Daher könnte im folgenden versucht werden, diese Methode auch in anderen Bereichen zu etablieren. So wäre der Einsatz im Bereich *Consumer Electronics*, im Bereich *Automobilkommunikation, Mobilfunk* oder auch *Automatisierungstechnik* zu überprüfen. Hierbei könnte vor allem auf Probleme harter Realzeitanforderungen im Zusammenhang mit fehlerfreier Übertragung von Informationen eingegangen werden. Erste Ansätze hierzu werden von Patrik Kessler im Rahmen seiner Dissertation an der Universität GH Essen entwickelt. An der Universität von Athen wird von Georgios Nikolaidis und seiner Arbeitsgruppe der Einsatz der QUEST-Methode bei der Entwicklung des UMTS-Standards erprobt [2].
- Die kombinierte Anwendung der QUEST-Methode mit Methoden des spezifikationsgesteuerten Monitorings oder der Einsatz der Konzepte und Methoden im Umfeld von Performance-Testing stellt ebenfalls einen interessanten Ansatz zu weiteren Forschungsarbeiten dar.
- Im Bereich der Multimedia-Protokolle wäre die vergleichende Analyse unterschiedlicher Protokolle möglich. So ist ein Vergleich der Internet-Protokolle *ST2*$^{+}$ und *RSVP* auf der Grundlage einer SDL-Spezifikation sicherlich interessant. Dabei könnten insbesondere auch Aspekte der Implementierbarkeit auch in großen Systemen untersucht werden.
- Im Rahmen dieser Arbeit ist anhand mehrerer Beispiele demonstriert worden, wie die Beschreibung des Performance-Verhaltens unterliegender Medien durchgeführt werden kann. So kann insbesondere im Rahmen von Diplomarbeiten versucht werden, eine vollständige Bibliothek von Medien-, aber auch von Lastbausteinen zu entwickeln und für die Verwendung innerhalb des QUEST-Tools bereitzustellen.
- Als weiterer methodischer Forschungsschwerpunkt wäre zu untersuchen inwieweit sich die QUEST-Methodik zur Analyse vollständiger Protokoll-Stacks eignet. So könnten die Ergebnisse einer Analyse eines Schicht-I-Protokolls so aggregiert werden, daß sie als vereinfachter I-Dienst angeboten wird. Dann müßte man bei der Analyse eines Schicht-I+1-Protokolls nicht die vollständige Spezifikation der I-Schicht mit in das Modell aufnehmen, sondern könnte einen Medienprozeß spezifizieren, der den Aufruf des I-Dienstes in seinem zeitlichen Verhalten durch einen Request modelliert. So könnte sukzessive ein vollständiger Protokoll-Stack in seinem Performance-Verhalten analysiert werden.

Die Liste von fortsetzenden Arbeiten zeigt, daß mit der Entwicklung der QUEST-Methode ein Ansatz gelungen ist, der die bisher grundsätzlich vorhandenen Probleme einer entwurfsbegleitenden Perfomance-Analyse löst, so daß sich ein weites Feld für weitere Forschungsarbeit ergibt.

Kapitel 12

Literatur

[1] D. Anderson et.al: *A Session Reservation Protocol for Guaranteed-Performance Communication in the Internet.* TR-90-006, ICSI Berkeley, Februar 1990.

[2] A. Alonistioti, G. Nikolaidis und I. Modeas: SDL-based Modelling and Design of IN/ UMTS Handover Functionality; in: A. Cavalli and A. Sarma (Ed.) SDL'97 Time for Testing SDL,MSC and Trends. Tagungsband des 8. SDL-Forums, Evry 1997. Elsevier, Amsterdam 1997.

[3] ANSI87: American National Standards Institute. *FDDI Token Ring Media Access Control-2 (MAC-2).* ANSI X3T9/92-120, 1992.

[4] ANSI88: American National Standards Institute. *FDDI Physical Layer Protocol (PHY).* ANSI X3.148, 1988.

[5] ANSI90: American National Standards Institute. *FDDI Physical Layer Medium Dependent (PMD).* ANSI X3.166, 1990.

[6] ANSI93: American National Standards Institute. *FDDI Station Management (SMT).* ANSI X3T9.5, Review 7.2, 1993.

[7] A. Alonistioti et.al: *SDL-based Modelling and Design of IN/UMTS Handover Functionality*; in: A. Cavalli and A. Sarma (Ed.) SDL'97 Time for Testing SDL,MSC and Trends. Tagungsband des 8. SDL-Forums, Evry 1997. Elsevier, Amsterdam 1997.

[8] A. Badach, E. Hoffmann und O. Knauer: High Speed Internetworking. Grundlagen und Konzepte für den Einsatz von FDDI und ATM. Addison-Wesley, Bonn 1994.

[9] A. Banerjea, B. Mah: *The Real-time Channel Administration Protocol.* 2nd International Workshop on Network and Operating System Support for Digital Audio and Video, heidelberg, November 1991.

[10] B. Baumgarten: *Petri-Netze; Grundlagen und Anwendungen*. BI Wissenschaftsverlag, Mannheim 1990.

[11] F. Bause, P. Buchholz: *Qualitative and Quantitative Analysis of Timed SDL Specifications*; in: N. Gerner et.al. (Ed.) Kommunikation in verteilten Systemen, Informatik Aktuell. Springer-Verlag, Berlin 1993.

[12] F. Bause et.al: *SDL and Petri Net Performance Analysis of Communicating Systems*; in: 15. International Symposium on Protocol Specification, Testing and Verification. Warschau, Juni 1995.

[13] H. Beilner: *Workload Characterization and Performance Modelling Tools*; in: G.Serazzi (Eds.) Workload Characterization of Computer Systems ans Computer Networks. North-Holland, Amsterdam 1986.

[14] H. Beilner et al: *Towards a Performance Modelling Environment: News on HIT*; in 4th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation, Palma de Mallorca, 1988.

[15] F. Belina, D. Hogrefe, A. Sarma: *SDL with Application from Protocol Specification*. Prentice Hall, Hertfordshire 1991.

[16] U. Black: OSI, A Model for Computer Communications Standards. Prentice Hall, Englewood Cliffs, New Jersey 1991.

[17] S. Böhmer, R. Klafka: *A new Approach to Performance Evaluation of Formally Specified Protocols*. 7th International Conference on Formal Description Techniques (FORTE'94), Bern, 1994.

[18] G. v. Bochmann, J. Vaucher: *Adding Performance Aspects to Specification Languages*. Proc. 8th Workshop on Protocol Specification, Testing and Verification, Atlantic City. North-Holland, Amsterdam 1988.

[19] R. Braden, D. Clark, S. Shenker: *Integrated Services in the Internet Architecture: an Overview*. RFC 1633, Juni 1994.

[20] R. Braden, L. Zhang, S. Berson, S. Herzog, S. Jasmin: *Resource ReServation Protocol (RSVP)- Version 1; Functional Specification.* Internet Draft Oktober 1996.

[21] P. Brady: A *Technique for Investigating On-Off Patterns of Speech* in: The Bell System Technical Journal Volume XLIV, Nr.1, Januar 1965.

[22] P. Brady: *A Model for Generating On-Off Speech Patterns in Two-Way Conversation* in: The Bell System Technical Journal Volume XLVIII, Nr. 9, September 1969.

[23] K. Brandenburg et. al: *The ISO/MPEG-Audio Codec: A Generic Standard for Coding of High Quality Didital Audio* in: Journal of Audio Engineering Society, Vol. 42, Nr. 10, Oktober 1994.

[24] M. Bütow, M. Mestern, C. Schapiro, P.S. Kritzinger: *Performance Modelling with the Formal Specification Language SDL* in: R. Gotzhein und J. Bredereke (Ed.) Formal Description Techniques IX; Theory, application and tools. Tagungsband IFIP TC6 / 6.1 International Conference on Formal Description Techniques IX / Protocol Specification, Testing and Verification XVI, Kaiserslautern, 1996. Chapman and Hall, London 1996.

[25] I. Busse, B. Deffner, H. Schulzrinne: *Dynamic QoS Control of Multimedia Applications based on RTP;* In: First International Workshop on High Speed Networks and Open Distributed Platforms. St. Petersburg, Russland, Juni 1995.

[26] C. Cassandras: *Discrete Event Systems Modeling and Performance Evaluation*. Aksen Associates Incorporated Publishers, Boston 1993.

[27] COMNETIII: *User's Manual Release 1.4 Planning for Network Managers.* CACI Products Company, Camberly 1997.

[28] A. Conway: *Performance Modelling of Multi-Layered OSI Communication Architectures*. In: IEEE International Conference on Communications, Juni 1989.

[29] P. Csurgay und F.A. Aegesen: *A Distributed SDL-based Intelligent Network Laboratory*; in: R. Braek and A. Sarma (Ed.) SDL'95 with MSC in CASE. Tagungsband des 7. SDL-Forums, Oslo 1995. Elsevier, Amsterdam 1995.

[30] C. Cunha et.al: Characteristics of WWW Client-based Traces: Technischer Bericht, Boston University, Boston, MA, 1995.

[31] A. Danthine (Ed.): *The OSI 95 Transport Service with Multimedia Support*. Research Report ESPRIT Projekt 5341, Springer-Verlag, Berlin 1995.

