TECHNISCHE
UNIVERSITÄT
MÜNCHEN

INSTITUT FÜR INFORMATIK

Sonderforschungsbereich 342:
Methoden und Werkzeuge für die Nutzung
paralleler Rechnerarchitekturen

Evaluierung der Leistungsfähigkeit eines ATM-Netzes mit parallelen Programmierbibliotheken

Tobias Müller, Stefan Lamberts, Ursula Maier und Georg Stellner

TUM-I9701
SFB-Bericht Nr.342/1/97 A
Januar 1997

Vorwort

Im SFB 342 ist der Lehrstuhl für Rechnertechnik und Rechnerorganisation (LRR-TUM) für die Bereitstellung grundlegender Systemumgebungen und Werkzeuge durch das Teilprojekt A1 zuständig. In diesen Rahmen fügt sich die vorliegende Arbeit ein, die als Fortgeschrittenenpraktikum von Tobias Müller durchgeführt wurde. Sie erklärt den Einsatz neuartiger und leistungsfähiger Netztechnologien zur Parallelverarbeitung auf Netzen aus Arbeitsplatzrechnern und untersucht die Tragfähigkeit dieser Vorgehensweise. Zentralen Gesichtspunkt hierbei bildet die eingehende Untersuchung erzielbarer Bandbreiten und Latenzzeiten. Die durchgeführten Messungen bilden schließlich die Grundlage zur Ableitung von Richtlinien zur Optimierung der Kommunikation aus Systemsicht. Die Arbeit dient damit innerhalb des Querschnittsprojekts „Universell programmierbare Architektur und Basis-Software“ (UPAS) sowohl als Entscheidungshilfe über den Einsatz ATM-vernetzter Systeme als auch zur Unterstützung bei der Verbesserung der Leistungsfähigkeit von Anwendungen, die auf ATM-Clustern laufen.

München, Januar 1997

Georg Stellner

Inhalt

1 Einleitung und Motivation **1**

2 Einführung in ATM **3**

2.1 Eigenschaften und Möglichkeiten von ATM 3

2.2 Zellen, virtuelle Pfade und Kanäle 4

2.3 ATM Schichtenmodell 5

2.3.1 Physikalische Schicht 6

2.3.2 ATM-Schicht - Zellaufbau 7

2.3.3 ATM-Anpassungsschicht AAL 8

2.4 Beschreibung des ATM-Netz am LRR-TUM 10

3 FORE-API **13**

3.1 Adressierung, Verbindungsaufbau und Datentransport 13

3.2 Beschreibung der API-Aufrufe 14

4 PVM für ATM **19**

4.1 Eignung von ATM für parallele Programmierumgebungen 19

4.2 Kommunikationspfade innerhalb von PVM 20

4.3 Nutzung des Fore-API durch PVM 22

5 Leistungsmessungen im ATM-Netz **25**

5.1 Technik der Meßprogramme 26

5.1.1 Durchsatzmessungen 26

5.1.2 Messung von Latenzzeiten 27

5.2 Durchsatzmessungen mit `nttcp` 28

5.2.1 Test 1: TCP-Stream (ATM versus Ethernet) 29
5.2.2 Test 2: TCP-Stream (ATM: Variable Socket-Puffergrößen) 29
5.2.3 Test 3: UDP-Datagramme (ATM versus Ethernet) 31
5.2.4 Test 4: Fore-API (AAL 3/4, AAL 5) 32
5.2.5 Zusammenfassung 34
5.3 HP-Benchmark: netperf 34
5.3.1 Test 5: TCP-Stream-Durchsatz (hpbode3/4) 35
5.3.2 Test 6: TCP-Stream-Durchsatz (ibode2/4) 35
5.3.3 Test 7: Fore-API (AAL 3/4, AAL 5) 37
5.3.4 Test 8, 9: Fore-API (AAL 5): Sender- und Empfängerdurchsatz 37
5.3.5 Test 10 - 13: Latenzzeitmessungen mit netperf 40
5.4 Der PVM-Benchmark 44
5.4.1 Funktionsweise des PVM-Benchmarks 44
5.4.2 Test 14 – 17: Durchsatzmessungen mit dem PVM-Benchmark 45
5.4.3 Test 18 – 21: Latenzzeitmessungen mit dem PVM-Benchmark 47

6 Optimierungsmöglichkeiten für TCP/UDP über ATM 53
6.1 Paketdatenfluß 54
6.2 Optimierungsparameter der Protokollschichten 55
6.3 AIX: Verwaltung des mbuf-Pools 57

7 Zusammenfassung 61

A Aufrufparameter von nttcp 63

B Testkonfigurationen: nttcp 65

C Testkonfigurationen: netperf 67

D Konfiguration des IBM TurboWays ATM-Adapter 71

Literatur 73

1

Einleitung und Motivation

Parallele Programmierumgebungen schaffen die Voraussetzungen, eine Anzahl heterogener Rechner zur Problemlösung bei sehr rechenintensiven Aufgaben heranzuziehen. Hierbei werden die Rechner, die über ein Kommunikationsnetz miteinander verbunden sind, als *eine* Ressource genutzt. Die Rechenleistung, die die kombinierte CPU-Kapazität und der gesamte Hauptspeicher eines solchen Workstation-Cluster bieten, kann die eines Parallelrechners oder Supercomputers sogar übertreffen. Gegenüber der sehr teuren Anschaffung eines Parallelrechners hat der Einsatz von vernetzten Workstations mit parallelen Programmierumgebungen den Vorteil, daß die Rechner und das Netz meist schon vorhanden sind und beispielsweise nachts oder an Wochenenden ungenutzt sind [21]. Die Kommunikation zwischen den parallelen Prozessen stellt hohe Anforderungen an die Leistungsfähigkeit des Netzes, die in traditionellen lokalen Netzen wie z.B. beim Ethernet sehr begrenzt ist.

Der **Asynchrone Transfer Modus – ATM** ist gegenwärtig die aufstrebende Netztechnologie. Es verspricht, das universelle Netz für alle Kommunikationsanwendungen im Nahverkehrsbereich (LAN) wie auch im Weitverkehrsbereich (WAN) zu werden. In lokalen Datennetzen wird die Netzgeschwindigkeit seit vielen Jahren vom weit verbreiteten Ethernet bestimmt. Alle Endsysteme eines Netzsegments konkurrieren hier um die zur Verfügung stehende Bandbreite von 10 MBit/s. ATM bietet dagegen sehr hohe Bandbreiten (z.B. 155 MBit/s), die Endsystemen dediziert zur Verfügung stehen, bei geringen Latenzzeiten. Mit diesen Eigenschaften könnte ATM verteiltes Rechnen und den Einsatz von parallelen Programmierumgebungen wie PVM um vieles effizienter machen.

Die **Leistungsfähigkeit** von ATM wird derzeit meist nur als Funktion der Bandbreite der Hardware betrachtet. Interessant ist aber, wieviel Leistung eine Anwendung erhält, die über die Standardmechanismen für Internetworking wie TCP/IP Datenaustausch über ATM betreibt. Wie gut sind die noch sehr jungen Implementierungen der Hersteller für *IP over ATM*? Lohnt sich der Einsatz von neuen proprietären Kommunikationsschnittstellen wie dem Fore-API, die für ATM entwickelt wurden? Welche Leistungssteigerung erfährt die parallele Programmierumgebung PVM bei der Kommunikation über ATM? Mit diesen Fragestellungen beschäftigt sich diese Arbeit. Dazu wurde die Leistung des ATM-Netzes am Lehrstuhl für Rechnertechnik und Rechnerorganisation der Technischen Universität München (LRR-TUM) unter den Gesichtspunkten Durchsatz und Latenzzeiten bei verschiedenen Protokollkombinationen analysiert.

Kapitel 2 dieser Ausarbeitung gibt eine kurze Einführung in die Funktionsweise von ATM und beschreibt das lehrstuhleigene ATM-Netz. In Kapitel 3 wird die von Fore entwickelte Programmierschnittstelle für ATM, das Fore-API, vorgestellt. Das vierte Kapitel beschreibt die IP-basierten Kommunikationsmechanismen der parallelen Programmierumgebung PVM und eine Version von PVM, die auf dem Fore-API aufsetzt. In Kapitel 5 finden sich die Ergebnisse der Leistungsmessungen auf dem ATM-Netz, die mit den Programmen nttcp, netperf und einem PVM-Benchmark durchgeführt wurden. Kaptitel 6 beschäftigt sich mit möglichen Optimierungen von Parametern für IP, UDP, TCP und BSD-Sockets über ATM. Kapitel 7 faßt die Arbeit zusammen.

2

Einführung in ATM

2.1 Eigenschaften und Möglichkeiten von ATM

Der Asynchrone Transfer Modus **(ATM)** ist ein Übertragungsverfahren, welches mit Zellen fester Länge (53 Bytes) und asynchronem Zeit-Multiplexing arbeitet. Im Gegensatz zum Bus beim Ethernet verwendet ATM eine kollisionsfreie sternförmige Topologie. Beim ATM handelt es sich um eine verbindungsorientierte Kommunikation, dessen Verbindungen Kanäle genannt werden, in denen die Zellen zwischen zwei Endpunkten auf dem Weg durch das ATM-Netz von den Netzknoten (Switches) vermittelt werden. Im Gegensatz zum synchronen Zeit-Multiplexing, bei dem der Benutzer zur Übertragung von Daten einen festen Zeitabschnitt innerhalb eines Übertragungsrahmens erhält, ist beim ATM-Multiplexing jede Zelle mit einer Kanal-Identifikationsnummer versehen, die die Zelle einer Benutzerverbindung zuordnet. Die insgesamt zur Verfügung stehende Übertragungsbandbreite kann so flexibel (*bandwidth on demand*) an die Benutzer verteilt werden, andererseits ist auch die Reservierung einer festen Übertragungsbandbreite für eine Benutzerverbindung möglich. Dadurch können in einem einzigen ATM-Netz Dienste mit unterschiedlichen Anforderungen an folgende Service-Parameter integriert werden.

- Verbindungsart (verbindungsorientiert / verbindungslos)
- Nachrichtenlaufzeiten (*delay*) und Gleichlaufschwankungen (*jitter*)
- Bedarf an Bandbreite (gering / hoch)
- Bitrate (variabel / konstant)

Audio- und Videoübertragung, sowie Anwendungen aus dem Multimediabereich, erfordern hohen Datendurchsatz und Realzeitbedingungen. Durch die Reservierung von Bandbreite werden die isochronen Ströme von Echtzeitanwendungen nicht durch Lastspitzen (*bursts*) des übrigen asynchronen Verkehrs beeinträchtigt.

Die **ATM-Zelle** setzt sich aus 48 Bytes Nutzinformation und 5 Bytes Header zusammen. Die geringe Größe der ATM-Zelle ist ein Kompromiß zwischen den Anforderungen bei der Sprachübertragung und der Datenübertragung (z.B. *file transfer*). Audio- und Videoübertragung erfordern kleine Verzögerungszeiten, um den Informationsfluß für Auge und Ohr nicht zu unterbrechen. Für die Übertragung großer Datenmengen (*bulk transfer*) sind größere Pakete geeigneter, um den Aufwand für das Zerteilen und Wiederzusammensetzen (*segmentation and reassembly*) gering zu halten.

Der Transport der ATM-Zellen ist nicht auf ein einziges physikalisches Leitungsmedium beschränkt. Ebenso ist die Geschwindigkeit, mit der die Zellen auf dem Medium übertragen werden, nicht festgelegt. Basierend auf optischen Leitungen (*SONET – Synchronous Optical NETwork*) werden mit dem Übertragunsverfahren SDH (Synchrone Digitale Hierarchie) der CCITT Durchsätze von 155 Mbit/s, 622 Mbit/s und mehr erzielt. ATM bietet somit auch *einen* Übertragungsstandard für LANs, MANs und WANs.

Zusammenfassung der Vorteile von ATM:

- Integration unterschiedlicher Dienste in *einem* Netz
- Flexible Bandbreitenzuordnung
- Skalierbare Übertragungsraten
- Zuverlässige Hochgeschwindigkeitsvermittlung

2.2 Zellen, virtuelle Pfade und Kanäle

In einem ATM-Netz gibt es keine physikalischen Kanäle um den Netzverkehr aufzuteilen. Um dennoch unterschiedliche Dienste wie Sprache, Video oder Daten unterscheiden zu können, werden nach Bedarf logische Verbindungen zwischen den Kommunikationsendpunkten aufgebaut. Diese logischen Verbindungen werden durch Tupel bestehend aus virtuellem Pfad (*VP – Virtual Path*) und virtuellem Kanal (*VC – Virtual Channel*) adressiert.

Der virtuelle Kanal ist ein Konzept, um unidirektionalen Transport von ATM-Zellen einer logischen Verbindung zu beschreiben. Jedem virtuellen Kanal wird ein eindeutiger Identifikator VCI (*Virtual Channel Identificator*) zugeordnet. Ein virtueller Pfad ist ein Bündel von virtuellen Kanälen und besitzt einen eindeutigen Identifikator VPI (*Virtual Path Identificator*). Abbildung 2.1 illustriert das Konzept der virtuellen Pfade und Kanäle.

Der Header einer ATM-Zelle enthält VPI und VCI, um die Zelle einer logischen Verbindung zuordnen zu können. Das Tupel VPI und VCI hat jeweils nur lokale Gültigkeit zwischen zwei benachbarten Knoten im Netzwerk. Knoten können hierbei Datenendeinrichtungen (z.B. Hosts) als auch ATM-Switches sein. Alle Zellen einer Ende-zu-Ende-Verbindung werden über den gleichen Pfad im Netz geleitet. Jede Verbindung besteht aus einer Abfolge von Strecken, die durch VPI und VCI eindeutig bestimmt sind. Eine Verbindung ist also durch eine Folge von Werten für VPI und VCI im Zellheader charakterisiert. Da die Folge für alle Zellen dieser Verbindung auf dem Weg durch das Netz gleich ist, ist auch die richtige Reihenfolge der Zellen gesichert.

Der Switch liest aus dem Header einer ankommenden Zelle die Werte für VPI/VCI und bildet diese mit Hilfe von Tabellen, die bei Verbindungsaufbau und -abbau aktualisiert werden, auf die neue Belegung ab. Die neuen Werte werden in den Zellheader geschrieben, anschließend wird die Zelle weitergeleitet. Gemäß der Hierarchie zwischen virtuellen Kanälen und virtuellen

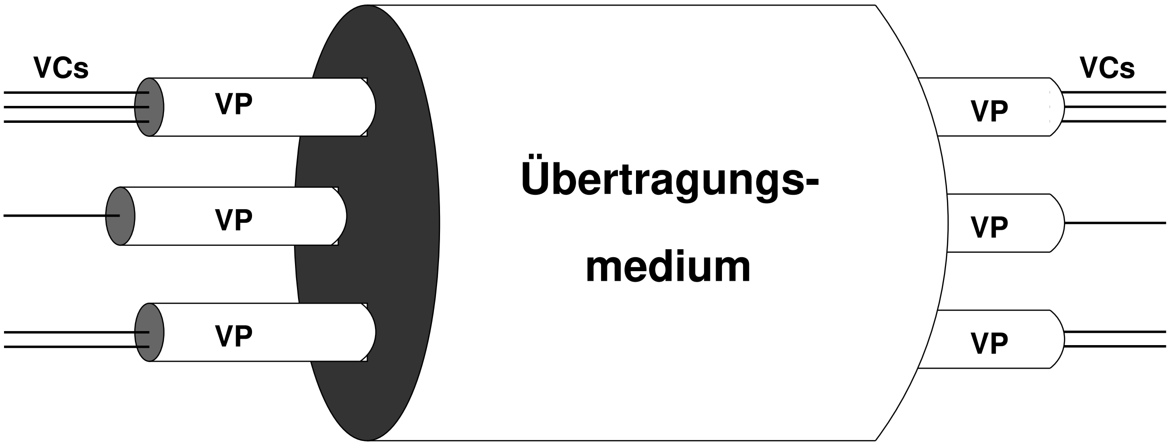

Abbildung 2.1: Beziehung zwischen virtuellen Pfaden und Kanälen

Pfaden wird das Multiplexing sowie das Switching der Zellen zuerst nach VPs und dann erst nach VCs gemacht.

Abbildung 2.2 zeigt ein ATM-Netz mit drei Switches und drei Rechnern. Zwischen Host A und Host B bestehen drei logische Verbindungen über die Switches 1 und 2. Beide Switches betreiben VP-Switching, d.h. nur die Pfadidentifikatoren werden verändert. Die Folge von VPIs einer Zelle von Host A nach Host B mit VCI=5 ist demnach 20, 32, 20. An diesem Beispiel sieht man auch die Lokalität von VPI/VCI: Auf den unterschiedlichen Strecken Host A — Switch 1 und Switch 2 — Host B wird das gleiche Tupel 20/5 für VPI/VCI verwendet. Switch 3 betreibt sowohl VP- als auch VC-Switching. Zellen einer Verbindung zwischen Host A und Host C, die Switch 3 mit VPI/VCI 48/3 erreichen, verlassen diesen mit VPI/VCI 16/9.