[32] P. Danzig und S. Jamin: *tcplib: A library of TCP Internetwork Traffic Characteristics*; Report CS-SYS-91-01, Computer Sciences Department, Universität Süd Kalifornien 1991. Verfügbar über World Wide Web: http://netweb.usc.edu/jamin/tcplib/.

[33] L. Delgrossi: *Design of Reservation Protocols for Multimedia Communication*. Kluwer Academic Publishers, Dordrecht 1996.

[34] L. Delgrossi und L. Berger: *RFC 1819 Internet Stream Protocol Version2 (ST 2)*. 1995.

[35] M. Diefenbruch, E. Heck, J. Hintelmann, B. Müller-Clostermann: *Performance Evaluation of SDL systems adjunct by queueing models*; in: R. Braek and A. Sarma (Ed.) SDL'95 with MSC in CASE. Tagungsband des 7. SDL-Forums, Oslo 1995. Elsevier, Amsterdam 1995.

[36] M. Diefenbruch, J. Hintelmann, B. Müller-Clostermann: *The QUEST-Approach for the Performance Evaluation of SDL-Systems*; in: R. Gotzhein und J. Bredereke (Ed.) Formal Description Techniques IX Theory, application and tools. Tagungsband IFIP TC6 / 6.1 International Conference on Formal Description Techniques IX / Protocol Specification, Testing and Verification XVI, Kaiserslautern, 1996. Chapman and Hall, London 1996.

[37] M. Diefenbruch: *Funktionale und quantitative Verifikation von zeit- und ressourcenerweiterten SDL-Systemen mittels Model-Checking*; in: K.Irmscher et.al. (Ed.) MMB'97 Messung, Modellierung und bewertung von Rechen und Kommunikationssystemen. Tagungsband der 9. ITG/GI-Fachtagung, Freiberg, 1997. VDE-Verlag, Berlin 1997.

[38] M. Diefenbruch: *Quantitative Analyse zeit- und ressourcenerweiterter SDL-Systeme mit Hilfe von Zustandsraumexploration und Model-Checking*. Dissertation, Universität Essen, 1998.

[39] Ehrig, Mahr: *Fundamentals of Algebraic Specification*. Springer-Verlag, Bonn 1985.

[40] J. Ellsberger, D. Hogrefe, A. Sarma: *SDL Formal Object-oriented Language for Communicating Systems*. Prentice Hall, Hertfordshire 1997.

[41] K. Fall, S. Floyd: *Simulation-based Comparison of Tahoe, Reno and SACK TCP*. Computer Communication Review Vol. 26, July 1996 p. 5 - 21.

[42] D. Ferrari, G. Serazzi, A. Zeigner: *Measurement and Tuning of Computer Systems*; Prentice Hall, Englewood Cliffs, 1983.

[43] D. Ferrari: *Considerations on the Insularity of Performance Evaluation*. IEEE Trans. on Softw. Eng., Vol SE-12, No. 6, June 1986.

[44] D. Ferrari: *Real-Time Communication in an Internetwork*. TR-92-001. International Computer Science Institute, Berkeley, Januar 1992.

[45] D. Ferrari: *The Tenet Experience and the Design of Protocols for Integrated Services Internetworks.* Multimedia Systems, June 1996.

[46] A. Ferrero: *The evolving ethernet*. Addison-Wesley, Bonn 1996.

[47] R. Fielding, U. Irvine, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee: *RFC 2068 Hypertext Transfer Protocol - HTTP/1.1*. 1997.

[48] J. Forgie: *ST: A proposed Internet Stream Protocol*. IEN 119, September 1979.

[49] E. Gelenbe, G. Pujolle: *Introduction to Queueing Networks*. Wiley and Sons, New York 1987.

[50] F. Goudenove, L.Doldi: *Use of SDL to specify Airbus Future Air Navigation Systems*; in: R. Braek and A. Sarma (Ed.) SDL'95 with MSC in CASE. Tagungsband des 7. SDL-Forums, Oslo 1995. Elsevier, Amsterdam 1995.

[51] R. Händel, M. Huberl: *Integrated Broadband Networks - An Introduction to ATM-based Networks*. Addison-Wesley, Bonn 1991.

[52] E. Heck: *On a Combination of SDL and HIT*; in: K.Kronlöf (Ed.). Method Integration: Concepts and Case Study. Wiley and Sons, New York 1992.

[53] E. Heck: *Performance Evaluation of Formally Specified Systems -The Integration of SDL with HIT*. Krehl-Verlag, Münster 1997.

[54] E. Heck, D. Hogrefe, B. Müller-Clostermann: *Hierarchical Performance Evaluation Based on Formally Specified Communication Protocols*; in: IEEE Trans. on Computers, Vol. 40, Nr. 4, 1991 (Special Issue on Protocol Engineering).
[55] B. Heinrichs: *Multimedia im Netz*. Springer-Verlag, Berlin 1996.
[56] R. Henke, H. König, A. Mitschele-Thiel: *Derivation of Efficient Implementations from SDL Specifications Employing Data Referencing, Integrated Packet Framing and Activity Threads*. in: A. Cavalli and A. Sarma (Ed.) SDL'97 Time for Testing SDL,MSC and Trends. Tagungsband des 8. SDL-Forums, Evry 1997. Elsevier, Amsterdam 1997.
[57] R. Henke, A. Mitschele-Thiel, H. König, P. Langendörfer: *Automated Derivation of Efficient Implementations from SDL-Specifications*; Technischer Bericht 11/96 IMMD VII, Universität Erlangen-Nürnberg, November 1996.
[58] J. Hennessy, D. Patterson: *Rechnerarchitektur, Analyse, Entwurf, Implementierung, Bewertung*; Vieweg Verlag, Braunschweig 1994.
[59] J. Hintelmann: *Integration of SDL based QoS-evaluation in protocol design*; in: A. Javor et.al. (Ed.) Modelling and Simulation 1996. Tagungsband European Simulation Multiconference 1996. Society for Computer Simulation International, Istanbul 1996.
[60] J. Hintelmann, R. Westerfeld: Performance A*nalysis of TCP's Flow Control Mechanisms using Queueing SDL*; in: A. Cavalli and A. Sarma (Ed.) SDL'97 Time for Testing SDL,MSC and Trends. Tagungsband des 8. SDL-Forums, Evry 1997. Elsevier, Amsterdam 1997.
[61] A. Hirche: *Implementation eines Model-Checking-Algorithmus für das Tool QUEST*. Diplomarbeit Universität Essen, 1997.
[62] D. Hogrefe: *Estelle, LOTOS und SDL; Standard Spezifikationssprachen für verteilte Systeme*; Springer-Verlag, Berlin 1989.
[63] D. Hogrefe, S. Zorn: Simulation of SDL specified Models; in: Computer Networks and Simulation, 3, Seite 103-121; North Holland, 1986.
[64] IEEE 802.2: *IEEE Standards for Local Area Networks: Logical Link Control*.
[65] IEEE 802.3: *IEEE Standards for Local Area Networks: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specification*.
[66] IEEE 802.4: *IEEE Standards for Local Area Networks: Token-Passing Bus Access Method and Physical Layer Specification*.
[67] IEEE 802.5: *IEEE Standards for Local Area Networks: Token-Passing Ring Access Method and Physical Layer Specification*.
[68] IETF: http://diffserv.lcs.mit.edu
[69] ISO IEC JTC1: Information Technology - Coding of Moving Pictures and Associated Audio for Digital Storage Media up to about 1.5 Mbit/s; International Standard ISO/IEC IS 11172, 1993
[70] ISO TC97/SC21: *LOTOS- A formal Description technique Based on Temporal Ordering of Observational Behaviour* (ISO/IS 8807), 1988.
[71] ITU-T: Recommendation Z.105 - *SDL combined with ASN.1*, Genf 1995.
[72] ITU-T: Recommendation Z.100 - *CCITT Specification and Description Language (SDL)*, Genf 1993.
[73] ITU-T: Recommendation Z.120 - *Message Sequence Charts (MSC)*, Genf 1993.
[74] R. Jain: *The Art of Computer Systems Performance Analysis - Techniques for Experimental Design, Measurements, Simulation and Modeling*. Wiley and Sons, New York 1991.
[75] H. Kabutz: *Analytical Performance Evaluation of Concurrent Communicating systems using SDL and stochastic Petri nets*. Dissertation University of Cape Town, Südafrika 1997.
[76] H. König, H. Krumm: *Implementierung von Kommunikationsprotokollen*; in: Informatik-Spektrum 19 (1996), Seite 316-325, Springer-Verlag Berlin 1996.

[77] O. Kyas: *ATM-Netzwerke*; Datacomverlag, Bergheim 1993.
[78] W. Li und D. Latchman: *WWW Traffic Characterization and HTTP Performance on Serial Line & LAN Connections*; Technischer Bericht, Universität Florida, Gainsville Florida 1995.
[79] A. Mitschele-Thiel und B. Müller-Clostermann: *Performance Engineering of SDL/MSC Systems*; in: A. Cavalli and A. Sarma (Ed.) SDL'97 Time for Testing SDL, MSC and Trends. Tutorials des 8. SDL-Forums, Evry 1997.
[80] A. Mitschele-Thiel und B. Müller-Clostermann: *Performance Engineering of SDL/MSC Systems*; in: Computer Networks; The International Journal of Computer and Telecommunications Networking; Nr. 31(1999) S. 1801-1815, Elsevier, 1999.
[81] T.I. Morley: *The Application of SDL to Control Systems for Industrial Machines*; in: R. Braek and A. Sarma (Ed.) SDL'95 with MSC in CASE. Tagungsband des 7. SDL-Forums, Oslo 1995. Elsevier, Amsterdam 1995.
[82] R. Nagarajan, C.Vogt: Guaranteed-Performance *Transport of Multimedia Traffic over the Token Ring*. IBM TR 43.901, IBM ENC, Heidelberg, Januar 1992.
[83] R. Onvural: *Asynchronous Transfer Mode Networks: Performance Issues*. Artech House, Boston 1993.
[84] OPNET-Planner: Informationen via WWW. http://www.mil3.com.
[85] V. Paxson: *Empirically-Derived Analytic Models of Wide-Area TCP Connections*; in: IEEE/ACM Transaction on Networking, Vol. 2, Nr. 4, Seite 316ff, August 1994.
[86] V. Paxson und S. Floyd: *Wide-Area Traffic: The Failure of Poisson Modeling;* in IEEE/ACM Transaction on networking, Vol.3, Nr. 3, Seite 226ff, Juni 1995.
[87] J. Postel und J. Reynolds: *RFC 791 Internet Protocol Specification.* 1981.
[88] J. Postel und J. Reynolds: *RFC 793 Transmission Control Protocol Specification.* 1981.
[89] J. Postel und J. Reynolds: *RFC 854 Telnet Protocol Specification.* 1983.
[90] J. Postel und J. Reynolds: *RFC 959 File Transfer Protocol.* 1985.
[91] QUEST: User Manual; Verfügbar im Web unter: http://www.cs.uni-essen.de/QUEST.
[92] A. Robnik, M. Dolenc, M. Alcin: *Industrial Experience of Using SDL in IskraTel*; in: R. Braek and A. Sarma (Ed.) SDL'95 with MSC in CASE. Tagungsband des 7. SDL-Forums, Oslo 1995. Elsevier, Amsterdam 1995.
[93] C. Rodemeyer: Entwicklung und Implementierung eines Auswertungs- und Visualisierungssystems für QSDL-Simulatoren. Diplomarbeit Universität Essen, 1995.
[94] O. Rose: *Statistical properties of MPEG video traffic and theit impact on traffic modeling in ATM systems*; in: Prooceedings of 20th Annual Conference on Local Computer Networks, Minneapolis, Oktober 1995.
[95] O. Rose: Simple and Efficient Models for Variable Bit Rate MPEG Video Traffic; Technischer Bericht, Universität Würzburg, 1995.
[96] H. Rudin: *Time in Formal Protocol Specifications;* in: Kommunikation in verteilten Systemen I, Karlsruhe, März 1985.
[97] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobsen: *RFC1889 RTP: A Transportprotocol for Real Time Applications* 1996.
[98] SDT 3.1 *Reference Manual & User's Guide*. TeleLogic Malmö AB, 1997.
[99] SES-Strategizer. Informationen über www. http://www.ses.com.
[100] J. Shoch, Y. Dalal, D. Redell, R. Crane: *Evolution of the Ethernet local computer network*. Computer, August 1982.
[101] SPEC: The Standard Performance Evaluation Corporation; via world wide web http://www.spec.org/results.html.
[102] B. Sprunt, L.Sha, J. Lehoczky: *Aperiodic Task Scheduling for Hard Real-Time Systems*; The Journal of Real-Time Systems, Vol. 1, Seite 27-60, 1989.

[103] W. Stallings: *Handbook of Computer Communication Standards; Volume 2 Local Network Standards*. MacMillan 1987.
[104] W. Stallings: Networking Standards; A Guide to OSI, ISDN, LAN, and MAN Standards. Adisson-Wesley, Bonn 1993.
[105] R. Steinmetz: *Multimedia Technologie-Einführung und Grundlagen*. Springer-Verlag, Berlin 1993.
[106] M. Steppler und M. Lott: *SPEET - SDL Performance Evaluation Tool* in: A. Cavalli and A. Sarma (Ed.) SDL'97 Time for Testing SDL, MSC and Trends. Tagungsband des 8. SDL-Forums, Evry 1997. Elsevier, Amsterdam 1997.
[107] W.R. Stevens: *TCP/IP Illustrated Volume I The Protocols*; Addison-Wesley, Bonn 1994.
[108] A. Tanenbaum: *Computer Networks*. Prentice Hall 1992.
[109] M. Tatipamula, B. Khasnabish (Ed.): *Multimedia Communications Networks, Technologies and Services*; Artech House, London 1998.
[110] W. Textor: *Ein Macintosh-Simulationswerkzeug für Kommunikationsprotokolle*. Diplomarbeit, Universität Essen, 1994.
[111] C. Topolcic: *RFC 1190: Internet Stream Protocol Version 2 (ST-II)* 1990.
[112] Verilog: ObjectGEODE SDL Simulator - Reference Manual; http:\\www.verilogusa.com.
[113] M. Walch: *Evaluating Parallel Processing of Communication Protocols*; Oldenbourg Verlag, München 1994.
[114] C. Wohlin: *Performance analysis of SDL systems from SDL descriptions* in: O. Faergemand und R. Reed (Ed.) SDL'91 Evolving Methods. Tagungsband des 5. SDL-Forums, Glasgow 1991. North Holland, Amsterdam 1991.
[115] R. Wolff: *Stochastic Modeling and the Theory of Queues*. Prentice Hall, New Jersey 1989.
[116] M. Zitterbart: *Flexible und effiziente Kommunikationssysteme für Hochgeschwindigkeitsnetze*. International Thomson Publishing, Bonn 1995.

Abbildungsverzeichnis

Tabellenverzeichnis

Stichwortverzeichnis

C

D

E

F

H

I

M

Q

R

S

T

U

V

W

Z

Anhang A Semantik von QSDL

Das dynamische Verhalten eines QSDL-Systems ergibt sich durch das Verhalten des eingebetteten SDL-Systems, sowie durch das Verhalten der spezifizierten Maschinen. Weiterhin sind die Ergänzungen von Prozessen in bezug auf request, awake und output-delay zu berücksichtigen.

Zunächst sollen einige vorbereitende Bezeichnungen und Definitionen eingeführt werden.

A.1 Begriffe und Definitionen

Mit $SCHED = \{FCFS, PS, IS, Random, FCFSPP, FCFSPNP\}$ wird die Menge der verfügbaren Schedulingstrategien bezeichnet.

Die Menge $MS = \{ms_1, ..., ms_k\}$ enthält als Elemente alle im System verfügbaren Maschinendienste.

Unter einem von einer Maschine angebotenen Dienst *d* wird ein Maschinendienst und die für die Maschine spezifische Geschwindigkeit, mit der diese Maschine den Dienst erbringt, verstanden. Formal ausgedrückt gilt:

$$d = \langle ms_i, \alpha_i \rangle \in (MS \times \Re_0^{plus}) \text{ , wobei } 1 \le i \le k \text{ .}$$

Mit $D = \{d_1, ..., d_w\}$ wird die Menge aller Dienste bezeichnet

Eine konkrete Dienstanforderung, die per request von einem QSDL-Prozeß initiiert wird, wird als 3-Tupel $r = \langle ms_i, \omega, priority \rangle \in (MS \times \Re_0^{plus} \times \aleph_0)$ bezeichnet. Dabei bezeichnet ms_i den gewünschten Maschinendienst, ω den Umfang des Bedienwunsches und *priority* die Priorität des Dienstes. Wird bei einer Dienstanforderung kein Wert für *priority* angegeben, gilt standardmäßig der Wert 0 (niedrigste Priorität). Mit $R = \{r_1, ..., r_p\}$ wird die Menge aller Dienstanforderungen bezeichnet.
Als Dienstelement wird eine Dienstanforderung zusammen mit dessen Ankunftszeitpunkt t_A , der Schedulingzeit t_{Sched} und der Restbedienzeit t_{Rest} bezeichnet. Es gilt:

$$e = \langle r, t_A, t_{Sched}, t_{Rest} \rangle \in (R \times \Re_0^{plus} \times \Re_0^{plus} \times \Re_0^{plus})$$

Die Menge $E = \{e_1, \ldots, e_t\}$ enthält alle Dienstelemente. Zusätzlich läßt sich die Menge der Dienstelemente in disjunkte Teilmengen bezüglich *priority* zerlegen. Es sei $E = \bigcup^{u} E_i$. Dann gilt: $\forall e_i \in E \qquad e_i \in E_f \Leftrightarrow (Prior(e_i) = f)$.

In QSDL-Systemen sind zwei Ereignisse, die Dienstaufrufe betreffend, beobachtbar.

1. Ankunft einer Dienstanforderung $An(r_j, t_{now})$ an einer Maschine zum Zeitpunkt t_{now} .
2. Bedienende eines Dienstelementes $Be(e_i, t_{now})$ zum Zeitpunkt t_{now} .