Zwischen zwei Endpunkten kann der Verwalter des Netzes permanente Verbindungen (*PVC – Permanent Virtual Circuit/Channel*) für dedizierten lange andauernden Datenaustausch zweier Stationen einrichten. Normalerweise werden die Verbindungen aber dynamisch nur bei Bedarf und nur für die Dauer der Datenübertragung hergestellt. Das entspricht dem Telefongespräch bei der Sprachkommunikation. Zum Aufbau der dynamischen Kanäle (*SVC – Switched Virtual Circuit/Channel*) wird ein Signalisierungsprotokoll (*signaling*) verwendet. Verbindungsaufbauwünsche werden vom Benutzer *outband*, d. h. über einen eigenen logischen Steuerkanal, an den nächsten Switch geschickt. Wird der Wunsch akzeptiert, erhält der Benutzer vom Switch eine Rückmeldung, welches Tupel VPI/VCI die soeben errichtete Verbindung besitzt.

2.3 ATM Schichtenmodell

Abbildung 2.3 beinhaltet das ATM-Schichtenmodell, welches auf der B-ISDN-Netzwerkarchitektur basiert. Das B-ISDN-Referenz-Modell definiert außer den Schichten drei Ebenen. Die Ebenen (Benutzerebene, Steuerebene und Managementebene) verbinden die vier Schichten des Protokollmodells.

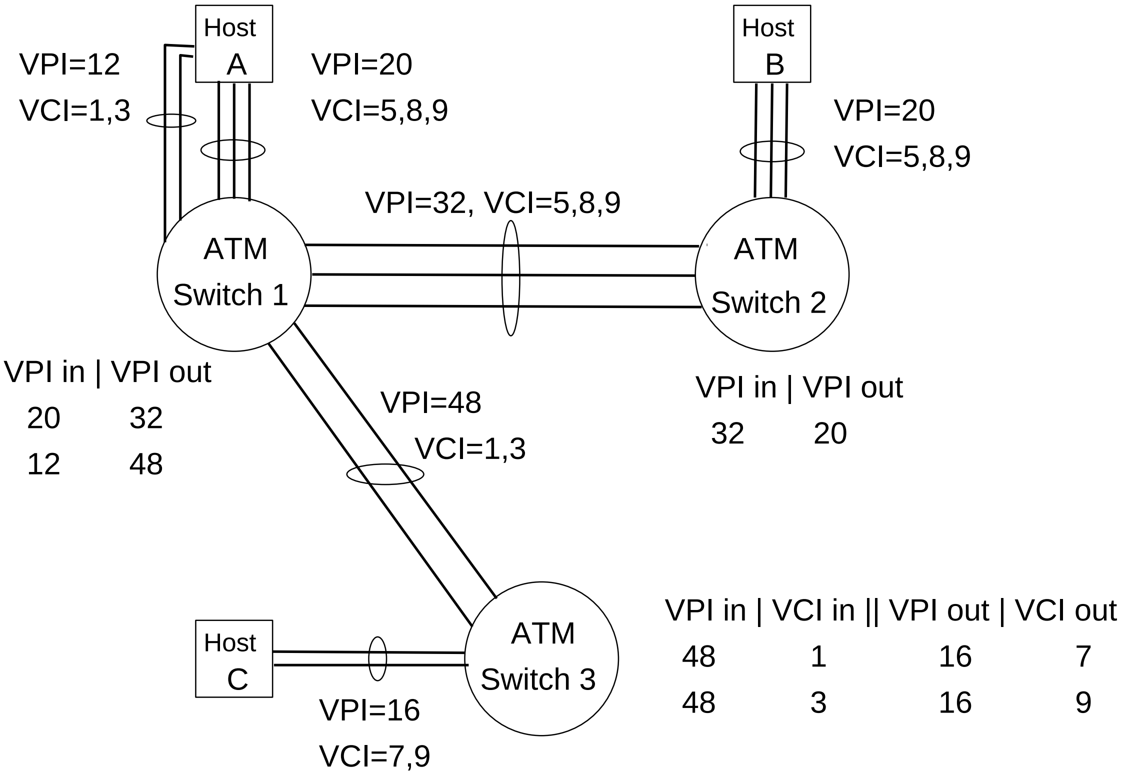

Abbildung 2.2: Virtuelle Pfade und Kanäle im ATM-Netz

Innerhalb der Benutzerebene bewegt sich der Datenfluß der Benutzer durch alle Schichten hindurch. Diese Ebene behandelt auch die Flußkontrolle sowie Übertragungsfehlererkennung und -korrektur.

Die Steuerebene baut u.a. ATM-Verbindungen auf und ab und aktualisiert Abbildungstabellen in Switches. Diese Funktionen sind im ATM-Signalisierungsverfahren zusammengefaßt, für das ein eigener logischer Signalisierungskanal zur Verfügung steht.

Die Managementebene ist unterteilt in Schichten- und Ebenenmanagement. Das Ebenenmanagement koordiniert die Funktionen aller drei Ebenen. Das Schichtenmanagement bietet Funktionen zur Überwachung der Netzleistung und zum Fehlermanagement. Für diese Funktionen gibt es spezielle OAM-Zellen (*operation and maintenance*).

Die Schichten des OSI-Referenzmodells vereint der Begriff „Höhere Schichten“ die absichtlich nicht genau spezifiziert sind, da ATM nicht eindeutig in das OSI-Modell eingefügt werden kann. Im folgenden werden die drei ATM-Schichten kurz beschrieben.

2.3.1 Physikalische Schicht

Da ATM-Zellen über unterschiedliche physikalische Medien übertragen werden können, besteht die Physikalische Schicht aus einer vom Medium abhängigen (*Physical Medium Dependend – PMD*) sowie aus einer unabhängigen Teilschicht (*Transmission Convergence Layer – TCL*). Die PMD-Teilschicht ist für die Bitsynchronisation auf dem physikalischen Medi-

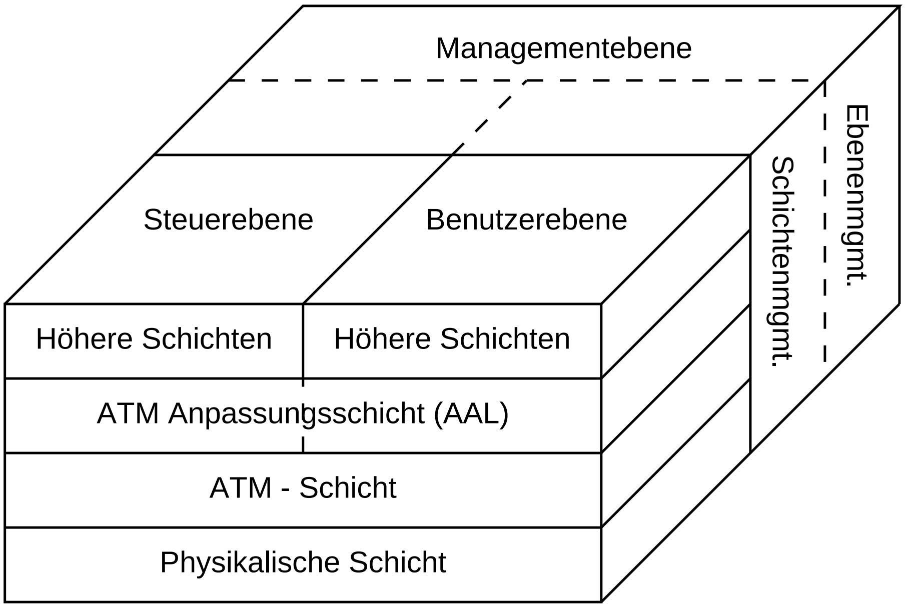

Abbildung 2.3: B-ISDN-Protokoll-Referenz-Modell

um verantwortlich. Die TCL-Teilschicht übernimmt die Übertragungsanpassung der von der ATM-Schicht generierten Zellen an das jeweils verwendete physikalische Medium. Prinzipiell können ATM-Zellen direkt über das physikalische Medium verschickt werden oder vorher in Übertragungsrahmen sog. *Frames* verpackt werden. Im TCL werden Übertragungsrahmen erzeugt und ATM-Zellen ein- bzw. ausgepackt, Zellgrenzen erkannt, Headerkontrollinformation zur Fehlererkennung ausgewertet und bei geringem Netzverkehr *Idle*-Zellen eingefügt.

2.3.2 ATM-Schicht - Zellaufbau

Die ATM-Schicht ist die zellverarbeitende Schicht und steht im Mittelpunkt von ATM. In ihr geschieht das Multiplexing und das Switching der Zellen. Zellen unterschiedlicher logischer Verbindungen werden zu einem kontinuierlichen Zellstrom zusammengefügt (Multiplexen), der an die physikalische Schicht weitergereicht wird. Das Demultiplexen ist der umgekehrte Vorgang, bei dem Zellen je nach logischer Verbindung und Empfänger aus dem Zellstrom herausgenommen werden. Um diese Aufgaben zu bewältigen, wertet die ATM-Schicht die Headerinformationen jeder Zelle aus. Abbildung 2.4 zeigt den Aufbau einer ATM-Zelle und des zugehörigen UNI-Zellheaders. An den Schnittstellen zwischen Benutzer und Netz (*User Network Interface – UNI*) kümmert sich die ATM-Schicht um die Generierung bzw. Beseitigung der Zellheader. Zwischen den ATM-Switches (*Network Network Interface – NNI*) sieht der Zellheader geringfügig anders aus. Es entfällt das GFC-Feld, welches am UNI das Verkehrsaufkommen beim Netzzugang steuert. Die Funktion der Felder VPI und VCI wurden bereits besprochen. Anhand des PT-Felds kann die ATM-Schicht feststellen, ob die betrachtete Zelle Benutzer- oder Netzmanagementinformationen enthält. Ist das CLP-Bit einer Zelle gesetzt, darf der ATM-Switch die Zelle bei Überlastung des Netzes verwerfen. Da der Informationsgehalt *einer* Zelle bei Audio- und Videoanwendungen relativ gering ist, kann ein Benutzer eine fehlende Zelle ohne weiteres tolerieren. Anders verhält es sich z.B. beim File-

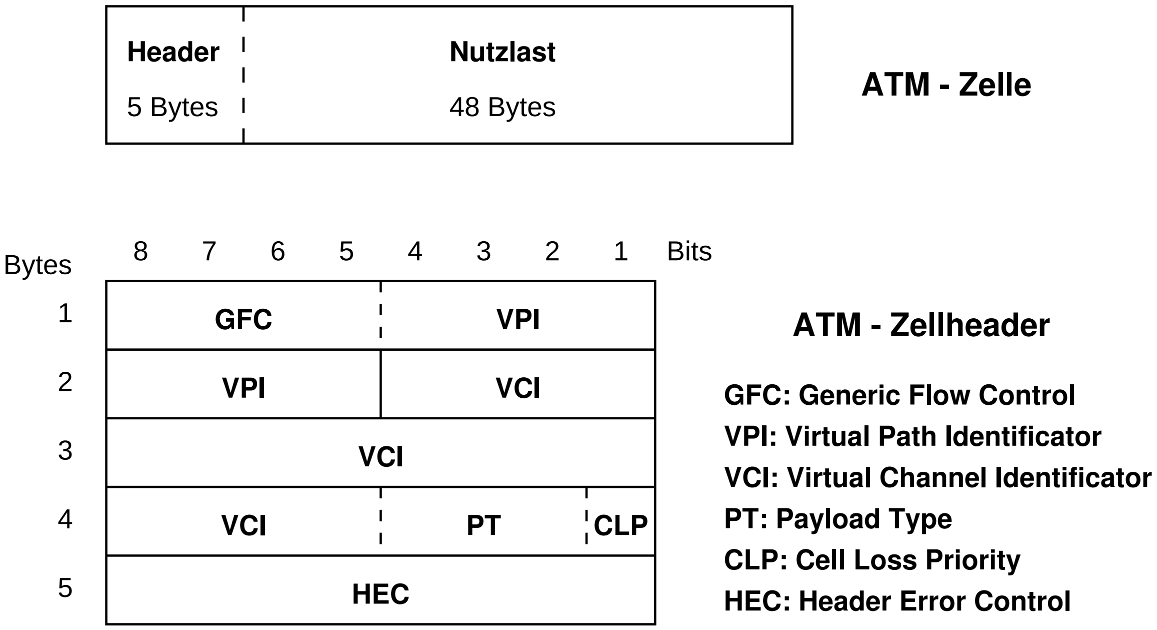

Abbildung 2.4: Aufbau einer ATM-Zelle

transfer, wo eine fehlende Zelle die wiederholte Versendung eines ganzen Pakets erforderlich macht.

Ferner ist die ATM-Schicht für die Überwachung der Dienstgüteparameter (*Quality of Service*) verantwortlich, die der Benutzer beim Aufbau seiner Verbindung festlegen muß. Falls die Netzbelastung die Erfüllung einer angeforderten Qualität nicht ermöglicht, wird der Aufbau verweigert. Bislang sind drei Parameter definiert:

- *Peak cell rate* ist die maximale Anzahl von ATM-Zellen pro Zeiteinheit, die der Endbenutzer am UNI übergeben darf.
- *Sustained cell rate* ist die durchschnittliche Zellrate über eine längere Zeitperiode.
- *Maximum burst size* ist das größte Datenpaket, welches das UNI akzeptiert.

Darüberhinaus gibt es weitere Serviceparameter für bestimmte Anwendungen (beispielsweise Zellverzögerung für Echtzeitübertragung), die beim Verbindungsaufbau durch Angabe einer gewünschten Serviceklasse festgelegt werden und die ebenfalls von der ATM-Schicht überwacht werden müssen. Die Serviceklassen sind in der ATM-Anpassungsschicht definiert.

2.3.3 ATM-Anpassungsschicht AAL

Die ATM-Anpassungsschicht (*ATM Adaption Layer – AAL*) stellt die Verbindung zwischen den Diensten in den höheren Schichten und der zellbasierten ATM-Schicht her. Hier erfolgt die Abbildung der Datenstrukturen der Anwendungsschichten (Audio, Video, Datentransfer) auf die einheitliche Zellenstruktur von ATM. Da die Anwendungen unterschiedliche Anforderungen bezüglich Zeitkompensation (z.B. Echtzeitbedingungen), Bitrate und Verbindungsmodus

Parameter / Serviceklasse	Klasse A	Klasse B	Klasse C	Klasse D
Zeitkompensation	erforderlich		nicht erforderlich	
Bit-Rate	konstant	variabel		
Verbindungsmodus	verbindungsorientiert			verbindungslos
Beispiel	Circuit-Emulation	Audio, Video	Datentransfer conn. orient.	Datentransfer conn. less
AAL-Typ	AAL 1	AAL 2	AAL 3 AAL 5	AAL 4

Abbildung 2.5: Serviceklassen und AAL-Typen

an das Kommunikationsnetz stellen, definiert die ITU-T vier Serviceklassen mit zugehörigen AAL-Typen:

- Klasse A: Audio/Video isochron mit konstanter Bitrate
- Klasse B: Audio/Video mit variabler Bitrate (komprimiert)
- Klasse C: Verbindungsorientierter Datentransfer
- Klasse D: Verbindungsloser Datentransfer

Abbildung 2.5 zeigt die Zuordnung von AAL-Typen zu Serviceklassen.

Die AAL-Schicht setzt sich aus den Teilschichten *SAR – Segmentation And Reassembly* und *CS – Convergence Sublayer* zusammen. Die SAR-Schicht zerlegt auf der Senderseite die Datenstrukturen der Anwendungen in kleine Pakete, die als Payload von den Zellen transportiert werden, und setzt sie beim Empfänger wieder zusammen. Die CS-Schicht stellt sicher, daß die Anforderungen der Anwendung an die *Quality of Services* vom erzeugten Zellstrom eingehalten werden.