Mit E_v wird die Menge der Ereignisse bezeichnet, d.h

$$E_v = \{An(r_j, t), Be(e_i, t)\}$$

Mit Hilfe dieser Definitionen kann man eine QSDL-Maschine wie folgt beschreiben:

Eine Maschine $M = (n, strat, AD, Z, \delta)$ ist ein 5-Tupel mit den folgenden Elementen:

$n \in \aleph$	Anzahl Bedieneinheiten
$strat \in sched$	Schedulingstrategie
$AD \subseteq D$	Menge der angebotenen Maschinendienste und die dienstspezifischen Geschwindigkeiten
Z	(unendliche) Menge der Zustände, die eine Maschine einnehmen kann
δ: $Z \times E_v \rightarrow Z$	Zustandsübergangsfunktion

Dabei kann der Zustand einer QSDL-Maschine als 3-Tupel Z_M = (W,B,A) beschrieben werden, wobei:

$W = \langle w_1, \ldots, w_m \rangle$	$w_i \in E$	wartende Dienstelemente
$B = \langle b_1, \ldots, b_r \rangle$	$b_i \in E$	in Bedienung befindliche Dienstelemente r $\leq$ n
$A = \langle (ms_1, \alpha_1), \ldots, (ms_p, \alpha_p) \rangle$	$(ms_i, \alpha_i) \in D$	Tupel von Diensten

Mit den eingeführten Notationen und Ereignissen lassen sich nun Eigenschaften und das dynamische Verhalten der QSDL-Maschinen formal aufschreiben.

A.2 FCFS-Maschine

Eine Maschine mit Schedulingstrategie FCFS arbeitet wie eine Warteschlange. Dienstaufrufe werden entsprechend der Reihenfolge ihres Eintreffens abgearbeitet. Eventuell vorhandene Prioritätsangaben in den Dienstanforderungen werden ignoriert. Für diese Maschinen gelten die folgenden Bedingungen:

1. Es gibt keine wartenden Kunden, wenn noch Bediener frei sind, dh. falls $B = \langle b_1, \ldots, b_k \rangle$ und $k \leq n \Rightarrow (W = \langle \varepsilon \rangle)$.

2. Wenn der Warteraum nicht leer ist, sind alle Bediener belegt, d.h.

   Falls $W \neq \langle \varepsilon \rangle \Rightarrow \left(B = \langle\ b_1, \ldots, b_n\ \rangle \right)$.

3. Innerhalb des Warteraumes sind die Dienstelemente nur nach Ankunftszeitpunkten sortiert, d.h:
   Falls $W \neq \langle \varepsilon \rangle$ dann beschreibt $<_t$ die Ordnungsrelation *steht vor* und es gilt: $w_i <_t w_j \Rightarrow t_A(w_i) \leq t_A(w_j)$, sowie $t_A(w_i) < t_A(w_j) \Rightarrow w_i <_t w_j$.

4. Für alle Dienstelemente im Bedienraum gilt, daß ihre Ankunftzeit vor der Ankunftzeit des ersten wartenden Elementes ist. Formal ausgedrückt:
   $(\forall e \in B)$ gilt: $t_A(e) < t_A(w_1)$.

Die dynamische Semantik einer FCFS-Maschine, d.h. deren funktionales Verhalten kann durch die Funktion δ wie folgt beschrieben werden:

1. Ankunft einer Dienstanforderung an einer leeren Maschine führt zur unmittelbaren Bedienung der Anforderung, formal
   $\delta((\langle\varepsilon\rangle, \langle\varepsilon\rangle, A), An(r_i, t_{now})) = (\langle\varepsilon\rangle, \langle b_1\rangle, A)$, mit $b_1 = \langle r_i, t_{now}, t_{now}, \omega\rangle$ und $r_i = \langle ms_i, \omega, prior(r_i)\rangle$.
   Gleichzeitig wird das Endeereignis $Be(e_i, t_{now} + \omega/\alpha_i)$ generiert, wobei $e_i = \langle r_i, t_{now}, t_{now}, 0\rangle$. Der Zeitpunkt ergibt sich aus dem Ankunftzeitpunkt, dem Bedienwunsch und der Bediengeschwindigkeit der Maschine.

2. Ankünfte von Dienstanforderungen an Maschinen mit mindestens einem freien Bediener werden sofort vom ersten freien Bediener bearbeitet, formal
   $\delta((\langle\varepsilon\rangle, \langle b_1, \ldots, b_k\rangle, A), An(r_i, t_{now})) = (\langle\varepsilon\rangle, \langle b_1, \ldots, b_k, b_{k+1}\rangle, A)$ mit
   $b_{k+1} = \langle r_i, t_{now}, t_{now}, \omega\rangle$ und $r_i = \langle ms_i, \omega, prior(r_i)\rangle$.
   Auch hier kann das zugehörige Ende-Ereignis $Be(e_i, t_{now} + \omega/\alpha_i)$ generiert werden.

3. Bei Ankünften an einer vollständig belegten Maschine wird die neuankommende Anforderung hinter der letzten wartenden einsortiert, formal
   $\delta((\langle w_1, \ldots, w_m\rangle, \langle b_1, \ldots, b_n\rangle, A), An(r_i, t_{now})) = (\langle w_1, \ldots, w_m, e\rangle, \langle b_1, \ldots, b_n\rangle, A)$ mit
   $e = \langle r_i, t_{now}, nd, \omega\rangle$ und
   $r_i = \langle ms_i, \omega, prior(r_i)\rangle$.
   Ein Ende-Ereignis kann nicht bestimmt werden, der Wert für t_{Sched} ist undefiniert (*nd*).

4. Bei Beendigung eines Dienstelementes und nicht leerem Warteraum wird das an erster Stelle wartende Dienstelement dem frei gewordenen Bediener zugeordnet,, formal
   $\delta((\langle w_1, \ldots, w_m\rangle, \langle b_1, \ldots, b_n\rangle, A), Be(e_i, t_{now})) =$
   $(\langle w_2, \ldots, w_m\rangle, \langle b_1, \ldots, b_{i-1}, b_{i'}, \ldots, b_n\rangle, A)$. Weiterhin gilt:
   $b_{i'} = w_1 = \langle ms_k, \omega, prior(e_i), t_A, t_{now}, \omega\rangle$
   Dessen Bedienzeit ergibt sich wieder durch Division des Bedienwunsches durch die Geschwindigkeit und daraus berechnet sich der Zeitpunkt des Bedienendes zu $(t_{now} + \omega/\alpha_k)$.

5. Bei Beendigung eines Dienstelementes und leerem Warteraum wird ein weiterer Bediener untätig, formal
   $\delta((\langle\varepsilon\rangle, \langle b_1, \ldots, b_k\rangle, A), Be(e_i, t_{now})) = (\langle\varepsilon\rangle, \langle b_1, \ldots, b_{i-1}, b_{i+1}, \ldots, b_k\rangle, A)$.

6. Der Abgang des letzten Kunden hinterläßt eine leere Maschine, also $\delta((\langle\varepsilon\rangle, \langle b_1\rangle, A), Be(e_i, t_{now})) = (\langle\varepsilon\rangle, \langle\varepsilon\rangle, A)$.

Damit ist die dynamische Semantik der FCFS-Maschine vollständig beschrieben.

A.3 Random-Maschine

Diese Maschine entnimmt die Dienstelement aus dem Warteraum zufällig. Die Wahrscheinlichkeit, aus dem Warteraum entnommen zu werden, ist für alle wartenden Elemente gleich und hat den Wert $\frac{1}{k}$ bei k-wartenden Elementen. Auch diese Maschine ignoriert Prioritätsangaben.

Für diese Maschine lassen sich folgende Bedingungen angeben:

1. Es gibt keine wartenden Kunden, wenn noch Bediener frei sind, dh. falls $B = \langle b_1, \ldots, b_k\rangle$ und $k \le n \Rightarrow (W = \langle\varepsilon\rangle)$.
2. Wenn der Warteraum nicht leer ist, sind alle Bediener belegt, d.h.

   Falls $W \neq \langle\varepsilon\rangle \Rightarrow \left(B = \langle\ b_1, \ldots, b_n\ \rangle\right)$.
3. Innerhalb des Warteraumes sind die Dienstelemente nur nach Ankunftszeitpunkten sortiert, d.h: Falls $W \neq \langle\varepsilon\rangle$, dann $w_i <_t w_j \Rightarrow t_A(w_i) \le t_A(w_j)$, sowie $t_A(w_i) < t_A(w_j) \Rightarrow w_i <_t w_j$.

Die dynamische Semantik ergibt sich durch die folgende Funktion δ:

1. Ankunft einer Dienstanforderung an einer leeren Maschine führt zur unmittelbaren Bedienung der Anforderung, formal

   $\delta((\langle\varepsilon\rangle, \langle\varepsilon\rangle, A), An(r_i, t_{now})) = (\langle\varepsilon\rangle, \langle b_1\rangle, A)$, mit $b_1 = \langle r_i, t_{now}, t_{now}, \omega\rangle$ und $r_i = \langle ms_i, \omega, prior(r_i)\rangle$.

   Gleichzeitig wird das Ende-Ereignis $Be(e_i, t_{now} + \omega/\alpha_i)$ generiert, wobei $e_i = \langle r_i, t_{now}, t_{now}, 0\rangle$. Der Zeitpunkt ergibt sich aus dem Ankunftzeitpunkt, dem Bedienwunsch und der Bediengeschwindigkeit der Maschine.
2. Ankünfte von Dienstanforderungen an Maschinen mit mindestens einem freien Bediener werden sofort vom ersten freien Bediener bearbeitet, formal

   $\delta((\langle\varepsilon\rangle, \langle b_1, \ldots, b_k\rangle, A), An(r_i, t_{now})) = (\langle\varepsilon\rangle, \langle b_1, \ldots, b_k, b_{k+1}\rangle, A)$ mit $b_{k+1} = \langle r_i, t_{now}, t_{now}, \omega\rangle$ und $r_i = \langle ms_i, \omega, prior(r_i)\rangle$.

   Auch hier kann das zugehörige Ende-Ereignis $Be(e_i, t_{now} + \omega/\alpha_i)$ generiert werden.
3. Bei Ankünften an einer vollständig belegten Maschine wird die neuankommende Anforderung hinter der letzten wartenden einsortiert, formal

   $\delta((\langle w_1, \ldots, w_m\rangle, \langle b_1, \ldots, b_n\rangle, A), An(r_i, t_{now})) = (\langle w_1, \ldots, w_m, e\rangle, \langle b_1, \ldots, b_n\rangle, A)$ mit $e = \langle r_i, t_{now}, nd, \omega\rangle$ und $r_i = \langle ms_i, \omega, prior(r_i)\rangle$.

   Ein Ende-Ereignis kann nicht bestimmt werden, der Wert für t_{Sched} ist undefiniert (*nd*).
4. Falls ein Dienstelement mit seiner Bedienung zu Ende ist, wird zufällig eines der wartenden Elemente ausgewählt, dh.

   $\exists(j \in \{1, \ldots, m\})$ so daß

$\delta((\langle w_1, \ldots, w_m\rangle, \langle b_1, \ldots, b_n\rangle, A), Be(e_i, t_{now}))$ =

$(\langle w_1, \ldots, w_{j-1}, w_{j+1}, \ldots, w_m\rangle, \langle b_1, \ldots, b_{i-1}, b_i, \ldots, b_n\rangle, A)$, mit

$b_i = w_j = \langle ms_g, \omega, prior(e_j), t_A, t_E, \omega\rangle$. Die Wahrscheinlichkeit, aus dem Warteraum in Bedienung zu gelangen, ist für alle wartenden Dienstelemente gleich und ergibt sich zu:

$p[w_j] = \frac{1}{m} \quad \forall(j \in \{1, \ldots, m\})$.

Eine Maschine, die ein zufälliges Scheduling innerhalb von Prioritätsklassen vornimmt, ist nicht vorgesehen, aber prinzipiell denkbar.

A.4 FCFSPP-Maschine

Eine FCFSPP[a]-Maschine ist eine Maschine, die mit Prioritäten arbeitet. Der Warteraum enthält Äquivalenzklassen bezüglich der Priorität von Dienstelementen. Die Dienstelemente mit höchster Priorität werden zuerst bedient. Innerhalb der Prioritätsklassen sind die wartenden Dienstelemente nach Ankunftzeitpunkten sortiert. Neuankommende Dienstelemente können in Bedienung befindliche Elemente unterbrechen, wenn diese eine geringere Priorität aufweisen. Für die Maschinen vom Typ FCFSPP lassen sich folgende Aussagen formulieren:

1. Es gibt keine wartenden Kunden, wenn noch Bediener frei sind, dh. falls $B = \langle b_1, \ldots, b_k\rangle$ und $k \leq n \Rightarrow (W = \langle\varepsilon\rangle)$.
2. Wenn der Warteraum nicht leer ist, sind alle Bediener belegt, d.h.

   Falls $W \neq \langle\varepsilon\rangle \Rightarrow \left(B = \langle\ b_1, \ldots, b_n\ \rangle\right)$.
3. Innerhalb des Warteraumes sind die Dienstelemente nach Priotitäten und innerhalb der Prioritätsklassen nach Ankunftszeitpunkten sortiert, d.h. falls $W \neq \langle\varepsilon\rangle$ dann $w_i <_t w_j \Leftrightarrow$

   $Prior(w_i) > Prior(w_j) \vee (Prior(w_i) = Prior(w_j)) \wedge t_A(w_i) < t_A(w_i)$
4. Weiterhin gilt für alle Dienstelemente des Bedienraumes, daß entweder die Priorität höher ist als die des an erster Position wartenden Elementes oder falls diese gleich sind, dann gilt, daß der Ankunftszeitpunkt des ersten wartenden Elementes größer ist, als die Ankunftzeitpunkte aller in Bedienung befindlicher Elemente. Formal: $(\forall e \in B)$ gilt:

   $Prior(e) > Prior(w_1) \vee ((Prior(e) = Prior(w_1)) \wedge t_A(e) < t_A(w_1))$

Die dynamische Semantik der Maschine wird durch die folgende Funktion δ beschrieben:

1. Ankunft einer Dienstanforderung an einer leeren Maschine führt zur unmittelbaren Bedienung der Anforderung, formal

   $\delta((\langle\varepsilon\rangle, \langle\varepsilon\rangle, A), An(r_i, t_{now})) = (\langle\varepsilon\rangle, \langle b_1\rangle, A)$, mit $b_1 = \langle r_i, t_{now}, t_{now}, \omega\rangle$ und $r_i = \langle ms_i, \omega, prior(r_i)\rangle$.

   Gleichzeitig wird das Ende-Ereignis $Be(e_i, t_{now} + \omega/\alpha_i)$ generiert, wobei $e_i = \langle r_i, t_{now}, t_{now}, 0\rangle$. Der Zeitpunkt ergibt sich aus dem Ankunftzeitpunkt, dem Bedienwunsch und der Bediengeschwindigkeit der Maschine.

a. First Come First Serve Prior Preemptive Resume

2. Ankünfte von Dienstanforderungen an Maschinen mit mindestens einem freien Bediener werden sofort vom ersten freien Bediener bearbeitet, formal

   $\delta((\langle\varepsilon\rangle, \langle b_1, \ldots, b_k\rangle, A), An(r_i, t_{now})) = (\langle\varepsilon\rangle, \langle b_1, \ldots, b_k, b_{k+1}\rangle, A)$ mit

   $b_{k+1} = \langle r_i, t_{now}, t_{now}, \omega\rangle$ und $r_i = \langle ms_i, \omega, prior(r_i)\rangle$.

   Auch hier kann das zugehörige Ende-Ereignis $Be(e_i, t_{now} + \omega/\alpha_i)$ generiert werden.

3. Bei Ankunft einer Dienstanforderung an einer vollständig belegten Maschine ergeben sich zwei Fälle:

   (a) Die Priorität des neuankommenden Dienstelement ist kleiner oder gleich dem Minimum aller Prioritäten im Bedienraum, also:

   $Prior(r_i) \leq Prior(j)) \quad \forall(j \in \{1, \ldots, n\})$.

   Dann wird das neuankommende Element innerhalb seiner Prioritätsklasse als letztes einsortiert, formal

   $\delta((\langle w_{l1}, \ldots, w_{lr}, \ldots, w_{p1}, \ldots, w_{pp}, \ldots, w_{01}, \ldots, w_{0q}\rangle, \langle b_1, \ldots, b_n\rangle, A), An(r_i, t_{now}))$ =

   $(\langle w_{l1}, \ldots, w_{lr}, \ldots, w_{p1}, \ldots, w_{pp+1}, \ldots, w_{01}, \ldots, w_{0q}\rangle, \langle b_1, \ldots, b_n\rangle, A)$, falls

   $prior(r_i) = p$.

   Ein Ende-Ereignis kann nicht bestimmt werden, der Wert für t_{Sched} ist undefiniert.

   (b) Es gibt im Bedienraum Dienstelemente, deren Priorität kleiner ist als die des Neuankommenden. Es wird dann das Element gesucht, dessen Priorität am geringsten ist und innerhalb dieser Klasse das Element, dessen Ankunftzeit den größten Wert hat. Diese Element wird unterbrochen, seine neue Restbedienzeit wird berechnet als Differenz aus Ereigniszeitpunkt *now* (t_{now}) und dem Zeitpunkt, zu dem dieses Element in Bedienung gekommen ist (t_{Sched}). Dann wird das unterbrochene Element innerhalb seiner Prioritätsklasse vorne im Warteraum einsortiert.

   Formal bedeutet das:

   $\delta((\langle w_{l1}, \ldots, w_{lr}, \ldots, w_{p1}, \ldots, w_{pu}, \ldots, w_{01}, \ldots, w_{0q}\rangle, \langle b_1, \ldots, b_n\rangle, A), An(r_i, t_{now}))$ =

   $(\langle w_{l1}, \ldots, w_{lr}, \ldots, w_{p1}, \ldots, w_{ps}, \ldots, w_{01}, \ldots, w_{0q}\rangle, \langle b_1, \ldots, b_{j-1}, e, b_{j+1}, \ldots, b_n\rangle, A)$ mit

   $p = min(Prior(j)) \quad \forall(j \in \{1, \ldots, n\})$ und

   $w_{p1'} = b_j = \langle ms_g, \omega, p, t_A, nd, t_{sched} - t_{now}\rangle$ sowie

   $w_{pv'} = w_{pv-1} \quad \forall(v' = 2, \ldots, s)$ und $s = u + 1$.

   Das Ereignis Bedienende des unterbrochenen Kunden muß gelöscht werden.

4. Bei Beendigung eines Dienstelementes und nicht leerem Warteraum wird das an erster Stelle wartende Dienstelement dem frei gewordenen Bediener zugeordnet, formal

   $\delta((\langle w_{l1}, \ldots, w_{lr}, \ldots, w_{p1}, \ldots, w_{pp}, \ldots, w_{01}, \ldots, w_{0q}\rangle, \langle b_1, \ldots, b_n\rangle, A), Be(e_i, t_{now})) =$

   $(\langle w_{l2}, \ldots, w_{lr}, \ldots, w_{p1}, \ldots, w_{pp}, \ldots, w_{01}, \ldots, w_{0q}\rangle, \langle b_1, \ldots, b_{i-1}, b_i, \ldots, b_n\rangle, A)$, wobei $b_i = w_{l1}$.