Da sich AAL 3 und AAL 4 nur in der Benutzung eines 10 Bit langen Felds unterscheiden, wurden beide zu AAL 3/4 zusammengefaßt. Die Information in diesem Feld wird bei AAL 4 benötigt, um die Zellen dem richtigen Empfänger zuordnen zu können, da die Zellen aller verbindungslosen Übertragungen über den gleichen festgelegten VC transportiert werden. Beim verbindungsorientierten AAL 3 entfällt dieses Feld. AAL 3/4 weist aber einen für den Datentransfer gravierenden Nachteil auf. Der normale Overhead einer ATM-Zelle beträgt 10.4% (5 Bytes Header respektive 48 Bytes Benutzerinformation). Bei AAL 3/4 ist für jede SAR-Transporteinheit (SAR-PDU, 48 Bytes) ein 2 Byte langer Header und Trailer erforderlich. Dadurch verringert sich die Größe der Benutzerinformation pro Zelle auf 44 Bytes. Der Overhead beträgt dann 20.5% was für Datentransfer kaum akzeptabel ist. Aus diesem

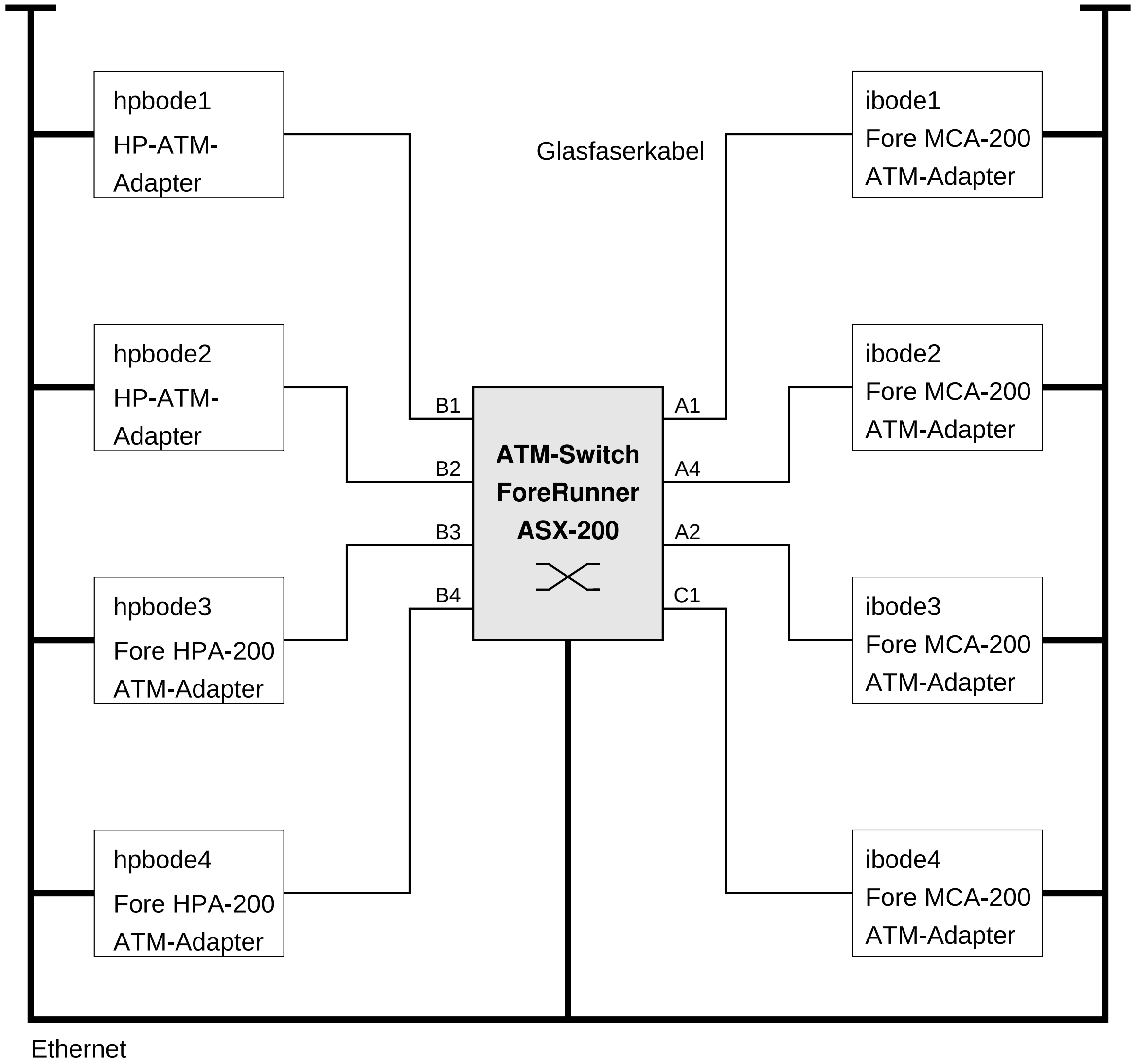

Abbildung 2.6: ATM-Netz am LRR-TUM

Grund wurde AAL 5 (*simple and efficient adaption layer*) für verbindungsorientierte Klasse-C-Dienste entwickelt. Hier entfällt der durch Reihenfolgesicherung und Integritätsprüfung verursachte zusätzliche Overhead von AAL 3/4. In diesem Fall müssen die Protokolle der höheren Schichten diese Funktionen abdecken. Daher hat sich AAL 5 für den Transport von TCP/IP-Paketen über ATM durchgesetzt.

2.4 Beschreibung des ATM-Netz am LRR-TUM

Das lokale ATM-Netz des Lehrstuhls für Rechnertechnik und Rechnerorganisation (LRR-TUM), dargestellt in Abbildung 2.6, ist ein heterogener Verbund von Workstations, die sternförmig an den ATM-Switch angeschlossen sind.

Hardware: Im Mittelpunkt steht der ForeRunner ASX-200 ATM-Switch mit einer Gesamtleistung von 2.5 Gbps und 16 Anschlüssen (*ports*). Die maximale Übertragungsrate pro Port beträgt 155 Mbps. Auf dem Switch läuft die Software ForeThought Version 4.0. Die Workstations sind über Glasfaserkabel mit dem Switch verbunden. Es handelt sich um vier IBM RS6000/390 unter AIX 3.2.5 (ibode[1..4]) mit 64 MB Hauptspeicher und vier HP9000/735/125 unter HP-UX-9.05 (hpbode[1..4]) mit 32 MB Hauptspeicher. Die IBM-Maschinen sind mit ForeRunner Microchannel ATM-Adapterkarten ausgerüstet. Die Firmware der ATM-Karten hat die Version 2.2.0, die Treiber-Software die Version 2.3.2. Die zwei HP-Rechner hpbode1/2 haben HP-ATM-Adapterkarten, die anderen zwei HP-Rechner ForeRunner EISA Adapterkarten. Die Firmware der Fore-Adapter hat die Version 2.1.0, die Fore-Treiber haben die Version 3.0.3.

Physikalische Schicht: Die für Glasfaserkabel entworfene Übertragungsarchitektur *SONET (Synchronous Optical NETwork)* OC-3c setzt auf der physikalischen Schicht auf. Zwischen den Netzknoten bestehen synchrone Ströme von Übertragungsrahmen. Ein OC-3c-Rahmen ist 2430 Bytes groß und wird alle 125 Mikrosekunden übertragen. Daraus resultiert die Übertragungsbandbreite von 155,52 Mbit/s auf dieser Schicht. Für den Transport von ATM-Zellen stehen pro Rahmen 2340 Bytes zur Verfügung. Die übrigen 90 Bytes oder 3,8% sind der Section- bzw. Path-Overhead von *SONET*. Damit verringert sich die Bandbreite, die der ATM-Schicht zur Verfügung steht, auf 149,76 Mbit/s. Nähere Informationen zu *SONET* finden sich in [12] und [15].

Signalisierung: Zum Aufbau von dynamischen logischen Verbindungen (*switched virtual channel – SVC*) im ATM-LAN wird die von Fore entwickelte, proprietäre Signalisierung *SPANS – Simple Protocol for ATM Network Signaling* eingesetzt. Da die HP-ATM-Adapter *SPANS* nicht unterstützen, werden für die beiden Rechner hpbode1 und hpbode2 feste Punkt-zu-Punkt-Verbindungen (PVCs) zu den übrigen Rechnern benutzt.

Anwendungsschnittstellen: Benutzeranwendungen können über die zwei in Abbildung 2.7 dargestellten Schnittstellen über das ATM-Netz kommunizieren. Auf TCP/IP basierende Anwendungen funktionieren unverändert auch über ATM, da zu den Adapterkarten Device-Treiber für IP installiert sind. Darüberhinaus stellt Fore eine Programmierschnittstelle (API) zur ATM-Anpassungsschicht zur Verfügung, über die Verbindungen vom Typ AAL 3/4 und AAL 5 realisiert werden. Das Fore-API mit der Versionsnummer 2.3 wird im folgenden Kapitel beschrieben.

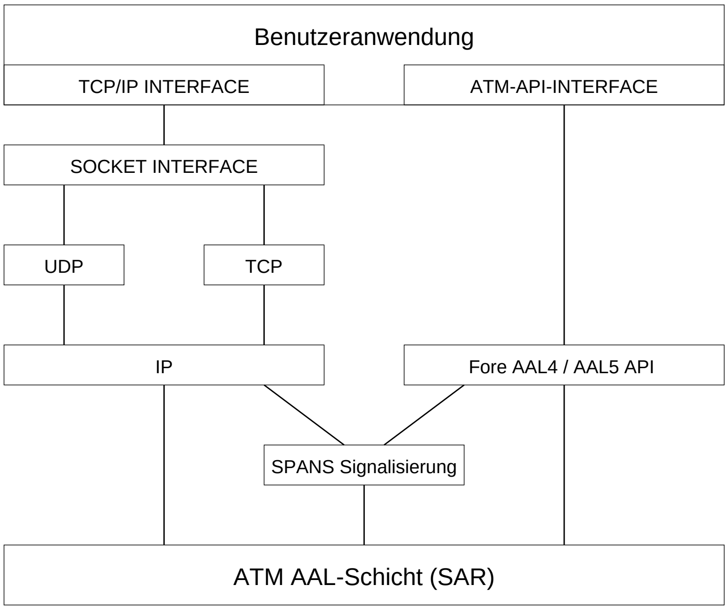

Abbildung 2.7: Schnittstellen für Benutzeranwendungen

3

FORE-API

Fore Systems stellt mit dem Fore-API eine portable Schnittstelle zur ATM-Anpassungsschicht für die Entwicklung von Anwendungen, die über ATM kommunizieren sollen, zur Verfügung. Das Fore-API enthält Bibliotheksroutinen für ein verbindungsorientiertes Client/Server-Modell. Bevor Daten zwischen zwei Kommunikationspartnern übertragen werden können, muß zwischen ihnen eine Verbindung aufgebaut werden. Daten werden nach bestem Bemühen (*best effort*) über das ATM-Netz transportiert. Die Flußkontrolle zwischen den Endsystemen sowie die wiederholte Übertragung eines fehlerhaften Pakets sind aber der Anwendung überlassen. Die Bibliotheksfunktionen sind von der Syntax her an die BSD-Socket-Schnittstelle von UNIX angelehnt. Nähere Informationen über das Fore-API enthalten die mitgelieferten *man pages* und [8].

3.1 Adressierung, Verbindungsaufbau und Datentransport

Adressierung: ATM-Endsysteme und Switches werden auf der ATM-Schicht durch eindeutige Dienstzugangspunkte (*NSAP – Network Service Access Point*) adressiert. Da jedes Endsystem an einem festen Port eines Switches angeschlossen ist, wird der NSAP über den Switch-Identifikator und die Portnummer gebildet. Die Anwendungen, die auf einem ATM-Endsystem laufen, erhalten zusätzlich Anwendungszugangspunkte (*ASAP – Application Service Access Point*). Ein ASAP ist innerhalb eines Endsystems eindeutig und wird vom ATM-Gerätetreiber an einen Filedeskriptor gebunden. Die Adresse für einen ATM-Endpunkt setzt sich aus NSAP und ASAP zusammen:

```
typedef u_int Atm_sap;       /* Service Access Point */

struct Atm_Adress {          /* Hardware Adress */
        char addr[8]; };
typedef struct Atm_Adress Atm_Adress;

struct Atm_endpoint {
        Atm_sap    asap;     /* Appl. Service Access Point */
        Atm_adress nsap; };  /* Network Service Access Point */
typedef struct Atm_endpoint Atm_endpoint;
```

Verbindungsaufbau: Verbindungen werden über das von Fore entwickelte, proprietäre Signalisierungsprotokoll SPANS (*Simple Protocol for ATM Network Signaling*) aufgebaut. Anwendungen öffnen zunächst einen Filedeskriptor (`atm_open()`), der mit einem ASAP und dem lokalen NSAP assoziiert wird (`atm_bind()`). Der Client stellt seinen Verbindungswunsch mittels `atm_connect()`, den der Server durch `atm_accept()` akzeptiert. Hierdurch werden auch die entfernten Dienstzugangspunkte (remote ASAP / NSAP) an den gleichen Filedeskriptor gebunden. Beim Verbindungsaufbau muß übereinstimmend von Client und Server angegeben werden, ob Daten nur in eine Richtung (*simplex*), in beide Richtungen (*duplex*) oder an mehrere Empfänger (*multicast*) übertragen werden sollen. Je nachdem werden der Anwendung (ASAP) eine, zwei oder mehrere logische Verbindungen zugeordnet, für die das SPANS dynamisch Tupel bestehend aus VPI und VCI vergibt. Das API sieht auch Datenstrukturen zur Anforderung der in Kapitel 2.3.2 beschriebenen Dienstgüteparameter (*Quality of Services*) vor, die bei Verbindungsaufbau durch die Signalisierung ausgehandelt werden. Diese werden aber derzeit vom proprietären SPANS nicht ausgewertet, was zur Folge hat, daß sich alle Verbindungen die zur Verfügung stehende Bandbreite teilen müssen. Eine Reservierung von Bandbreite für eine logische Ende-zu-Ende-Verbindung ist bei Verwendung von SPANS nicht möglich. Dies wird erst von dem vom ATM-Forum standardisierten Protokoll UNI 3.0 [1] unterstützt. Bei Überlastung einzelner Verbindungen kommt es zu Paketverlusten auf Grund von überlaufenden Puffern bei den ATM-Gerätetreibern oder zu Zellverlusten im Switch.

Datentransport: Die Bibliotheksroutinen unterstützen die Datenübertragung nach AAL-Typ 3/4 und 5. Der Übertragungsmodus verbindungslos bzw. verbindungsorientiert wird bei AAL 3/4 nicht unterschieden. Die Daten werden über einen Puffer, dessen maximale Größe (*MTU – Maximum Transport Unit*) vom ATM-Gerätetreiber festgelegt wird, ausgetauscht. Die Implementierungen der Fore-Gerätetreiber basieren auf der STREAM-Schnittstelle des jeweiligen Betriebssystems (Kernel). Die Größe der *MTU* beträgt unter AIX 4092 Bytes, unter HP-UX 9188 Bytes. Zum Senden und Empfangen von Daten gibt es die Funktionen `atm_send()` und `atm_recv()`.

3.2 Beschreibung der API-Aufrufe

Das Schema in Abbildung 3.1 zeigt die API-Funktionsaufrufe bei einem verbindungsorientierten Protokoll zwischen Client und Server. Zur Vereinfachung wird von einer unidirektionalen Datenübertragung vom Client zum Server ausgegangen. Im weiteren wird kurz auf die einzelnen Funktionen des API eingegangen.

```
atm_open (const char* device, int flags, Atm_info* info);
```

Mittels `atm_open()` öffnet die Anwendung einen Filedeskriptor für das ATM-Device. In der Struktur `atm_info` werden Informationen über die ATM-Verbindung zurückgeliefert. Momentan ist in der Struktur aber nur die maximale Puffergröße *MTU* enthalten.