   Dessen Bedienzeit ergibt sich wieder durch Division des Bedienwunsches durch die Geschwindigkeit und daraus berechnet sich der Zeitpunkt des Bedienendes zu $(t_{now} + \omega/\alpha_i)$.

5. Bei Beendigung eines Dienstelementes und leerem Warteraum wird ein weiterer Bediener untätig, formal

   $\delta((\langle\varepsilon\rangle, \langle b_1, \ldots, b_k\rangle, A), Be(e_i, t_{now})) = (\langle\varepsilon\rangle, \langle b_1, \ldots, b_{i-1}, b_{i+1}, \ldots, b_k\rangle, A)$.

6. Der Abgang des letzten Kunden hinterläßt eine leere Maschine, also $\delta((\langle\varepsilon\rangle, \langle b_1\rangle, A), Be(e_i, t_{now})) = (\langle\varepsilon\rangle, \langle\varepsilon\rangle, A)$

A.5 FCFSPNP-Maschine

Eine FCFSPNP[b]-Maschine ist eine Maschine, die ebenfalls mit Prioritäten arbeitet. Der Warteraum zerfällt wie bei FCFSPP in Äquivalenzklassen bezüglich der Priorität von Dienstelementen. Die Dienstelemente mit höchster Priorität werden zuerst bedient. Innerhalb der Prioritätsklassen sind die wartenden Dienstelemente nach Ankunftzeitpunkten sortiert. Neuankommende Dienstelemente können in Bedienung befindliche Elemente nicht unterbrechen. Für die Maschinen vom Typ FCFSPP lassen sich folgende Aussagen formulieren:

1. Es gibt keine wartenden Kunden, wenn noch Bediener frei sind, dh. falls $B = \langle b_1, \ldots, b_k\rangle$ und $k \leq n \Rightarrow (W = \langle\varepsilon\rangle)$.
2. Wenn der Warteraum nicht leer ist, sind alle Bediener belegt, d.h.

   Falls $W \neq \langle\varepsilon\rangle \Rightarrow \left(B = \langle\ b_1, \ldots, b_n\ \rangle\right)$.
3. Innerhalb des Warteraumes sind die Dienstelemente nach Prioritäten und innerhalb der Prioritätsklassen nach Ankunftszeitpunkten sortiert, d.h. falls $W \neq \langle\varepsilon\rangle$ dann $w_i <_t w_j \Leftrightarrow$

   $$Prior(w_i) > Prior(w_j) \vee (Prior(w_i) = Prior(w_j)) \wedge t_A(w_i) < t_A(w_i)$$
4. Für alle Dienstelemente im Bedienraum gilt, daß ihre Ankunftzeit vor der Ankunftzeit des ersten wartenden Elementes derselben Prioritätsklasse ist. Formal ausgedrückt: $(\forall e_k \in B)$ gilt: $t_A(e_k) < t_A(w_{k1})$ wobei $e_k \in E_k \Leftrightarrow (Prior(e_k) = k)$ und $w_{ki} \in W_k \subseteq E_k \Leftrightarrow (Prior(w_{ki}) = k)$

Da neuankommende Dienstelemente bereits in Bedienung befindliche Elemente nicht unterbrechen können, kann die Aussage 4. aus Abschnitt A.4 für Maschinen vom Typ FCFSPNP nicht aufrecht erhalten werden. Dies gilt ebenso für die Aussage 4. aus Abschnitt A.2, da durch die prioritätsgesteuerte Auswahl wartender Dienstelemente, später eintreffende Elemente früher behandelt werden können, wenn sie zu einer höheren Prioritätsklasse gehören.

Die dynamische Semantik ergibt sich durch die Funktion δ wie folgt:

1. Ankunft einer Dienstanforderung an einer leeren Maschine führt zur unmittelbaren Bedienung der Anforderung, formal

   $\delta((\langle\varepsilon\rangle, \langle\varepsilon\rangle, A), An(r_i, t_{now})) = (\langle\varepsilon\rangle, \langle b_1\rangle, A)$, mit $b_1 = \langle r_i, t_{now}, t_{now}, \omega\rangle$ und $r_i = \langle ms_i, \omega, prior(r_i)\rangle$.

   Gleichzeitig wird das Ende-Ereignis $Be(e_i, t_{now} + \omega/\alpha_i)$ generiert, wobei $e_i = \langle r_i, t_{now}, t_{now}, 0\rangle$. Der Zeitpunkt ergibt sich aus dem Ankunftzeitpunkt, dem Bedienwunsch und der Bediengeschwindigkeit der Maschine.

b. First Come First Serve Prior Nonpreemptive Resume

2. Ankünfte von Dienstanforderungen an Maschinen mit mindestens einem freien Bediener werden sofort vom ersten freien Bediener bearbeitet, formal
   $\delta((\langle\varepsilon\rangle, \langle b_1, \ldots, b_k\rangle, A), An(r_i, t_{now})) = (\langle\varepsilon\rangle, \langle b_1, \ldots, b_k, b_{k+1}\rangle, A)$ mit
   $b_{k+1} = \langle r_i, t_{now}, t_{now}, \omega\rangle$ und $r_i = \langle ms_i, \omega, prior(r_i)\rangle$.
   Auch hier kann das zugehörige Ende-Ereignis $Be(e_i, t_{now} + \omega/\alpha_i)$ generiert werden.

3. Bei Ankünften an einer vollständig belegten Maschine wird die neuankommende Anforderung in die zugehörige Klasse hinter der letzten wartenden der entsprechenden Klasse einsortiert, formal
   $\delta((\langle w_{l1}, \ldots, w_{lr}, \ldots, w_{p1}, \ldots, w_{pp}, \ldots, w_{01}, \ldots, w_{0q}\rangle, \langle b_1, \ldots, b_n\rangle, A), An(r_i, t_{now})) =$
   $(\langle w_{l1}, \ldots, w_{lr}, \ldots, w_{p1}, \ldots, w_{pp+1}, \ldots, w_{01}, \ldots, w_{0q}\rangle, \langle b_1, \ldots, b_n\rangle, A)$, falls
   $prior(r_i) = p$.
   Ein Ende-Ereignis kann nicht bestimmt werden, der Wert für t_{Sched} ist undefiniert.

4. Bei Beendigung eines Dienstelementes und nicht leerem Warteraum wird das an erster Stelle wartende Dienstelement dem frei gewordenen Bediener zugeordnet, formal
   $\delta((\langle w_{l1}, \ldots, w_{lr}, \ldots, w_{p1}, \ldots, w_{pp}, \ldots, w_{01}, \ldots, w_{0q}\rangle, \langle b_1, \ldots, b_n\rangle, A), Be(e_i, t_{now})) =$
   $(\langle w_{l2}, \ldots, w_{lr}, \ldots, w_{p1}, \ldots, w_{pp}, \ldots, w_{01}, \ldots, w_{0q}\rangle, \langle b_1, \ldots, b_{i-1}, b_i, \ldots, b_n\rangle, A)$, wobei $b_i = w_{l1}$.
   Dessen Bedienzeit ergibt sich wieder durch Division des Bedienwunsches durch die Geschwindigkeit und daraus berechnet sich der Zeitpunkt des Bedienendes zu $(t_{now} + \omega/\alpha_i)$.

5. Bei Beendigung eines Dienstelementes und leerem Warteraum wird ein weiterer Bediener untätig, formal
   $\delta((\langle\varepsilon\rangle, \langle b_1, \ldots, b_k\rangle, A), Be(e_i, t_{now})) = (\langle\varepsilon\rangle, \langle b_1, \ldots, b_{i-1}, b_{i+1}, \ldots, b_k\rangle, A)$.

6. Der Abgang des letzten Kunden hinterläßt eine leere Maschine, also
   $\delta((\langle\varepsilon\rangle, \langle b_1\rangle, A), Be(e_i, t_{now})) = (\langle\varepsilon\rangle, \langle\varepsilon\rangle, A)$

A.6 IS-Maschine

Die IS[c]-Maschine steht für maximale Parallelität. Es steht jedem ankommenden Dienstelement sofort ein Bediener zur Verfügung, der mit der für den angeforderten Dienst spezifizierten Geschwindigkeit bedient. Daher existiert bei dieser Maschine kein Warteraum und es reichen wenige Gleichungen, um Eigenschaften und dynamische Semantik zu beschreiben.

Als Invariante gilt für alle Zustände der Maschine:

$W = \langle\varepsilon\rangle$ und $B = \langle b_1, \ldots, b_k\rangle$ für $k = 1, 2, 3, \ldots$

Die Funktion δ wird durch zwei Gleichungen beschrieben:

1. formal
   $\delta((\langle\varepsilon\rangle, \langle b_1, \ldots, b_k\rangle, A), An(r_i, t_{now})) = (\langle\varepsilon\rangle, \langle b_1, \ldots, b_k, b_{k+1}\rangle, A)$ mit $b_{k+1} = \langle r_i, t_{now}, t_{now}, \omega\rangle$ und $r_i = \langle ms_i, \omega, prior(r_i)\rangle$.
   Gleichzeitig ergibt sich das zugehörige Ereignis Bedienende zu
   $Be(b_{k+1}, t_{now} + \omega/\alpha_i)$

c. Invinite Server

2. Bei Bedienende eines Dienstelementes reduziert sich der Bedienraum um ein Element, formal

$$\delta((\langle\varepsilon\rangle, \langle b_1, \ldots, b_k\rangle, A), Be(e_i, t_E)) = (\langle\varepsilon\rangle, \langle b_1, \ldots, b_{i-1}, b_{i+1}, \ldots, b_n\rangle, A)$$

A.7 PS-Maschine

Die PS[d]-Maschine ist eine Approximation des Schedulingverfahrens Round Robin mit sehr kleiner Zeitscheibe. Alle Dienstelemente werden quasi-parallel bedient. Es gibt daher ebenfalls keinen Warteraum, allerdings reduziert diese Maschine ihre Geschwindigkeit pro neuankommenden Dienstelement, wenn mehr als n Dienstelemente in Bedienung sind.

Analog zur IS-Maschine gilt:

$W = \langle\varepsilon\rangle$ und $B = \langle b_1, \ldots, b_k\rangle$ für $k = 1, 2, 3, \ldots$

Die dynamische Semantik ist durch folgende Funktion δ gegeben:

1. Bei Ankünften an einer PS-Maschine mit mindestens einem freien Bediener wird das neuankommende Element einem freien Bediener zugeordnet, der es mit der spezifizierten Geschwindigkeit bedient, formal

   $\delta((\langle\varepsilon\rangle, \langle b_1, \ldots, b_k\rangle, A), An(r_i, t_{now})) = (\langle\varepsilon\rangle, \langle b_1, \ldots, b_k, b_{k+1}\rangle, A)$ mit $b_{k+1} = \langle r_i, t_{now}, t_{now}, \omega\rangle$ und $r_i = \langle ms_i, \omega, prior(r_i)\rangle$.

   Gleichzeitig ergibt sich das zugehörige Ereignis Bedienende zu
   $Be(b_{k+1}, t_{now} + \omega/\alpha_i)$

2. Bei Ankunft an einer Maschine mit mindestens n Dienstelementen kommt das neue Dienstelement in Bedienung. Allerdings müssen alle in Bedienung befindlichen Elemente ihre neue Restbedienzeit berechnen, die Bediengeschwindigkeit wird reduziert und alle Bedienende-Ereignisse sind neu zu berechnen, formal

   $\delta((\langle\varepsilon\rangle, \langle b_1, \ldots, b_k\rangle, A), Be(e_l, t_E)) = (\langle\varepsilon\rangle, \langle b_1, \ldots, b_{l-1}, b_{l+1}, \ldots, b_n\rangle, A')$

   mit $A' = \langle(ms_1, \alpha'_1), \ldots, (ms_k, \alpha'_k)\rangle$, $\alpha'_i = \frac{n \cdot \alpha_i}{k+1}$, $b_{k+1} = \langle r_i, t_{now}, t_{now}, \omega\rangle$ und $r_i = \langle ms_i, \omega, prior(r_i)\rangle$. Weiterhin gilt:

   $\forall b_i \in B \quad i \neq l$:

   (a) $t_{Rest} = t_{Rest} - \left(\frac{t_{now} - t_{sched}}{\alpha_i}\right)$

   (b) $t_{sched} = t_{now}$

   (c) neues Bedienende $Be\left(b_i, t_{now} + \frac{t_{sched}}{\alpha'_i}\right)$.

   Das Bedienende des neuankommenden Kunden ergibt sich analog zu

   $Be\left(b_{k+1}, t_{now} + \frac{t_{sched}}{\alpha'_i}\right)$.

d. Processor Sharing

3. Bei Kundenabgängen an Maschinen mit mindestens n+1 Kunden müssen alle Restbedienzeiten ermittelt werden, dann wird die neue Geschwindigkeit errechnet und die Bedienende-Ereignisse aller Elemente im Bedienraum müssen aktualisiert werden. Formal ausgedrückt

   $\delta((\langle\varepsilon\rangle, \langle b_1, \ldots, b_k\rangle, A), Be(e_l, t_E)) = (\langle\varepsilon\rangle, \langle b_1, \ldots, b_{l-1}, b_{l+1}, \ldots, b_n\rangle, A')$

   mit $A' = \langle(ms_1, \alpha'_1), \ldots, (ms_k, \alpha'_k)\rangle$, $\alpha'_i = \frac{n \cdot \alpha_i}{k-1}$.

   Weiterhin gilt:

   $\forall b_i \in B \quad i \neq l$

   (a) $t_{Rest} = t_{Rest} - \left(\frac{t_{now} - t_{sched}}{\alpha_i}\right)$

   (b) $t_{sched} = t_{now}$

   neues Bedienende $Be\left(b_i, t_{now} + \frac{t_{sched}}{\alpha'_i}\right)$.

4. Bei PS-Maschinen mit höchstens n Elementen im Bedienraum wird der Bedienraum um ein Element reduziert, formal

   $\delta((\langle\varepsilon\rangle, \langle b_1, \ldots, b_k\rangle, A), Be(e_i, t_E)) = (\langle\varepsilon\rangle, \langle b_1, \ldots, b_{i-1}, b_{i+1}, \ldots, b_n\rangle, A)$

Damit sind alle in QSDL definierten Maschinen formal beschrieben.

A.8 Semantik von request-Aufrufen

Das Absetzen eines request an eine Maschine ist eine weitere Aktion, die in einer Transition von Prozessen durchgeführt werden kann.

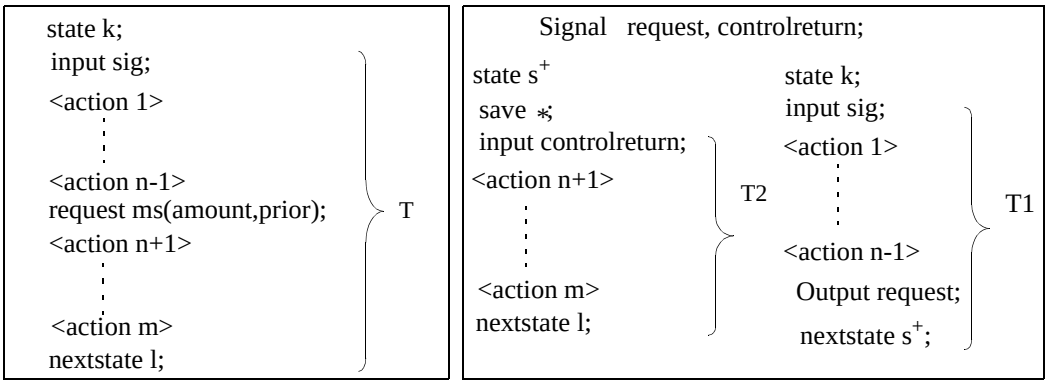


Abbildung A-1 **Request in SDL-Semantik**

Der aufrufende Prozeß wird für die Zeit, die benötigt wird, um die Dienstanforderung zu bearbeiten, blockiert, daß heißt, er kann keine anderen Aktionen ausführen. Alle während dieser Zeit eintreffenden Signale werden in dem Eingangspuffer gespeichert. Formal kann man die Semantik von request-Aufrufen auf die SDL-Semantik zurückführen, vgl. Abbildung A-1, rechts.

Für jede request-Aktion, die innerhalb einer Transition auftritt, werden die Pseudo-Signale *request* und *controlreturn* definiert, die den Anfang bzw. das Ende eines request aus Prozeßsicht

anzeigen. Außerdem wird ein neuer Zustand s^+ definiert. In diesem Zustand befindet sich der Prozeß, solange die Diensterbringung durch die Maschine nicht abgeschlossen ist. Die ursprüngliche Transition *T* wird durch die beiden Transitionen *T1* und *T2* ersetzt. *T1* enthält alle Aktionen von *T* bis zur request-Aktion. Diese Aktion wird ersetzt durch *Output request* und *nextstate* s^+. *T2* enthält alle Aktionen aus *T*, die nach der request-Aktion durchgeführt wurden. In s^+ werden alle Signale gesichert. Lediglich das Pseudo-Signal *controlreturn* kann konsumiert werden und triggert den Rest der ursprünglichen Transition.

Dieses Schema kann rekursiv auf alle Teiltransitionen angewendet werden, bis die ursprüngliche Transition vollständig ausgeführt worden ist.

A.9 Semantik des verzögerten Output

Die Angabe eines Wertes vom Typ duration im QSDL-Konstrukt output-delay führt zur zeitlichen Entkopplung von Sende- und Empfangsereignissen beim Signalaustausch zwischen (Q)SDL-Prozessen. Um die Auswirkungen formalisieren zu können, müssen erst einige vorbereitende Notationen eingeführt werden.

Mit $SIG = \{s_1, ..., s_k\}$ wird die Menge aller in einem SDL-System definierten Signale und Timer-Signale bezeichnet. Die Menge $SIG_{P_i} \subseteq SIG$ enthält alle Signale und Timer-Signale im Gültigkeitsbereich von Prozeß P_i.

Die Menge $V_j = \{v_{j1}, ..., vj_t\}$ enthält alle lokalen Variablen des Prozesses P_j und mit $dom(v_c)$ wird der Wertebereich der Variablen v_c bezeichnet.