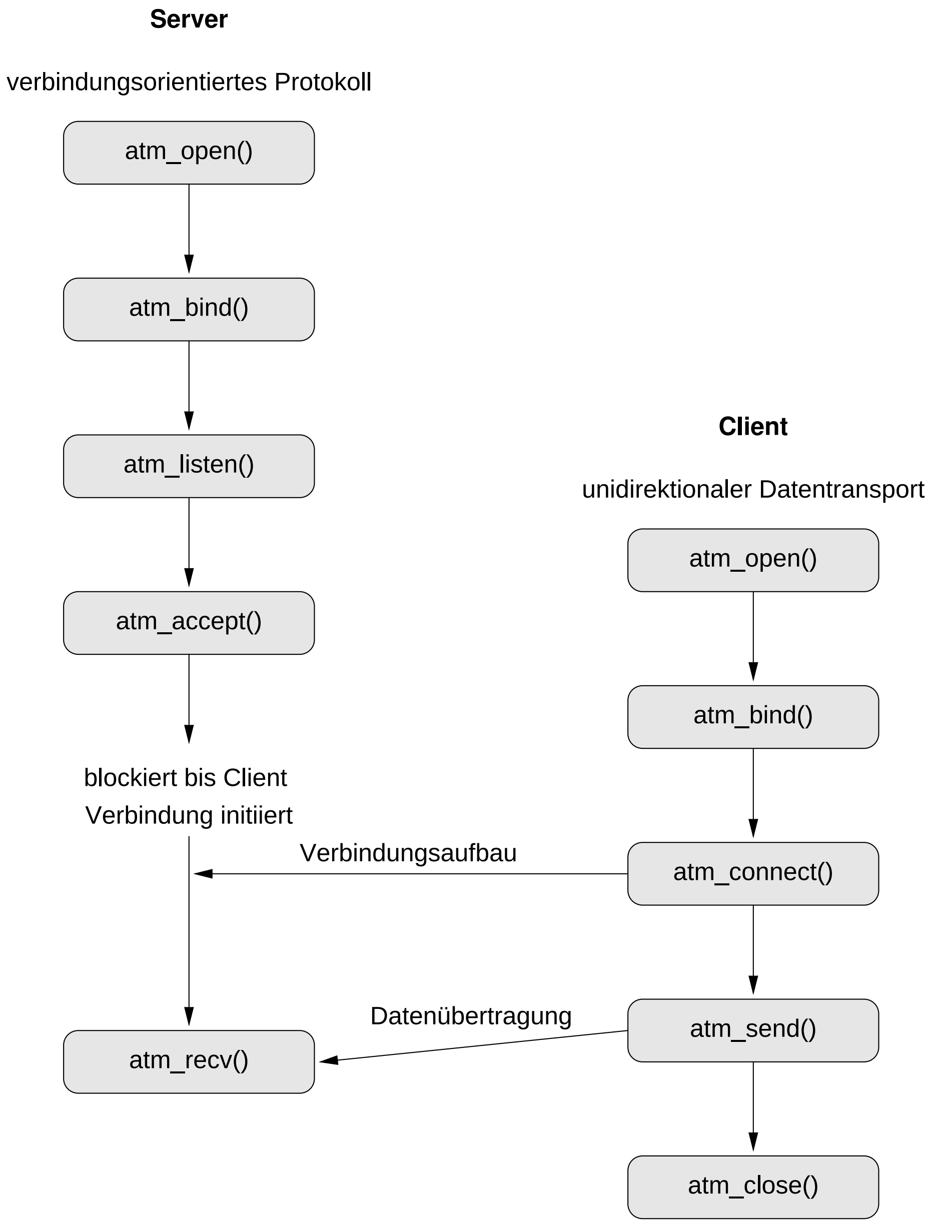

Abbildung 3.1: API-Funktionsaufrufe einer Client-Server-Anwendung

```
int atm_bind (int fd, Atm_sap asap, Atm_sap* asap_ret, int qlen);
```

`atm_bind()` bindet den Filedeskriptor `fd` an einen lokalen Application Service Access Point `asap`. Der Parameter `qlen` bestimmt bei einem Server die maximale Anzahl von Clients, die gleichzeitig auf einen Verbindungsaufbau zum Server warten dürfen.

```
int atm_listen (int fd, int* conn_id, Atm_endpoint* calling,
                Atm_qos* qos, Aal_type* aal);
```

Ein Server wartet beim Aufruf von `atm_listen()` auf einen Verbindungsaufbauwunsch durch einen Client. Wird dieser mittels `atm_connect()` gesendet, liefert die Funktion den Identifikator `conn_id`, der eindeutig dem Client zugeordnet wird, die ATM-Endsystemadresse `calling` des entfernten Rechners und die erwünschten Dienstgüteparameter `qos` und AAL-Typ zurück. Akzeptiert der Server die Verbindungsanforderung mittels `atm_accept()` wird der Identifikator `conn_id` an den Accept-Aufruf übergeben.

```
int atm_accept (int fd, int fdnew, int conn_id, Atm_qos* qos,
                Atm_dataflow dataflow);
```

Nimmt der Server eine Verbindungsanforderung an, kann für die aufgebaute Verbindung ein neuer Filedeskripter `fdnew` verwendet werden, damit der Server mit dem alten Filedeskriptor weiterhin Anforderungen durch `atm_listen()` behandeln kann. Der einen bestimmten Client bezeichnende Identifikator `conn_id` wird von `atm_listen()` geliefert. In `qos` sind wieder die geforderten Dienstgüteparameter gespeichert. Mit `dataflow` wird angeben, ob eine unidirektionale, bidirektionale oder Multicast-Verbindung aufgebaut werden soll. Im Falle einer bidirektionalen Verbindung wird von `atm_accept()` zusätzlich ein Rückkanal eingerichtet.

```
int atm_connect (int fd, Atm_endpoint* dest, Atm_qos* qos,
                 Atm_qos_sel* sel, Aal_type aal,
                 Atm_dataflow dataflow);
```

Die Funktion `atm_connect()` wird vom Client benutzt, um eine Verbindung zu initiieren. Die ATM-Endsystemadresse `dest` kann von `atm_gethostbyname()` ermittelt werden. Die Variable `qos` enthält die gewünschten Dienstgüteparameter, die vom ATM-Netz tatsächlich erbrachten *Quality of Services* für die Verbindung werden in `sel` zurückgeliefert. Die Parameter `aal` und `dataflow` wurden bereits erklärt.

```
int atm_gethostbyname (const char* hostname, Atm_address* dest);
```

Die Funktion gibt für einen gegebenen Hostnamen die ATM-Endsystemadresse zurück, indem sie zuerst `gethostbyname()` benutzt, um die IP-Adresse zu ermitteln, und anschließend diese mit der ARP-Tabelle (*Address Resolution Protocol Table*) des ATM-Gerätetreibers in die Endsystemadresse abbildet. Diese Abbildung funktioniert nur für Hosts, die *Classical IP and ARP over ATM* gemäß RFC 1577 [16] unterstützen.

```
int atm_send (int fd, const char* buf, int len);
```

Mit `atm_send()` wird versucht, die Anzahl `len` Bytes aus dem Puffer `buf` über die ATM-Verbindung, die mit dem Filedeskriptor `fd` gekoppelt ist, zu verschicken. Dabei darf `len` die

Puffergröße und die maximale Größe einer Transportdateneinheit *MTU* nicht überschreiten. Die tatsächliche Anzahl versendeter Bytes wird von der Funktion zurückgegeben.

```
int atm_recv (int fd, char* buf, int len);
```

Dies ist das Gegenstück zu `atm_send()`. Die über den Filedeskriptor `fd` gelesenen Daten werden im Puffer `buf` abgelegt. Die maximale Anzahl Bytes, die bei dem Funktionsaufruf gelesen wird, spezifiziert `len`. Da die tatsächliche Anzahl gelesener Bytes auch niedriger als `len` sein kann, muß die Funktion gegebenenfalls mehrfach aufgerufen werden, um alle Daten zu empfangen.

```
int atm_close (int fd);
```

Schließt die Verbindung, die mit dem Deskriptor `fd` assoziiert ist. Bei einer Duplex-Verbindung werden beide virtuellen Kanäle abgebaut.

4

PVM für ATM

PVM – *Parallel Virtual Machine* [11] ist eine Programmierumgebung zur Realisierung der Kommunikation in parallelen Anwendungen. Sie stellt auf einem vernetzten und möglicherweise heterogenen Workstation-Cluster eine virtuelle Maschine zur Ausführung von parallelen Programmen zur Verfügung. Die Kommunikation zwischen den Prozessen eines parallelen Programms erfordert den Austausch von Nachrichten (*message passing*). Die parallele Programmierumgebung PVM enthält Bibliotheksfunktionen zur Realisierung des Nachrichtenaustauschs. Die virtuelle Maschine einer parallelen Programmierumgebung kann sehr teure Hochleistungs- bzw. Parallelrechner ersetzen. Der „Flaschenhals" einer virtuellen Maschine war bisher die Kommunikation zwischen den Prozessen über das Netzwerk. Im Gegensatz zur Rechenleistung, die bei modernen Workstations in den vergangenen Jahren um Größenordnungen zugenommen hat, blieb die Leistung der traditionellen lokalen Netze wie z.B. Ethernet im wesentlichen über Jahre konstant. ATM-Hochleistungsnetze etablieren sich in jüngster Zeit stark im LAN-Bereich, obwohl sie eigentlich für den Weitverkehrsbereich entwickelt wurden. Sie können möglicherweise die hohen Anforderungen, die Systeme wie PVM an die Interprozeßkommunikation stellen, erfüllen.

4.1 Eignung von ATM für parallele Programmierumgebungen

Folgende Gründe sprechen für den Einsatz von ATM in parallelen Programmierumgebungen:

Hoher Durchsatz: Bei den herkömmlichen LANs wie Ethernet oder FDDI teilen sich architekturbedingt alle angeschlossenen Rechner die vom Medium angebotene Bandbreite. Diese bewegt sich im Bereich von 10 MBit/s (Ethernet) bis zu 100 MBit/s (FDDI). Der Gesamtdurchsatz eines ATM-Switches liegt dagegen im Gigabit-Bereich. Jeder Rechner ist darüberhinaus mit einer dedizierten Bandbreite von beispielsweise 155 MBit/s mit dem Switch verbunden.

Skalierbarkeit: In einem klassischen Datennetz steigt die Netzbelastung mit der Zahl der angeschlossenen Rechner. Ab einer gewissen Grenze tritt eine Sättigung des Netzes auf und die Anzahl der Kollisionen beim Zugriff auf das Medium steigt stark an. Im Fall von ATM skaliert die Zahl der angeschlossenen Rechner mit der Portanzahl des Switches.

Wird die Portanzahl vergrößert oder ein zusätzlicher Switch eingesetzt, kann das Netz vergrößert werden, ohne daß die Netzleistung pro angeschlossener Einheit sinkt.

Niedrige Latenzzeiten: Geringe Latenzzeiten sind besonders wichtig in parallelen Programmierumgebungen, da häufig kleine Nachrichten zwischen den Prozessen eines parallelen Programms ausgetauscht werden. Bei Parallelrechnern kann dies durch gemeinsamen Speicher (*shared memory*) oder durch spezielle Busarchitekturen zwischen den Prozessoren erreicht werden. Switchbasierte Hochleistungsnetze mit dedizierten Verbindungen wie ATM sollten auf Grund ihrer Architektur niedrige Latenzzeiten bieten, da im Gegensatz zum Ethernet keine Kollisionen auftreten und die Durchlaufzeiten im Switch im Mikrosekundenbereich (nach [7]) liegen. Dieser Vorteil wird aber derzeit noch durch die Tatsache relativiert, daß Hardware- und Softwarekomponenten für das Ethernet bereits für niedrige Latenzzeiten optimiert wurden, was bei der jungen ATM-Technologie noch nicht der Fall ist.

Multicasting: ATM unterstützt effizientes Multicasting. Auch dieser Aspekt ist für Systeme interessant, die öfter Operationen (z.B. Sperren, Aktualisierung) auf verteilten Datenkopien ausführen.

4.2 Kommunikationspfade innerhalb von PVM

PVM realisiert auf einem Workstationverbund eine virtuelle Maschine zur Ausführung von parallelen Anwendungen und stellt eine Programmierschnittstelle für den Nachrichtenaustausch (*message passing*) innerhalb der Anwendungen zur Verfügung. Parallele Programme können in C oder Fortran implementiert werden. Das Versenden einer Nachricht zwischen zwei Prozessen läuft unter PVM folgendermaßen ab:

1. Einrichten eines Sendepuffers mittels `pvm_initsend()` oder `pvm_mkbuf()`.
2. Einpacken der Nachricht (Feld eines bestimmten Datentyps) in den Puffer durch Aufruf der für den Datentyp geeigneten Packroutine `pvm_pk*()`.
3. Senden der Nachricht an einen (`pvm_send()`) oder mehrere (`pvm_mcast()`) Prozesse.
4. Auspacken der Nachricht beim Empfänger aus dem aktiven Empfangspuffer mittels `pvm_unpk*()` und Konvertierung in die ursprünglichen Datentypen.

Die Interprozeßkommunikation innerhalb von PVM basiert auf der BSD-Socket-Programmierschnittstelle [22]. Als Transportprotokolle werden TCP (*Transport Control Protocol*) und UDP (*User Datagram Protocol* benützt, die auf IP (*Internet Protocol*) aufsetzen. TCP bietet zuverlässige verbindungsorientierte Kommunikation mit Flußkontrolle und Reihenfolgesicherung. UDP ist ein paketorientiertes Protokoll. Für die Pakete wird nicht garantiert, daß sie den Empfänger überhaupt, in der richtigen Reihenfolge oder dupliziert erreichen. Durch den Einsatz weitverbreiteter Standardprotokolle konnte PVM auf nahezu jede Hardwareplattform portiert werden.

Auf jedem Rechner, der Teil der virtuellen Maschine ist, läuft der Dämonprozeß **pvmd**. Über die Dämonen pvmd wird die gesamte Kommunikation innerhalb von PVM abgewickelt. Sie tauschen untereinander Informationen über die Netzkonfiguration aus und organisieren das Hinzufügen und Herausnehmen von Rechnern aus dem Verbund. Außerdem sind sie für die PVM-Benutzerprozesse auf dem betreffenden Host verantwortlich und übernehmen das Verteilen und Routing der Nachrichten. Abbildung 4.1 zeigt eine virtuelle Maschine bestehend aus drei Hosts.

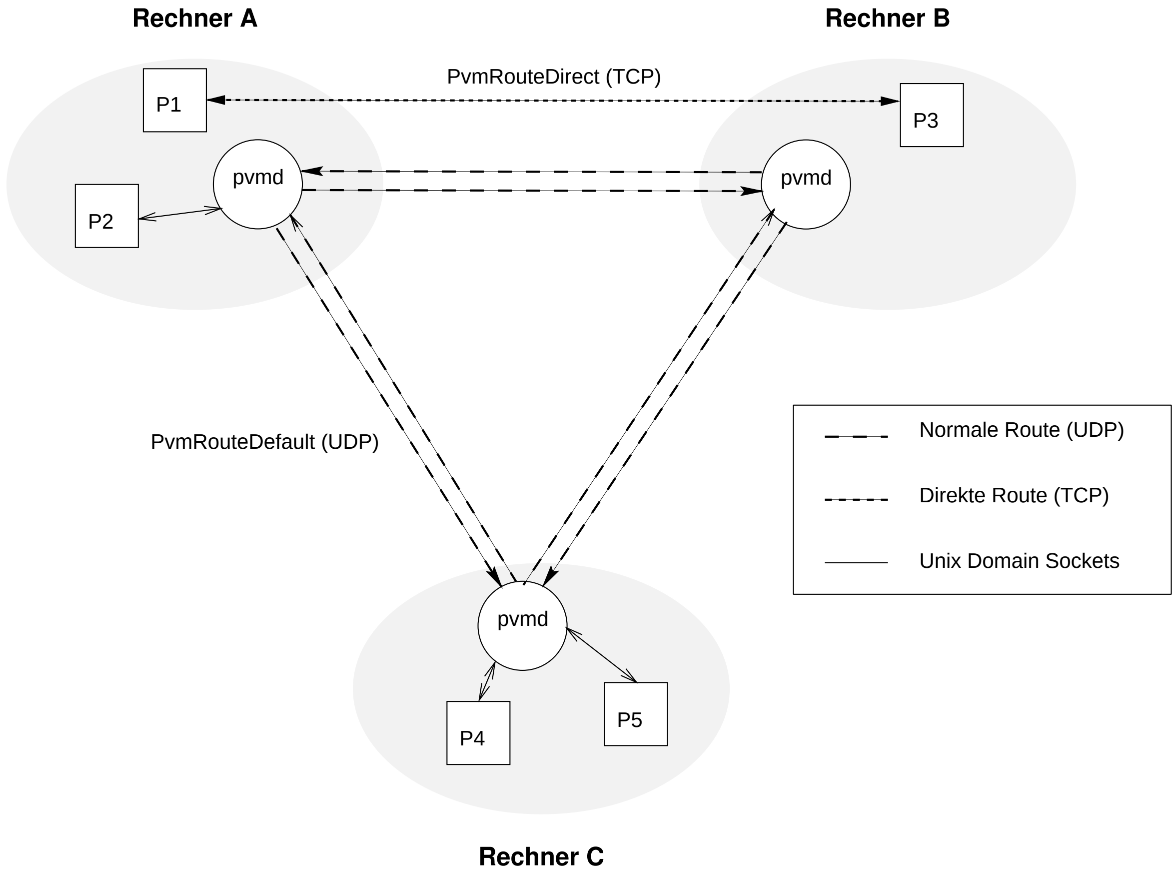

Abbildung 4.1: Kommunikationswege innerhalb einer virtuellen Maschine

Auf jedem Host läuft ein lokaler PVM-Dämon pvmd und eine Reihe von Benutzerprozessen (*tasks*). Der erste gestartete PVM-Dämon der virtuellen Maschine ist der *master pvmd*. Zwischen ihm und den übrigen pvmd besteht eine *Master–Slave*-Beziehung. Operationen wie das Starten und Beenden der virtuellen Maschine oder auch das Multicasting von Nachrichten wird vom *master pvmd* gesteuert. Im Normalbetrieb agieren die Dämonen aber als gleichberechtigte Prozesse, die sich um das Routing und um die Steuerung der Nachrichten kümmern. Die Kommunikation zwischen einem Benutzerprozeß und einem lokalen pvmd innerhalb eines Hosts wird über UNIX Domain Sockets abgewickelt. Untereinander kommunizieren die PVM-Dämonen über UDP Sockets. Für das Routing von Nachrichten zwischen zwei Benutzerprozessen auf verschiedenen Hosts, sind in PVM zwei Modi definiert. Im Normalfall (*PvmRouteDefault*) übergibt der Prozeß die Nachricht an den lokalen Dämon pvmd, dieser schickt sie an den entfernten pvmd, der sie wiederum an den Empfängerprozeß weiterreicht. Also werden zwei TCP- und zwei UDP-Verbindungen für die bidirektionale Kommunikation von je zwei Benutzerprozessen benötigt. Im Direktmodus (*PvmRouteDirect*) wird nur eine

TCP-Verbindung zwischen den kommunizierenden Benutzerprozessen errichtet. In der Abbildung besteht eine normale Verbindung zwischen den Prozessen P2 und P4 wohingegen die Prozesse P1 und P3 im Direktmodus Nachrichten austauschen.