Der Zustand z_P eines SDL-Prozesses, der auf asynchron kommunizierenden, erweiterten, endlichen Automaten basiert kann als 3-Tupel $z_P = (z_H, z_N, W)$ beschrieben werden. Dabei ist:

$z_H \in \{z_{H1}, ..., z_{Hm}\}$		Menge der Hauptzustände (SDL-States)
$z_N = \langle z_{N1}, ..., z_{Nr} \rangle$	$z_{nj} \in dom(j)$	Wertetupel der lokalen Variablen
$W = \langle s_1, ..., s_p \rangle$	$s_i \in SIG_h$	Nachrichten im Eingangspuffer

Ein SDL-Prozeß kann nun analog zu einem endlichen Automaten aufgefaßt werden als 5-Tupel $P_i = (SIG_{P_i}, SIG_{P_i}, Z, \delta, \alpha)$ wobei

$SIG_{P_i} \subseteq SIG$	Menge der gültigen Signale
$SIG_{P_i} \subseteq SIG$	Menge der gültigen Signale
$Z = \{z_{p1}, ..., z_{ph}\}$	Menge der Zustände, die der Prozeß einnehmen kann
δ: $Z \times SIG_{P_i} \rightarrow Z$	Zustandsübergangsfunktion
α: $Z \times SIG_{P_i} \rightarrow SIG_{P_i} \times \Im$	Ausgabefunktion

Mit $\Im = \{Z_1, ..., Z_u\}$ wird die Menge aller Mengen von Zuständen aller Prozesse eines Systemes bezeichnet. Im Gegensatz zu erweiterten, endlichen Automaten hat die Ausgabefunktion bei kommunizierenden Automaten einen Einfluß auf den Zusatnd der empfangenden Automaten, da

die gesendeten Signale direkt in die Eingangspuffer der Empfänger gelangen, vgl. Abbildung 2-2 auf Seite 19. Für die Zustandsübergangsfunktion δ gilt:

$\delta(z_{Hk}, \langle z_{N1}, ..., z_{Np}\rangle, \langle s_1, ..., s_q\rangle, s_1) = (z_{Hk+1}, \langle \tilde{z}_{N1}, ..., \tilde{z}_{Np}\rangle, \langle s_2, ..., s_q\rangle)$ falls das Signal s_1 in Zustand z_{Hk} eine Transition triggert, die mit *nextstate* $z_{\mathrm{hk+1}}$ endet und in der Aktionen definiert sind, die den Variablen v_1 bis v_p die Werte $(\tilde{z}_{N1}, ..., \tilde{z}_{Np})$ zuordnet. Für jede Transition, in der mindestens eine output-Aktion vorkommt, gilt:

$\alpha(z_{Hk}, \langle z_{N1}, ..., z_{Np}\rangle, \langle s_1, ..., s_q\rangle, s_1) =$

$(s_r, (z_{H_{1r}}, z_{N_{1p}}, \langle s_{11}, ..., s_{1t}\rangle), ..., (z_{H_{lr}}, z_{N_{lp}}, \langle s_{l1}, ..., s_{ls}, s_r\rangle), ..., (z_{H_{ur}}, z_{N_{up}}, \langle s_{u1}, ..., s_{us}\rangle))$,

falls das Signal s_1 eine Transition triggert, in der das Signal s_r an Prozeß P_l geschickt wird.

Mit der eingeführten Notation kann nun die Auswirkung des output-delay formalisiert werden. Die Ausgabefunktion α erhält als weiteren Parameter, den Zeitpunkt t, zu dem die zugehörige Transition T ausgeführt wird und es wird das zusätzliche Ereignis ***Empfang Signal S*** in QSDL-Systemen eingeführt.

Für jede Transition, in der output-delay vorkommt, wird die entsprechende Gleichung in α geändert. Damit wird α^+: $Z \times \Re_0^{plus} \times SIG_{P_i} \rightarrow SIG_{P_i}$ und gleichzeitig wird das Ereignis $ES(SIG_{P_j}, t_E)$ erzeugt.

Die Zustandsübergangsfunktion δ erhält ebenfalls einen Zeitparameter und weiterhin können Zustandsänderungen auch durch die ES_i ausgelöst werden, allerdings haben die ES_i nur Auswirkungen auf die Eingangswarteschlange W.

Sei *T* eine Transition eines SDL-Prozesses, in der eine output-delay Aktion vorkommt. Der Delay-Parameter sei Δt und *T* werde zum Zeitpunkt t_t ausgeführt. Dann gilt:

$\delta(z_{Hk}, \langle z_{N1}, ..., z_{Np}\rangle, \langle s_1, ..., s_q\rangle, s_1, t_t) = (z_{Hk+1}, \langle \tilde{z}_{N1}, ..., \tilde{z}_{Np}\rangle, \langle s_2, ..., s_q\rangle)$ und weiterhin:

$\alpha(z_{Hk}, \langle z_{N1}, ..., z_{Np}\rangle, \langle s_1, ..., s_q\rangle, t_t, s_1) =$

$(s_r, (z_{H_{1r}}, z_{N_{1p}}, \langle s_{11}, ..., s_{1t}\rangle), ..., (z_{H_{lr}}, z_{N_{lp}}, \langle s_{l1}, ..., s_{ls}\rangle), ..., (z_{H_{ur}}, z_{N_{up}}, \langle s_{u1}, ..., s_{us}\rangle))$

Es wird das Ereignis $ES(s_r, t_t + \Delta t)$ erzeugt und die Zustandsübergangsfunktion des Empfängers ändert sich in $\delta(z_{Hk}, \langle z_{N1}, ..., z_{Np}\rangle, \langle s_1, ..., s_q\rangle, ES(s_r), t) = (z_{Hk}, z_{N1}, ..., z_{Np}, \langle s_1, ..., s_q, s_r\rangle)$

A.10 Semantik des awake-Konstruktes

Mit Hilfe des awake-Konstruktes ist eine Art Interrupt-Mechanismus geschaffen worden, mit dessen Hilfe Reaktionen von Prozessen zu vordefinierten Zeitpunkten ermöglicht werden. Gleichzeitig werden spontane Transitionen, die in jedem Prozeß möglich sind, für die Zeit deaktiviert, die mittels awake festgelegt wurde. Normale Signal-getriggerte Transitionen können unbeeinflußt weiter schalten. Damit ist der Forderung der Z.100 genüge getan, die keinerlei Zusammenhang oder Prioritäten zwischen spontanen Transitionen und signalkonsumierenden Transitionen vorsieht. Falls zu einem Zeitpunkt, zu dem spontane Transitionen auch signalkonsumierende Transitionen möglich sind, wird zufällig unter allen möglichen Transitionen eine ausgewählt. Dabei ist die Wahrscheinlichkeit gewählt zu werden, für alle Transitionen gleich. Wie bei den request-Aufruf kann die formale Semantik auf die SDL-Semantik zurückgeführt werden.

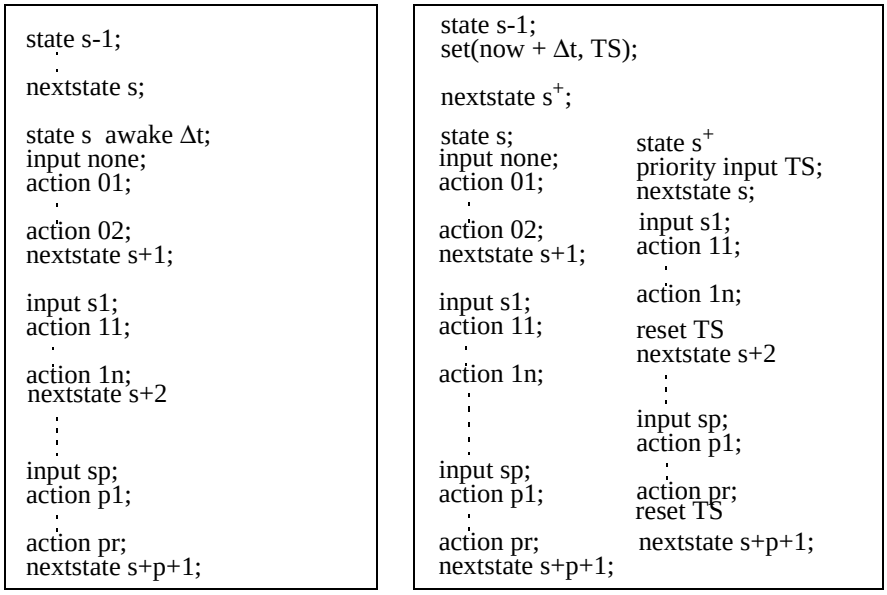


Abbildung A-2 **Awake in SDL-Semantik**

Für jeden Zustand s, in dem ein awake spezifiziert ist, wird ein neuer Zustand s^+ definiert, in dem alle Transitionen möglich sind, die auch in s möglich sind. Zusätzlich wird eine Transition definiert, die das Timersignal, das den Ablauf der awake-Zeitspanne signalisiert, priorisiert konsumiert. Das Konsumieren des Timersignals überführt den Prozeß wieder in den Originalzustand s, in dem aber jetzt kein awake spezifiziert ist. Nun stehen alle Transitionen inklusive der Spontanen zur Verfügung, die auch vorher aktiviert waren.

Wird der Zustand s^+ durch eine signalkonsumierende Transition verlassen, so wird auch der Timer gelöscht. Damit ist die Semantik aller Ergänzungen, die QSDL gegenüber SDL bietet, formal beschrieben.

Anhang B QSDL-Spezifikationen

In der beiliegenden CD befinden sich alle im Rahmen dieser Arbeit entwickelten QSDL/PR-Beschreibungen in ASCII-Format.