Der Vorteil des Direktmodus liegt in der größeren Effizienz des Kommunikationspfads. In [24] wird im Vergleich zum normalen Routing von einer mehr als doppelt so großen Leistung berichtet. Der Nachteil des Direktmodus ist die begrenzte Skalierbarkeit. Jede TCP-Verbindung benötigt einen Filedeskriptor. Bei einigen Betriebssystemen ist die Anzahl der offenen Filedeskriptoren aber auf 32 beschränkt.

4.3 Nutzung des Fore-API durch PVM

Transportprotokolle wie TCP und UDP, die auf IP basieren, wurden für Weitverkehrsnetze konzipiert, die Datenpakete vom Sender über eine Anzahl von Router zum Empfänger transportieren. TCP ist so robust, daß es Daten zuverlässig über Netze mit hohen Fehlerraten schicken kann. Die Robustheit, die durch aufwendige Flußkontrolle und durch das wiederholte Versenden von Segmenten im Fehlerfall erreicht wird, bringt einen großen Overhead für das Transportprotokoll mit. Bei einem ATM-Netz ist diese Robustheit nicht nötig. Die Fehlerrate in einem ATM-Netz ist viel geringer als in traditionellen Netzen. Bei ATM werden Daten im Gegensatz zur Store-and-Forward-Architektur herkömmlicher Netze auf logischen Verbindungen zwischen Sender und Empfänger transportiert. Das „Routing“ erfolgt durch Hardwarekomponenten (Switches) auf der Ebene der Zellschicht. ATM-Zellen einer Verbindung werden alle über den gleichen Pfad durch das Netz befördert. Daher erreichen die Zellen den Empfänger immer in der richtigen Reihenfolge. Zellen werden nicht verdoppelt und Zellverluste treten nur bei Überlastung eines Switches oder bei einem Pufferüberlauf an der ATM-Adapterkarte auf. Die Reaktion auf verlorene Zellen kann viel früher auf einer unteren Ebene geschehen als auf der Segmentschicht von TCP. Außerdem berücksichtigt TCP die Dienstgüteparameter von ATM nicht.

Aus den genannten Gründen wurde an der *Universität von Minnesota* eine auf PVM 3.3.2 basierende PVM-Version entwickelt, die die Kommunikation zwischen den PVM-Prozessen über das Fore-API betreibt. Für die Tests in dieser Arbeit wurde die Version PVM-ATM 3.3.2.0 vom 21.07.94 eingesetzt. Es werden die AAL-Typen 3/4 und 5 unterstützt. Für die PVM-Benutzerprogramme ist die Kommunikation über das Fore-API transparent. Lediglich eine erneute Übersetzung mit der PVM-ATM-Bibliothek ist notwendig. Probleme bei der Installation von PVM-ATM konnten durch Einsatz von PVM 3.3.0 als Basis und nach Korrektur einiger Makefiles umgangen werden. Darüberhinaus traten aber bei der Benutzung von PVM-ATM folgende Fehler auf, die die Tests erheblich einschränkten. Dies ist möglicherweise darauf zurückzuführen, daß PVM-ATM für Fore-Karten für SUN-Workstations unter Berücksichtigung spezieller Hardwareparameter entwickelt wurde.

- Der Master-PVM-Dämon terminiert beim Versuch ATM-Hosts in den Verbund herein- oder herauszunehmen (*add host* bzw. *delete host*). Nicht reproduzierbar.
  Meldung: `libpvm [t40001]: pvm_addhosts(): Can't contact local daemon`
- Der Direktmodus für das Routing von PVM-Nachrichten (*PvmRouteDirect*) hat im Test trotz Unterstützung im Quellcode von PVM-ATM nicht funktioniert. Damit ist

die Kommunikation über das Fore-API auf das Austauschen von Nachrichten zwischen den PVM-Dämonen beschränkt. Die Benutzerprozesse müssen die Nachrichten vorher über UNIX Domain Sockets an den lokalen pvmd geben.
Meldung: `libpvm [t80001]: atminput() frag with no message`

- Beim Auspacken von Nachrichten, die gemischte Datentypen enthalten, traten Fehler durch unerwartetes Pufferende auf. Gepackte Nachrichten, die nur Daten von einem Typ enthalten (z.B. Byte), werden fehlerfrei übertragen.
  Meldung: `libpvm [t80002]: pvm_upkdouble(): End of buffer`

Das folgende Kapitel beschäftigt sich mit der Messung zweier Leistungswerte, die beim Verschicken von Nachrichten über ein Datennetz eine große Bedeutung haben. Zum einen handelt es sich um den Durchsatz, der beim Transport einer großen Datenmenge (*bulk transfer*) über das Netz erreicht wird. Der zweite wichtige Parameter ist die Latenzzeit beim Übertragen einer kurzen Nachricht.

5

Leistungsmessungen im ATM-Netz

Im praktischen Teil dieser Arbeit wurden die Leistungswerte Durchsatz und Kommunikationslatenzzeit mit verschiedenen Benchmarks für das ATM-Netz am LRR-TUM gemessen. An erster Stelle steht die Frage nach dem erreichbaren Datendurchsatz für eine parallele Anwendung. Wie in Kapitel 2.4 beschrieben, ist oberhalb der physikalischen Schicht von ATM ein theoretischer Durchsatz von 149.76 MBit/s möglich. Interessant ist allerdings, wie hoch die resultierende Bandbreite für eine Anwendung auf einer höheren Ebene ist wie z.B. der Socket-Schnittstelle (TCP / UDP) oder dem Fore-API. Test1 in [2] hat gezeigt, daß die verwendeten Workstations von HP und IBM in der Lage sind, Daten mit einem Durchsatz von über 380 MBit/s aus dem Benutzerbereich über den Betriebssystemkernel an den ATM-Gerätetreiber zu transferieren. Dieser unter dem einfachen Protokoll UDP gemessene Wert bedeutet, daß die Rechner den oben genannten theoretischen Durchsatz von ATM bedienen können sollten. In den nächsten Abschnitten werden die gemessenen Leistungswerte folgender Protokollkombinationen vorgestellt.

- IP over ATM (AAL 5): Stream Sockets (TCP) (nttcp, netperf)
- IP over ATM (AAL 5): Datagramme (UDP) (nttcp)
- Fore-API: ATM AAL 3/4 und AAL 5 (nttcp, netperf)
- PVM (IP over ATM, PvmRouteDefault, UDP) PVM-Benchmark
- PVM (IP over ATM, PvmRouteDirect, TCP) PVM-Benchmark
- PVM (Fore-API, AAL 5, PvmRouteDefault) PVM-Benchmark

Vorher wird kurz auf die Technik der Meßprogramme eingegangen.

5.1 Technik der Meßprogramme

5.1.1 Durchsatzmessungen

Der Durchsatz einer Kommunikationsverbindung ist folgendermaßen definiert:

$$Durchsatz = \frac{\ddot{U}bertragene\ Datenmenge}{Ben\ddot{o}tigte\ \ddot{U}bertragungszeit}$$

Interessant ist vor allem der Durchsatz der bei der Übertragung einer großen Datenmenge (*bulk transfer*) kontinuierlich vom Netz erreicht wird. Unterschieden wird der Sender- und der Empfängerdurchsatz. Abhängig vom eingesetzten Transportprotokoll können die zwei Werte voneinander abweichen. Im Fall von TCP, das verbindungsorientiert mit Quittierung der Segmente arbeitet, sind die beiden Durchsatzwerte gleich. Anders verhält es sich bei UDP und bei dem Fore-API. Bei einer Socket-Verbindung unter UDP ist i. a. der Senderdurchsatz größer als der beim Empfänger. Beim Sender stoppt die Zeitmessung nach dem Verschicken des letzten Pakets. Gehen auf dem Weg durch das Netz einzelne Pakete verloren, sinkt die empfangene Datenmenge und damit der Empfängerdurchsatz. Genauso verhält es sich, wenn UDP-Datagramme in Puffern von Routern bei hoher Netzlast verzögert werden. Da das Fore-API keine Funktionen wie Flußkontrolle und Quittieren von Paketen beinhaltet, ist auch hier die Empfängerdatenrate im Fall von Zellverlusten oder bei Pufferüberlauf bei den ATM-Gerätetreibern geringer. Auf Anwendungsebene hat der Wert für die Empfängerdatenrate größere Bedeutung. Die für diese Arbeit eingesetzten Benchmarks messen den Durchsatz mit drei unterschiedlichen Methoden:

Getrennte Zeitmessung beim Sender und Empfänger: Dies ist die einfachste Methode. Die Zeitmessung beginnt beim Sender und Empfänger nach dem Aufbau der Verbindung vor dem Verschicken des ersten Pakets. Sie endet bei verbindungsorientierter Übertragung (TCP) nachdem der Sender alle Daten übertragen hat und die Verbindung schließt. Da UDP verbindungslos arbeitet, muß dem Empfänger Beginn und Ende der Übertragung durch ein zusätzliches wenige Byte langes Paket angezeigt werden. Diese Methode wird bei dem Benchmark `nttcp` (Kapitel 5.2) angewendet. Da die Messung der beiden Durchsätze unkorreliert ist, muß die tatsächlich den Empfänger erreichte Datenmenge geprüft werden, um aussagekräftige Werte zu erhalten.

Korrelierte Zeitmessung: Eine zusätzliche Kontrollverbindung zwischen Sender und Empfänger steuert die Zeitmessung. Hierbei muß auch eine eventuelle Zeitabweichung der Uhren (*clock drift*) von Sender und Empfänger berücksichtigt werden. Der Benchmark `netperf` (Kapitel 5.3) nutzt diese Methode.

Messung mit Echoprogramm: Abbildung 5.1 zeigt den Pseudocode von Client und Server eines Echoprogramms. Der Client baut eine Verbindung zum Server auf und sendet ein M Bytes großes Paket. Empfängt der Server ein Paket, schickt er es unverzüglich an den Client zurück. Der Client mißt die Verzögerung zwischen Senden und Empfangen eines Pakets (*round trip delay*). Die Latenzzeit für die Übertragung eines M Bytes großen Pakets ist ungefähr die Hälfte des *round trip delay*. Der Durchsatz d für ein Paket errechnet sich aus der folgenden Formel:

$$d = \frac{2 * M}{round\ trip\ delay}$$

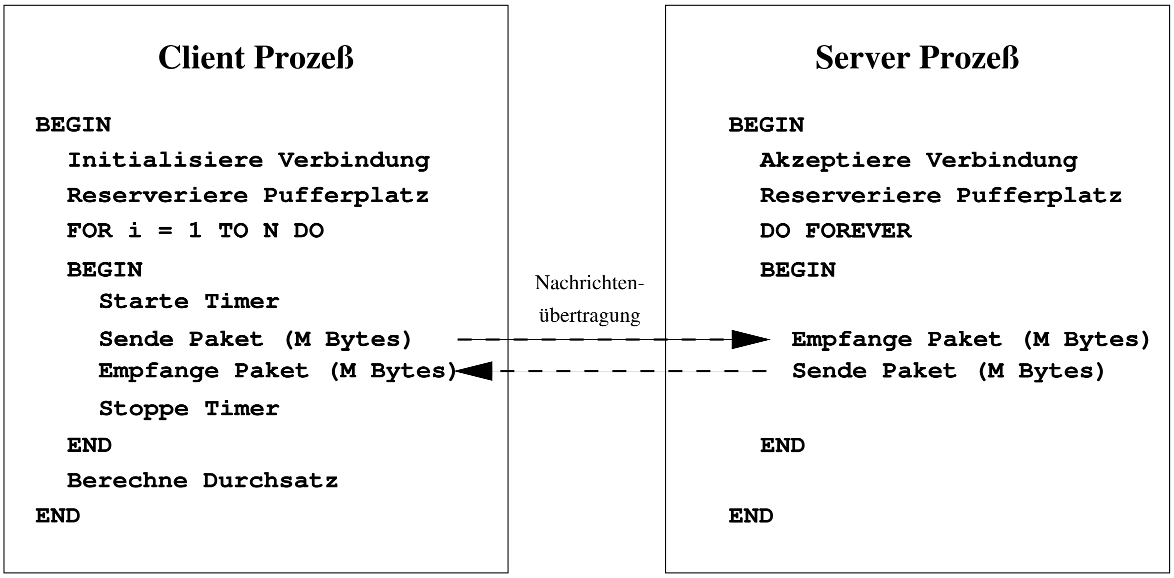

Abbildung 5.1: Pseudocode eines Echoprogramms

Dieser Vorgang wird N-mal wiederholt, um Durchschnittswerte für den Durchsatz und die Latenzzeit berechnen zu können. Mit einem Echoprogramm kann das Problem der Uhrensynchronisation von Sender und Empfänger bei der Messung von Latenzzeiten umgangen werden. Der PVM-Benchmark (Kapitel 5.4.1) ist ein einfaches Echoprogramm.

5.1.2 Messung von Latenzzeiten

Für parallele Anwendungen innerhalb PVM, die untereinander kurze Nachrichten austauschen, ist die Latenzzeit ein wichtiges Maß. Die Latenzzeit eines Kommunikationsnetzes ist definitionsgemäß die Zeitspanne zwischen dem Senden des ersten Bits eines Datenblocks beim Sender und dem Empfangen des letzten Bits des Blocks beim Empfänger. Aus Anwendungssicht wird die Latenzzeit durch viele Faktoren beeinflußt. Hierzu zählen die Implementierung des Protokollstapels (z.B. TCP/IP), der Gerätetreiber, Zugriffszeit auf das Netzmedium (z.B. Ethernet), Übertragungsraten, Signallaufzeiten und Verarbeitungszeiten in Switches und Routern. Beim Austausch von kurzen Nachrichten spielt die Übertragungsrate des Netzes nur eine kleine Rolle. Hier ist die sog. *startup latency* von Bedeutung, die in [17] als die Übertragungszeit einer Nachricht mit minimaler Länge definiert ist. In der Literatur finden sich für die minimale Länge Werte von 0, 1, 4 oder 16 Bytes. Bei der Messung ist zu berücksichtigen, daß abhängig vom Verbindungsmodus die Latenzzeit des ersten Pakets einer Serie erheblich größer sein kann. Bei verbindungsorientierter Kommunikation (TCP, Fore-API) kommt zur reinen Übertragungszeit noch die Verbindungsaufbauzeit dazu.

Latenzzeiten werden i.a. mit Echoprogrammen gemessen. Die Latenzzeit für ein Paket beträgt ungefähr die Hälfte der Antwortzeit (*round trip delay*). Gewöhnlich wird ein Durchschnittswert für die Latenzzeit angegeben, der aus den Meßdaten für die Antwortzeiten von N Paketen berechnet wird.

5.2 Durchsatzmessungen mit `nttcp`

Im ersten Teil der Messungen wurden Werte für den Durchsatz bei Übertragung einer großen Datenmenge ermittelt. In den folgenden Diagrammen werden die erzielten Leistungswerte von ATM für die Protokolle TCP und UDP und das Fore-API mit denen von herkömmlichen Ethernet (10 MBit/s) verglichen. Für die Messungen wurde das weit verbreitete C-Programm `ttcp` in einer modifizierten Version eingesetzt. Das Programm ist public domain, Grundlage für diese Arbeit war die Version von `ftp://ftp.fore.com/pub/utils/nttcp.c` vom 24.05.1995. Hiermit können Durchsatzwerte von TCP- und UDP-Strömen zwischen zwei Hosts A und B über die BSD-Socket-Programmierschnittstelle gemessen werden. Teil dieser Arbeit war es, `nttcp` für die Unterstützung des Fore-API zu erweitern. Die Funktionsweise von `nttcp` zeigt Abbildung 5.2.

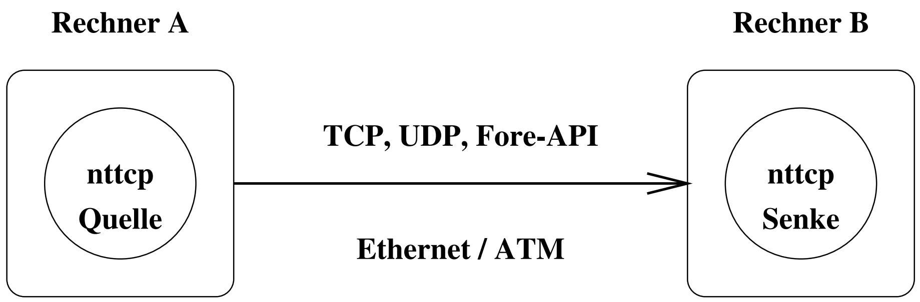

Abbildung 5.2: `nttcp` Benchmark

Auf Rechner A wird `nttcp` als Datenquelle, auf Rechner B als Datensenke konfiguriert. Die insgesamt zu übertragene Datenmenge beträgt n Blöcke (*send buffer*) mit Nachrichtenlänge l (*buffer size*). Der Sender-`nttcp` sendet kontinuierlich Datenblöcke zum Empfänger-`nttcp`, bis alle n Blöcke übertragen sind. Der Empfänger verwirft die Blöcke nach Erhalt. Über Aufrufparameter konfiguriert der Benutzer `nttcp` als Sender bzw. Empfänger, legt das Protokoll fest (TCP, UDP, Fore-API), spezifiziert protokollspezifische Einstellungen (z.B. TCP-Fenstergrößen) und gibt die Datenmenge an. Die Aufrufparameter von `nttcp` sind im Anhang A erklärt.

Zur Berechnung des Durchsatzes benutzt `nttcp` zwei Zeitstempel. Der erste Zeitstempel wird beim Sender unmittelbar vor dem ersten *write*-Systemaufruf genommen, der zweite nach Übergabe des letzten Blocks. Beim verbindungsorientierten TCP und auch beim Fore-API nimmt der Empfänger den ersten Zeitstempel nach Aufbau der Verbindung, den zweiten nach Abbruch der Verbindung durch den Sender. Wird das verbindungslose UDP eingesetzt, startet bzw. beendet ein vier Byte langes Datagramm, welches der Quell-`nttcp` vor und nach der eigentlichen Übertragung sendet, die Zeitmessung beim Empfänger. Die Zeitstempel werden über den Systemaufruf *gettimeofday* ermittelt und haben unter AIX und HP-UX eine Genauigkeit von einer Mikrosekunde. Sender- und Empfängerdurchsatz werden also unabhängig voneinander berechnet.

Bei allen Experimenten wurden jeweils 16 MByte zwischen Sender und Empfänger übertragen. Die ermittelten Durchsatzwerte sind Durchschnittswerte aus mehreren gleichen Messungen. Einzelne Ausreißer wurden eliminiert. Während der Messungen war das ATM-Netz frei von anderem Datenverkehr. Da der Test-Cluster kein eigenes Ethernet-Segment bildet, wurden die Messungen über Ethernet am späten Abend durchgeführt, wo das Netz annähernd unbelastet war. Es folgen die Ergebnisse der Messungen. Wenn nicht anders angegeben, handelt es sich bei den abgebildeten Werten um den Empfängerdurchsatz. Der Durchsatz ist in der Einheit MBit/s[1] angegeben. Die Konfiguration des Sender- und Empfänger-`nttcp` für die einzelnen Tests ist jeweils im Anhang B angegeben.

5.2.1 Test 1: TCP-Stream (ATM versus Ethernet)

In Test 1 wurde der Durchsatz von TCP-Stream-Verbindungen über Ethernet und ATM bei variabler Nachrichtenlänge gemessen. Testpartner waren jeweils untereinander die Maschinen hpbode3/4 und ibode2/4.

Die Nachrichtenlänge l wurde vom Startwert 512 Bytes bis zum Endwert 32 KByte jeweils verdoppelt. Die Nachrichtenlänge bezeichnet bei `nttcp` die Länge des Puffers für die Socket-Funktionen *read* und *write*. Sie ist nicht mit der Länge des IP-Pakets zu verwechseln, welches die Nachricht oder einen Teil davon transportiert. Die maximale Größe der Transporteinheit (*maximum transport unit – MTU*) am Netzinterface bestimmt die Maximalgröße eines IP-Pakets und beträgt für Ethernet 1500 Bytes und für ATM 9188 Bytes. Pro Meßwert wurden jeweils 16 MByte in n Nachrichten der Nachrichtenlänge l übertragen. Die abgebildeten Durchsatzwerte sind Durchschnittswerte aus drei gleichartigen Messungen.

Für den Test wurden folgende Socket-Parameter angegeben:

- `TCP_NODELAY`: Verhindert, daß kleine TCP-Pakete, die aufgrund von Fragmentierung entstehen können, vor dem Senden verzögert werden.
- Die Socket-Puffergröße (`SO_SNDBUF` / `SO_RCVBUF`) wurde beim Sender-`nttcp` und beim Empfänger-`nttcp` auf 32 KByte eingestellt. Vergleiche hierzu Test 2.

Die genauen Aufrufparameter von `nttcp` für Test 1 finden sich im Anhang B. In Abbildung 5.3 sind die gemessenen Durchsatzwerte abgebildet. Erwartungsgemäß beträgt der Durchsatz der Ethernet-TCP-Verbindungen gleichbleibend 7 - 8 Mbit/s. Der Durchsatz der TCP-ATM-Verbindungen steigt jeweils mit wachsender Nachrichtenlänge und geht erst bei Nachrichtenlängen über 16 KByte leicht zurück. Den Spitzendurchsatz von 95.2 MBit/s (11.9 MByte/s) erreichen die IBM-Maschinen bei 8 KByte-Nachrichtenlänge, die HP-Maschinen erreichen nur maximal 67.9 MBit/s (8.5 MByte/s) bei 16 KByte-Nachrichtenlänge.

5.2.2 Test 2: TCP-Stream (ATM: Variable Socket-Puffergrößen)

In Test 2 wurde ebenfalls der Durchsatz von TCP-Stream-Verbindungen über ATM gemessen. Diesmal wurde die Größe der Socket-Sende- und Empfangspuffer variiert. Die Nachrichtenlänge betrug 8 KByte. Pro Meßwert wurden 2048 Nachrichten verschickt, was wiederum einem Gesamtübertragungsvolumen von 16 MByte entspricht. Ebenfalls war die Socket-Option `TCP_NODELAY` gesetzt.

Die Socket-Puffergröße beim Sender (`SO_SNDBUF`) und beim Empfänger (`SO_RCVBUF`) wurde jeweils auf denselben Wert gesetzt. Die Puffergröße wurde ausgehend von 4 KByte bis zum

[1] Hier: 1 $MBit = 8 * 1024^2\ Bit$ und nicht 1 $MBit = 8 * 10^3 * 1024\ Bit$!!

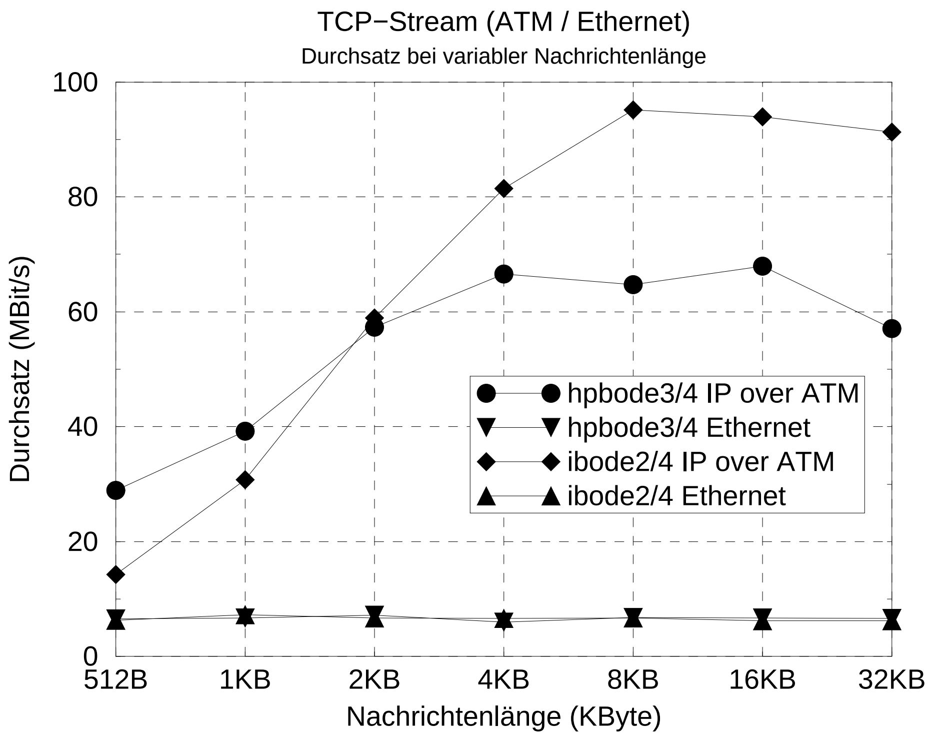

Abbildung 5.3: TCP: Durchsatz bei variabler Nachrichtenlänge

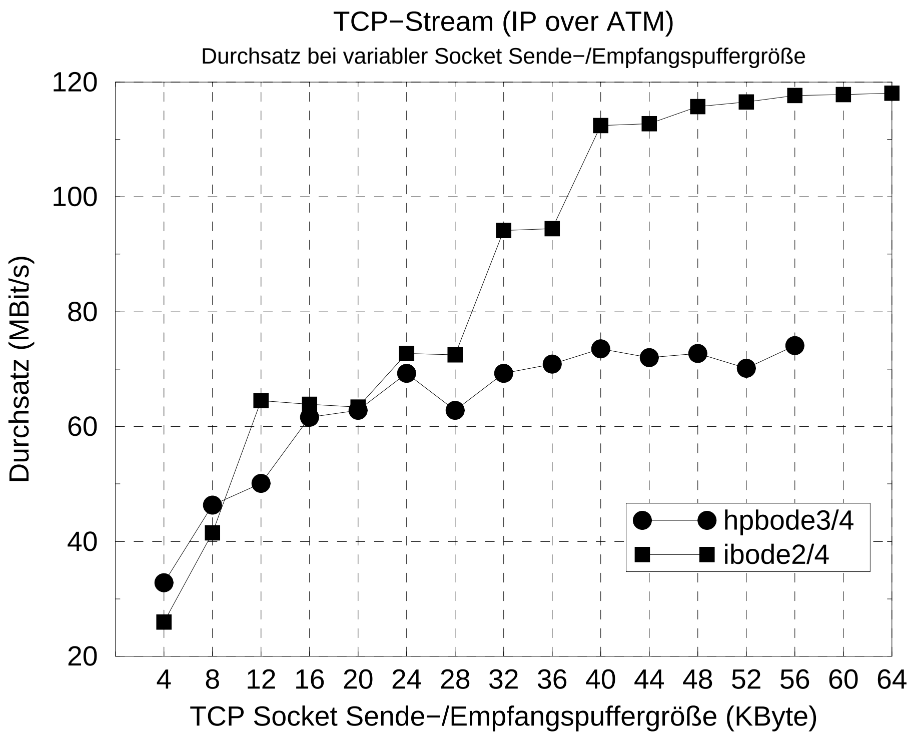

Abbildung 5.4: TCP: Durchsatz bei variabler Socket Sende-/Empfangspuffergröße

jeweiligen Maximalwert, der von der TCP/IP-Implementierung des Betriebssystems festgelegt ist, in 4KB-Schritten erhöht. Unter HP-UX beträgt der Maximalwert 56 KByte[2], per default hat der Puffer eine Größe von 8 KByte. Bei AIX ist der Standardwert gleichzeitig der Maximalwert nämlich 64 KByte[3].

In Abbildung 5.4 sind die Testergebnisse dargestellt. Die Socket-Puffergrößen haben einen großen Einfluß auf den erreichbaren Durchsatz. Die besten Durchsatzwerte werden jeweils bei der maximalen Socket-Puffergröße erreicht. Diese sind 74.1 MBit/s (9.3 MByte/s) bei den HP-Maschinen und 118 MBit/s (14.8 MByte/s) bei den IBM-Maschinen. Letzterer Wert kann als sehr gut angesehen werden, wenn man den Protokoll-Overhead von TCP bedenkt. Der erreichte Durchsatz bei HP ist eher mäßig. Die Treppenstufen in der Kurve von ibode2/4 entstehen dadurch, daß unter AIX ab einer Puffergröße von 8 KByte diese nur in 8-KByte-Schritten erhöht werden kann.

Der Standardwert für die Socket-Puffergröße von 8 KByte bei HP-UX erweist sich als viel zu gering für TCP-Datenübertragung über ATM. Weitere Informationen zu der TCP-Puffer-Problematik bei ATM-Netzen finden sich in [4].

5.2.3 Test 3: UDP-Datagramme (ATM versus Ethernet)

Im dritten Test wurde die Durchsatzleistung des einfachen Kommunikationsprotokolls UDP für Ethernet und ATM gemessen. Aufgrund des geringeren Protokoll-Overhead bei UDP im Vergleich zu TCP sind höhere Werte für den Durchsatz zu erwarten. Bei den Messungen über ATM trat aber folgendes Problem auf. Zum Teil war der Durchsatz beim Empfänger wesentlich kleiner als beim Sender, d.h. der Server-nttcp hat nur einen Bruchteil der insgesamt übertragenen Pakete empfangen. Da UDP verbindungslos ohne Flußkontrolle und Fehlerkorrektur arbeitet, ist nicht sichergestellt, daß der Empfänger-nttcp alle übertragenen Pakete korrekt erhält. Mit Hilfe der Werkzeuge `netstat`[4] und `atmstat`[5] wurde festgestellt, daß die Pakete vom Sender korrekt verschickt wurden und erst beim Empfänger verloren gehen. Ursache war ein Überlaufen des Socket-Puffer beim Empfänger, d.h. der Empfänger-nttcp hat die Pakete nicht schnell genug abgeholt. Durch Drosseln der Senderate mittels Vergrößern der Zeitspanne zwischen dem Abschicken zweier UDP-Pakete, wurde sichergestellt, daß ein Großteil der Pakete den Empfänger erreicht. Das Programm nttcp läßt eine Veränderung der Socket-Puffergröße bei UDP nicht zu. Diese Problematik wird ausführlich in [5] Kapitel 4 und 5 beschrieben.

Für den Test wurden Nachrichtenlängen von 512 Bytes, 1 KByte, 2 Kbyte, usw. bis zum Maximalwert von 8 KByte betrachtet. Der Maximalwert wurde nicht größer als die *MTU* von *IP over ATM* gewählt, um eine Fragmentierung der Nachrichten zu verhindern. Der Verzögerungswert für die UDP-Senderate (*inter packet delay*) wurde abhängig von der Nachrichtenlänge gerade so hoch eingestellt, daß mindestens 95% der Pakete den Empfänger-nttcp erreichten. Es wurden wieder jeweils 16 MByte pro Meßwert übertragen.

[2]Der Maximalwert ist eigentlich 58254 $Bytes = 56.89\ KByte$!

[3]Wurde mit dem Kommando `no -o tcp_sendspace = 65535` in `/etc/rc.net` als default gesetzt. Normalerweise beträgt der Default-Wert unter AIX 16 KByte.

[4]`netstat -i` gibt Statistik der gesendeten und empfangenen IP-Pakete eines Netzinterface aus.

[5]Tool von Fore zur Abfrage von Zählern auf AAL- und ATM-Schicht eines ATM-Interface (*Input/Output cells, CS-PDUs, Errors*)

Abbildung 5.5 enthält das Diagramm für diesen Test. Der Durchsatz bei der Übertragung über Ethernet liegt konstant bei 9 MBit/s also nahe am theoretischen Maximum von 10 MBit/s des traditionellen Ethernet. Da sich die Ethernet-Kurven bei HP und IBM nahezu überdecken, ist nur eine abgebildet. Der Durchsatz bei ATM steigt mit wachsender Nachrichtenlänge stark an. Bei kleinen Blockgrößen (512 Bytes) ist er aber nur etwa doppelt so groß wie beim Ethernet. Erwartungsgemäß wird der höchste Durchsatz bei der größten Nachrichtenlänge erreicht. Dieser beträgt für hpbode3/4 75.8 MBit/s (9.5 MByte/s) und für ibode2/4 128.4 MBit/s (16.1 MByte/s). Letzterer Wert kommt nahe an das mögliche Maximum in dieser Umgebung heran, vergleiche hierzu Test2 in [2]. Wiederum enttäuscht der Wert für HP. Dies läßt den Schluß zu, daß die Treiber für *IP over ATM* von Fore, die in den Betriebssystemkern von HP-UX eingebunden werden, noch zu optimieren sind.

5.2.4 Test 4: Fore-API (AAL 3/4, AAL 5)

Im letzten Test mit `nttcp` wurden Durchsatzwerte für die Kommunikationsschnittstelle Fore-API gemessen. Der Protokoll-Overhead ist im Vergleich zu TCP/IP geringer, da das API direkt auf der AAL-Schicht aufsetzt (vgl. Abbildung 2.7). Kommunikation über das Fore-API ist zwar verbindungsorientiert, aber nicht gesichert. Alle Funktionen der Transportschicht wie Flußkontrolle, wiederholtes Senden eines fehlerhaften oder verlorenen Pakets, etc. müssen von der Kommunikationsanwendung erfüllt werden. Bei den Messungen mit `nttcp` ergaben sich daher ähnliche Probleme wie bei der Verwendung von UDP.

Der Durchsatz wurde wiederum für wachsende Nachrichtenlängen gemessen. Die maximale Nachrichtenlänge wird durch die *MTU* des Fore-API bestimmt. Die *MTU* ist die maximale Größe des Sendepuffers, der dem API-Aufruf `atm_send()` übergeben wird. Unter HP-UX beträgt die *MTU* 9188 Bytes, unter AIX nur 4092 Bytes, obwohl die Implementierung von *IP over ATM* unter AIX ebenfalls eine *MTU* von 9188 Bytes vorsieht. Im Test wurden für IBM Nachrichtenlängen bis 4092 Bytes betrachtet (512 Bytes-Schritte), für HP Nachrichtenlängen bis 9188 Bytes (1 KByte-Schritte).

Die Messungen wurden sowohl für AAL 3/4 als auch für AAL 5 durchgeführt. Der Zell-Overhead von AAL 3/4 ist wesentlich größer als der von AAL 5. Beispielsweise beträgt er bei einer Nachrichtenlänge von 8704 Bytes 800 Bytes respektive 32 Bytes für AAL 5 ([18], Tabelle 1). Während der Tests hat sich herausgestellt, daß bei Verwendung des Fore-API die Senderate der Anwendung (`nttcp`) wie bei UDP gedrosselt werden muß. Ungebremst traten erhebliche Datenverluste sowohl beim Sender als auch beim Empfänger auf. Die erforderlichen Verzögerungswerte wuchsen mit der Nachrichtenlänge und waren bei AAL 3/4 größer als bei AAL 5. Sie wurden für die Messungen experimentell ermittelt. Auf Senderseite durften keine Daten verloren gehen, die Verlustrate beim Empfänger durfte 5% nicht überschreiten.

Die Ergebnisse sind in Abbildung 5.6 zusammengefaßt. Die Durchsatzwerte von AAL 5 sind gegenüber AAL 3/4 in beiden Fällen wesentlich besser. Die Maximalwerte werden bei AAL 5 jeweils knapp unterhalb der *MTU* erreicht und betragen für HP 87.0 MBit/s (10.9 MByte/s) und für IBM 95.4 MBit/s (11.9 MByte/s). Der Test zeigt, daß die Verwendung von AAL 3/4 nicht zu empfehlen ist. Bei kleinen Nachrichtenlängen ist die Leistung unter AAL 5 bei HP wesentlich besser, erst ab 3 KByte sind die IBM-Werte gleich gut oder sogar etwas besser.

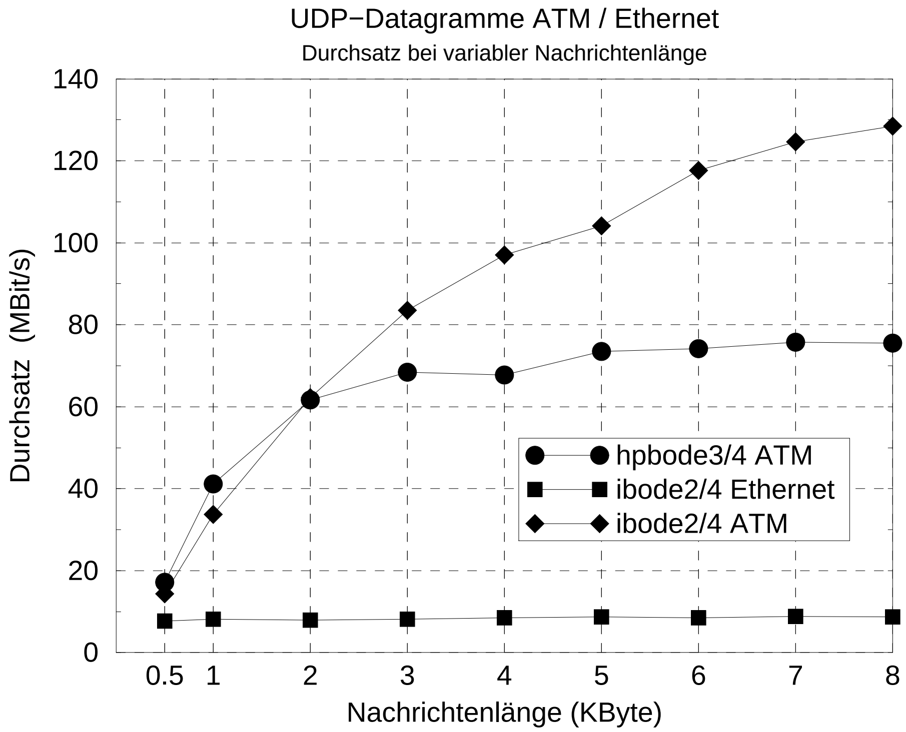

Abbildung 5.5: UDP: Durchsatz bei variabler Nachrichtenlänge

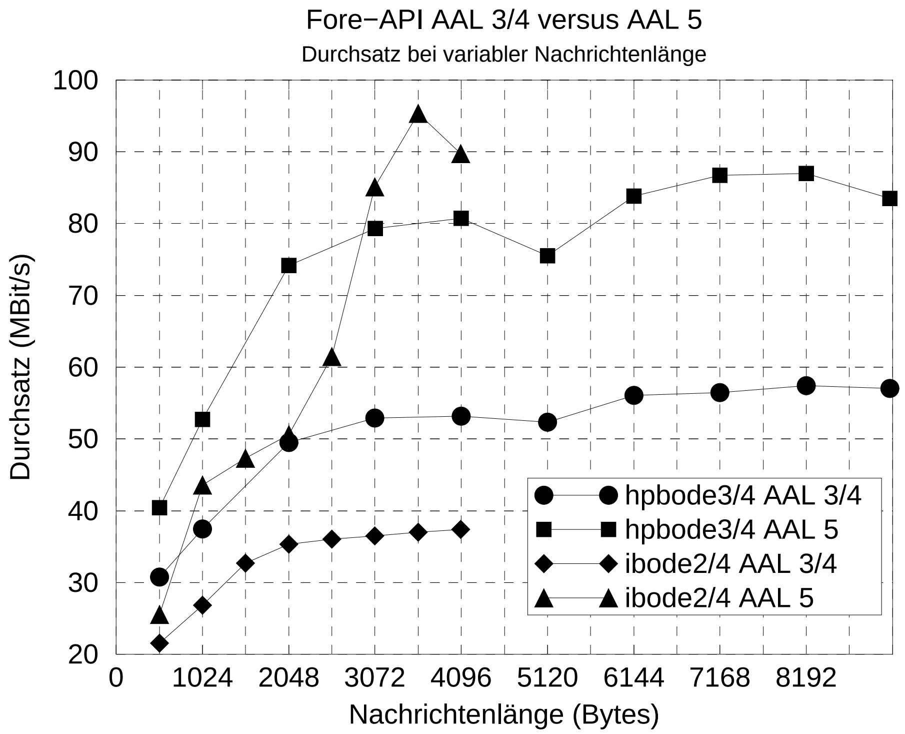

Abbildung 5.6: Fore-API AAL 3/4, AAL 5: Durchsatz bei variabler Nachrichtenlänge

5.2.5 Zusammenfassung

ATM läßt das Ethernet, was den erreichbaren Durchsatz unter TCP/IP anbelangt, weit hinter sich. Die Implementierung von *IP over ATM* unter AIX zeigt sowohl bei TCP als auch bei UDP sehr gute Werte. Die HP-Maschinen bieten hier weit weniger Leistung. Obwohl das Fore-API im Vergleich zu *IP over ATM* weniger Schichten und somit einen geringeren Protokoll-Overhead hat, konnten die gemessenen Durchsätze nicht überzeugen. Die HP-Maschinen erreichen über das Fore-API zwar den höchsten Durchsatz, der aber nicht in die Nähe des theoretisch möglichen Durchsatzes kommt. Vergleiche hierzu [6].

5.3 HP-Benchmark: netperf

Netperf ist ein Benchmark zur Messung verschiedener Leistungswerte eines Kommunikationsnetzes. Er wurde von der *Information Networks Division* bei Hewlett-Packard entwickelt. Standardmäßig wird die Messung der Durchsatzleistung bei der Übertragung großer Datenmengen (*bulk data transfer*) und die Messung von Antwortzeiten (*request/response performance*) unter den Protokollen UDP und TCP unterstützt. Der Benchmark enthält optionale Tests unter anderem zur Leistungsmessung des Fore-API. Für die Messungen in dieser Arbeit wurde die Revision 2.1 des Benchmark vom 15.02.1996 eingesetzt. Die Hewlett-Packard Company hat das Copyright an der Software. Für den Public-Domain-Gebrauch ist sie per ftp[6] erhältlich. Das Paket beinhaltet auch das im Literaturverzeichnis angegebene Manual [19] `netperf.ps` im Format PostScript.

Die Architektur von netperf basiert auf dem Client-Server-Modell. Auf einem Rechner wird der Client `netperf` gestartet. Auf dem anderen Rechner läuft der Server `netserver`. Beim Starten des Client wird eine Kontrollverbindung (TCP, BSD-Sockets) zum entfernten Server aufgebaut. Über diese Verbindung werden Konfigurationsinformationen und Testergebnisse zwischen Client und Server ausgetauscht. Für den eigentlichen Test wird anschließend eine separate Verbindung gemäß dem angegebenen Protokoll oder API über das Netz aufgebaut. Während eine Messung aktiv ist, wird auf der Kontrollverbindung kein Verkehr erzeugt.

Bei netperf wird für eine Messung die Dauer der Datenübertragung in Sekunden angegeben im Gegensatz zu nttcp, wo die insgesamt zu übertragene Datenmenge spezifiziert wird. Der zu messende Durchsatz errechnet sich dann aus dem während der Testdauer übertragenen Datenvolumen. Der Sender- und Empfängerdurchsatz wird separat ermittelt. Zusätzlich unterstützt netperf die Messung von Antwortzeiten über sog. Transaktionen. Eine Transaktion ist definiert als der Austausch einer einzelnen Anfrage-Nachricht (*request*) und einer einzelnen Antwort-Nachricht (*response*) zwischen Client und Server. Aus der Transaktionenrate bei einer bestimmten Testdauer und Nachrichtenlänge für Anfrage und Antwort kann die durchschnittliche Antwortzeit (*round trip time*) und die Latenzzeit einer Einzelnachricht abgeleitet werden.

Die Unterstützung des Fore-API erforderte eine Neuübersetzung der Benchmark-Software. Der Quelltext von `nettest_fore.c` und `nettest_fore.h` hierfür enthielt einige Fehler bei den verwendeten Datenstrukturen und ließ sich erst nach einer Korrektur übersetzen.

[6] `ftp://ftp.cup.hp.com/dist/networking/benchmarks`

Zur Vereinfachung der Messungen werden für die unterschiedlichen Tests Shell-Skripte mitgeliefert. Die für die Tests relevanten Meßparameter von netperf können hiermit komfortabel angepaßt werden. Die Skripte ermöglichen auch die automatisierte Erstellung von Testreihen. Im Anhang C finden sich die Testkonfigurationen der mit netperf durchgeführten Messungen. Eine genaue Erklärung der Parameter ist dem Manual [19] zu entnehmen.

Die mit netperf gewonnen Durchsatzwerte sollen die Ergebnisse der Durchsatzmessungen mit nttcp überprüfen bzw. ergänzen. Zusätzlich wurden Latenzzeitmessungen durchgeführt. Die Testdauer der einzelnen Tests wurde hinreichend groß gewählt (meist 30 Sekunden), um unverfälschte Meßwerte zu erhalten.

5.3.1 Test 5: TCP-Stream-Durchsatz (hpbode3/4)

Test 5 kombiniert die Messungen von Test 1 und Test 2 unter nttcp für TCP-Verbindungen über ATM zwischen hpbode3 und hpbode4. Es wurde der Durchsatz gemessen bei variablen Nachrichtenlängen für vier verschiedene Puffergrößen der Socket-Sende- und Empfangspuffer.

Die Nachrichtenlänge wurde ausgehend vom Startwert 512 Bytes bis zum Endwert von 64 KByte jeweils verdoppelt. Die Größe der Socket-Puffer wurde einheitlich beim Sender und Empfänger eingestellt. Es wurden die Größen 8 KByte (default), 16 KByte, 32 KByte und 56 KByte (Maximalwert) betrachtet. Die Socket-Option `TCP_NODELAY` war gesetzt.

Abbildung 5.7 zeigt die Meßwerte für Test 5. Während der Durchsatz bei der maximalen Puffergröße und einer Nachrichtenlänge von 16 KByte den Spitzenwert von 80.7 MBit/s (10.1 MByte/s) erreicht, beträgt er bei gleicher Nachrichtenlänge und Standardpuffergröße nur 47.2 MBit/s (5.9 MByte/s). Dies zeigt wiederum die Notwendigkeit von großen Socket-Puffern bei ATM. Diese sollten außerdem größer als die Nachrichtenlänge sein. Im Diagramm sinkt der Durchsatz, sobald die Nachrichtenlänge größer oder gleich der Socket-Puffergröße ist. Die gemessenen Werte sind etwas besser als bei den Tests mit nttcp.

5.3.2 Test 6: TCP-Stream-Durchsatz (ibode2/4)

Dieser Test ist analog Test 5 für die Maschinen ibode2 und ibode4. Bis auf den Maximalwert für die Socket-Puffergrößen, der bei AIX 64 KByte beträgt, herrschen gleiche Testbedingungen. Das Diagramm der Meßwerte (Abbildung 5.8) zeigt aber ein unterschiedliches Verhalten der IBM-Rechner. Während bei den HP-Rechnern die Durchsatzwerte zurückgehen, sobald die Nachrichtenlänge die Socket-Puffergröße übersteigt, bleibt der Durchsatz hier für jede Puffergröße ab einer Nachrichtenlänge von 8 KByte konstant. Die besten Werte – 125.4 MBit/s (15.7 MByte/s) – werden auch bei diesem Test bei der maximalen Puffergröße erreicht. Dieser Maximalwert ist sinnvoll ab einer Nachrichtenlänge von ca. 1.5 KByte. Im Gegensatz zu Test 5 sind die hier gemessenen Werte im Vergleich zu nttcp etwas geringer mit Ausnahme der Spitzenwerte. Der Spitzendurchsatz ist zwischen den IBM-Rechnern um die Hälfte größer als zwischen den HP-Rechnern. Hingegen ist die Leistung bei Nachrichten, die kleiner als 4 KByte sind, bei den HP-Maschinen erheblich besser.

Auf die Messung des Durchsatzes von UDP über ATM wurde verzichtet. Netperf bietet zwar auch die Möglichkeit, die Senderate zu beeinflussen, wenn es mit der Option `-DINTERVALS` im

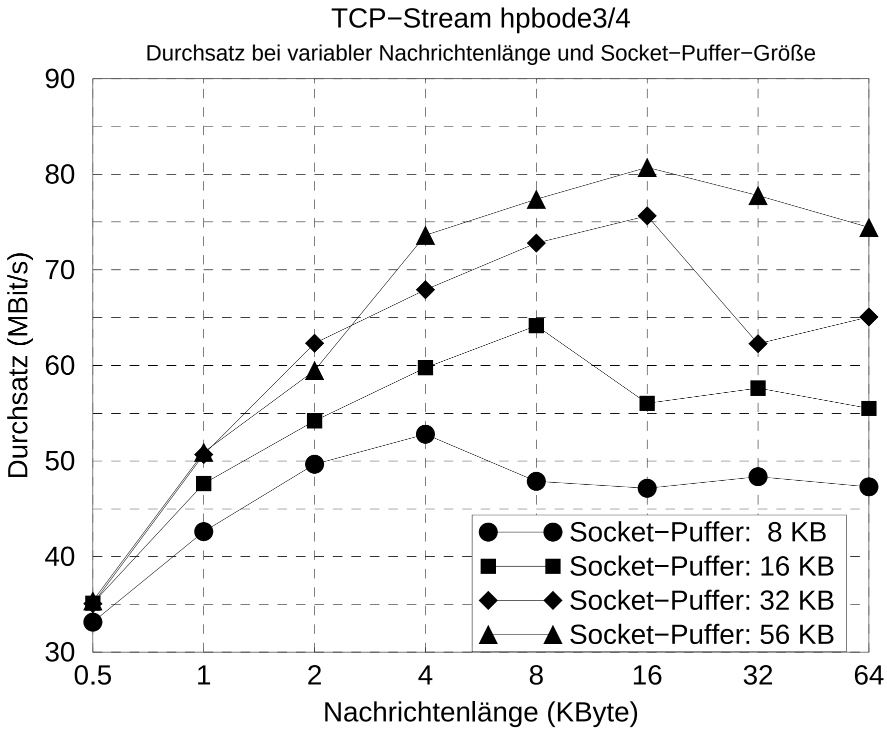

Abbildung 5.7: TCP: Durchsatz von hpbode3/4 für unterschiedliche Socket-Puffergrößen

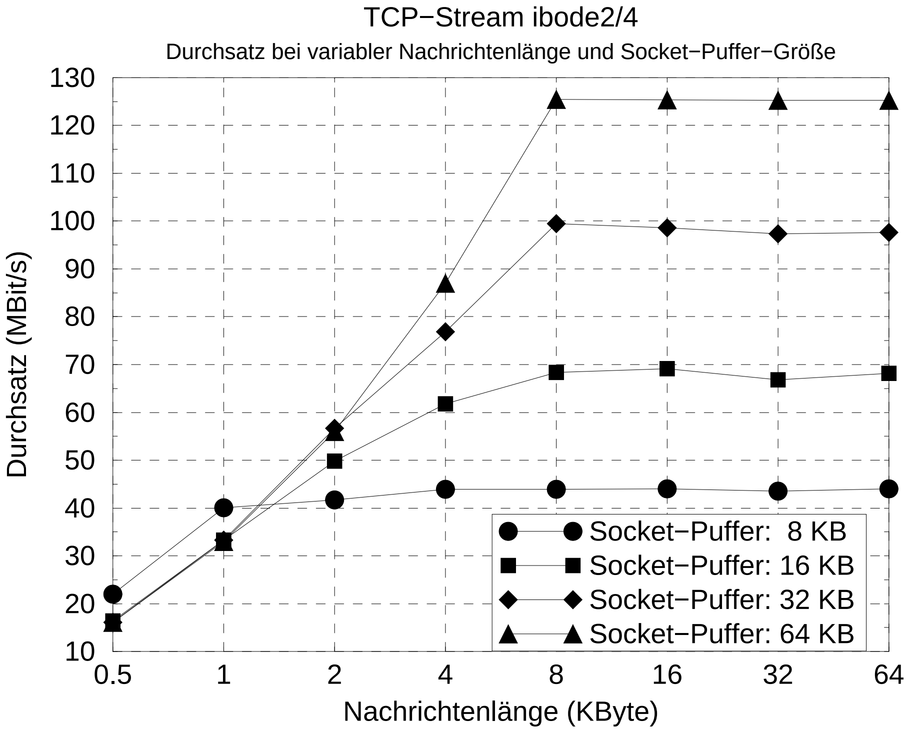

Abbildung 5.8: TCP: Durchsatz von ibode2/4 für unterschiedliche Socket-Puffergrößen

Makefile neu übersetzt wird. Hiervor sei allerdings ausdrücklich gewarnt. Unter AIX stürzt der Rechner, auf dem der `netserver` läuft, komplett ab und muß neu gebootet werden, wenn diese Funktion genutzt wird. Dieser Effekt konnte reproduziert werden und deutet auf einen Fehler in der Implementierung von *IP over ATM* unter AIX hin. Ebenso scheiterten Experimente mit dieser Funktion unter HP-UX. Es konnten keine vernünftigen Werte für den Empfängerdurchsatz gemessen werden. Die folgenden Tests behandeln daher das Fore-API.

5.3.3 Test 7: Fore-API (AAL 3/4, AAL 5)

Dieser Test ist vergleichbar mit Test 4 unter nttcp. Auch hier wurden die Werte für den Durchsatz bei Verwendung des Fore-API mit variabler Nachrichtenlänge und unterschiedlichen AAL-Typen 3/4 und 5 zwischen den HP- und den IBM-Maschinen gemessen. Während in Test 4 die Senderate manuell beeinflußt wurde, um den Datenverlust zu begrenzen, wurde diesmal beobachtet, wie hoch der Empfängerdurchsatz ist, wenn der Client ungebremst Daten über das Fore-API sendet. Bei netperf kann die Senderate nicht beeinflußt werden.

Die Nachrichtenlänge wurde vom Startwert 256 Bytes in 256-Bytes-Schritten bis zur MTU der jeweiligen Implementierung unter AIX (4092 Bytes) und HP-UX (9188 Bytes) gesteigert. Die Meßwerte für den erzielten Durchsatz sind in Abbildung 5.9 zusammengefaßt.

Die Ergebnisse werden mit denen von Test 4 (Abbildung 5.6) verglichen. Bis zu einer Nachrichtenlänge von 3 KByte werden für die HP-Maschinen bei AAL 3/4 kleinere Werte gemessen. Diese sind auf Verluste beim Sender (z.B. innerhalb des Device-Treibers) durch Pufferüberläufe zurückzuführen. Außerdem zeigen sich zwei deutliche Einbrüche bei Blöcken, die jeweils etwas größer als 2 bzw. 3 KByte sind. Ab 4 Kbyte treten keine Verluste mehr beim Sender auf, da die Werte mit denen von Test 4 ungefähr übereinstimmen. Die HP-Kurve bei AAL 5 zeigt kaum Einbrüche und gleicht der von Test 4. Ohne Flußkontrolle erreichen bei Übertragungen über das Fore-API zwischen den IBM-Maschinen nur noch sehr wenige Nachrichten den Empfänger. Entsprechend schlecht sind die gemessenen Werte. Bei AAL 3/4 bleibt der Empfängerdurchsatz unter 20 MBit/s, in Test 4 liegt er durchwegs darüber. Auch bei AAL 5 ist der Durchsatz erheblich kleiner gegenüber Test 4, erst nahe der MTU werden die gleichen Werte erreicht. Interessant ist auch, daß bei Nachrichtenlängen um 1 KByte sowohl bei AAL 3/4 als auch bei AAL 5 sämtliche Pakete verloren gehen. Der Test macht deutlich, daß eine Anwendung, die größere Datenmengen über das Fore-API übertragen soll, unbedingt eine Flußkontrolle und Mechanismen zur Fehlerkorrektur implementieren muß. In [6] ist eine Transportschicht (*ATM Transport Layer – ATL*) für die Fore-ATM-Gerätetreiber beschrieben, die diese Funktionen effizient erfüllt.

5.3.4 Test 8, 9: Fore-API (AAL 5): Sender- und Empfängerdurchsatz

Die folgenden zwei Tests behandeln ebenfalls das Fore-API (AAL 5). Es werden die Senderdurchsätze mit den Empfängerdurchsätzen jeweils für hpbode3/4 und ibode2/4 miteinander verglichen. Betrachtet wurden alle Nachrichtenlängen bis zur MTU in kleinen Schritten von 8 Bytes (AIX) bzw. 16 Bytes (HP-UX). Die beiden Diagramme zeigen völlig unterschiedliche Kurven.

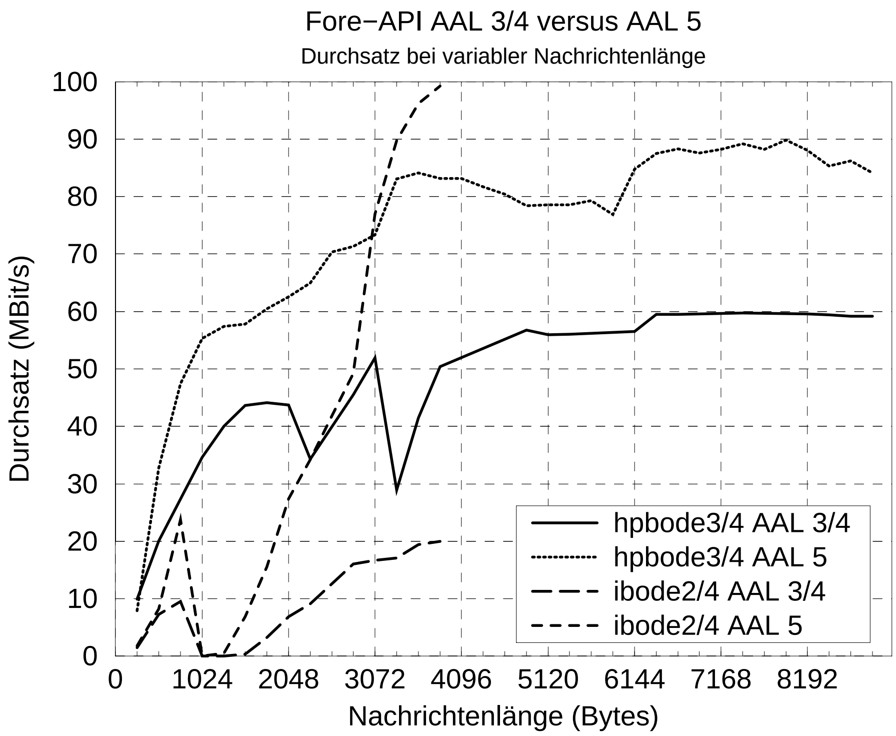

Abbildung 5.9: Fore-API: Vergleich des Durchsatzes unter AAL 3/4 und AAL 5

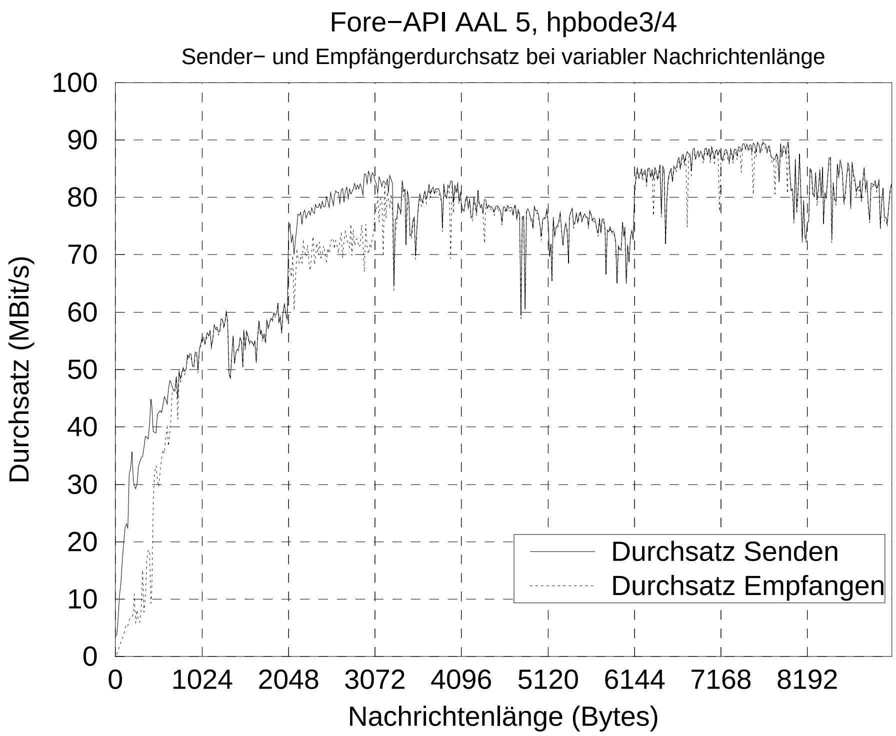

Abbildung 5.10: hpbode3/4: Sender- und Empfängerdurchsatz für Fore-API AAL 5

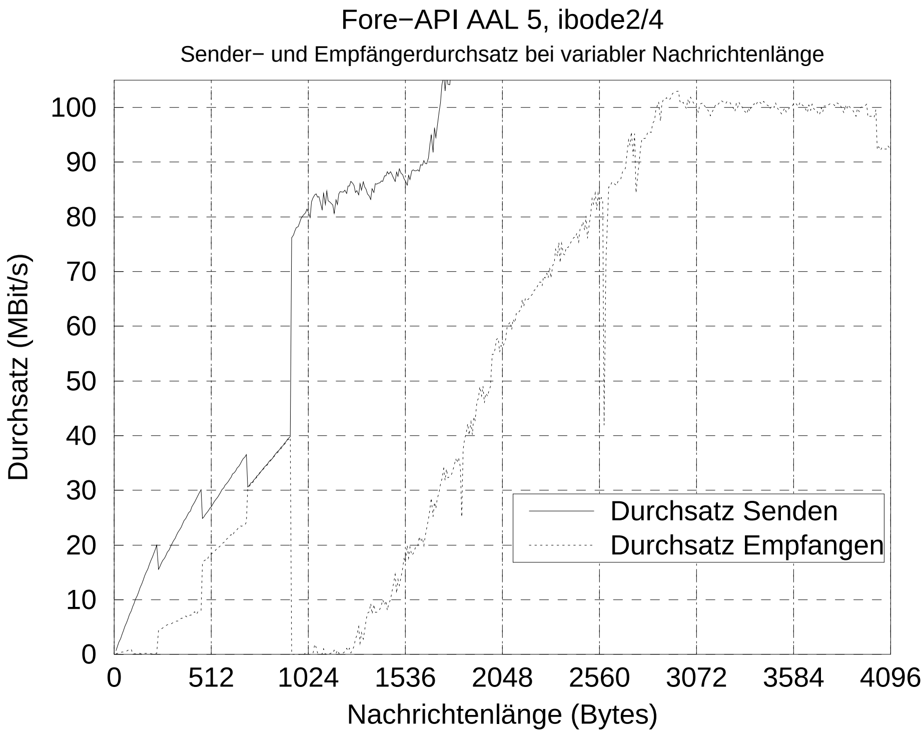

Abbildung 5.11: ibode2/4: Sender- und Empfängerdurchsatz für Fore-API AAL 5

Bei der HP-Messung, die in Abbildung 5.10 dargestellt ist, überdecken sich die Kurven für den Sender- und Empfängerdurchsatz bis auf zwei Bereiche. Bis zur Nachrichtenlänge von etwa 750 Bytes und im Bereich von 2048 bis 3100 Bytes wird ein erheblich größerer Senderdurchsatz gemessen. Ansonsten gibt es nur bei isolierten Meßwerten Abweichungen in der Größenordnung von 10 MBit/s. Andererseits schwanken die gemessenen Durchsatzwerte stark. Zwischen zwei benachbarten Meßwerten gibt es Unterschiede von bis zu 20 MBit/s.

Die Kurven der IBM-Messung in Abbildung 5.11 überdecken sich nur in einem kleinen Intervall um Nachrichtenlängen von 700 Bytes. In allen anderen Bereichen ist der Senderdurchsatz sehr viel höher als der Empfängerdurchsatz. Bei dem Einbruch der Empfängerdatenrate bei Nachrichten der Größe um 1 KByte handelt es sich um ein reproduzierbares Phänomen, das bereits in Test 7 (Abbildung 5.9) beobachtet wurde. Der Senderdurchsatz überschreitet bei einer Nachrichtenlänge von etwa 2 KByte den theoretisch maximalen Wert von 130 MBit/s des ATM-Interface und steigt weiter bis über 270 MBit/s an. Der Durchsatz wird an der Schnittstelle von Anwendung (netperf) und Fore-API gemessen. Auf der Senderseite ist dies die API-Funktion *atm_send()*. Die Funktion ist in den beiden Betriebssystemen HP-UX und AIX augenscheinlich unterschiedlich implementiert. Unter HP-UX wird die Anwendung bzw. der Aufruf von *atm_send()* blockiert, falls der Sendepuffer des ATM-Gerätetreibers voll ist. Die Einbrüche des Empfängerdurchsatzes in Abbildung 5.10 entstehen durch ein Überlaufen des Empfangspuffers, wenn die Anwendung die Daten nicht schnell genug mittels *atm_recv()* abholt. (Vergleiche hierzu [18], Abschnitt *API – Application Programming Interface.*) In der Implementierung unter AIX wird *atm_send()* nicht blockiert, eine Anwendung kann daher den ATM-Gerätetreiber mit Paketen überfluten. Hieraus resultiert der unsinnig hohe Senderdurchsatz auf der Anwendungsschicht.

Der Empfängerdurchsatz steigt bei der ibode4 bei Nachrichtenlängen im Bereich zwischen etwa 1200 Bytes und 3072 Bytes monoton bis ca. 100 MBit/s und bleibt dann bis zur MTU konstant auf diesem Niveau. Es gibt neben zwei kleinen nur einen drastischen Einbruch (40 MBit/s) bei 2568 Bytes. Die übrigen sägezahnförmigen Spitzen haben ein Intervall von 48 Bytes. Immer wenn die Nachrichtenlänge ein Vielfaches der Nutzlänge einer Zelle (48 Bytes) gerade überschreitet, gibt es einen kleinen Rückgang beim Durchsatz, da zur Übertragung eine zusätzliche Zelle nötig ist. Die Schwankungen, die bei den HP-Rechnern auftreten, werden hier nicht beobachtet.

Die folgenden Tests beschäftigen sich mit Latenzzeitmessungen.

5.3.5 Test 10 - 13: Latenzzeitmessungen mit netperf

Die Latenzzeit von Nachrichten kann mit netperf durch sog. Transaktionen gemessen werden. Eine Transaktion setzt sich aus einer Anfrage (*request*) des Client und einer Antwort des Server (*response*) zusammen. Netperf ermittelt über die gesamte Testdauer die Transaktionenrate, das ist die Anzahl der Transaktionen pro Sekunde. Hieraus kann die Antwortzeit für eine Transaktion errechnet werden. Ist die Nachrichtenlänge von Anfrage und Antwort gleich, beträgt die Latenzzeit für eine Einzelnachricht ungefähr die Hälfte der Antwortzeit. Die Größe für Anfrage und Antwort kann in netperf getrennt eingestellt werden.

In Test 10 (hpbode3/4) und Test 11 (ibode2/4) wurden die Latenzzeiten für kleine Nachrichten (1, 16, 64 und 128 Bytes) bei der Übertragung über das Fore-API (AAL 3/4 und AAL 5) und über TCP und UDP (ATM und Ethernet) gemessen. TCP ist ein verbindungsorientiertes Protokoll und eigentlich kein Nachrichtenprotokoll wie UDP. Netperf mißt die Transaktionenrate *einer* bestehenden TCP-Verbindung und baut nicht für jede Transaktion eine neue Verbindung auf. Genauso verhält es sich mit dem Fore-API.

Test 10 (hpbode3/4): Siehe Abbildung 5.12. Die kleinsten Latenzzeiten um 200 μs weist jeweils das Fore-API auf. Bei Typ AAL 5 ergeben sich minimal bessere Werte als bei AAL 3/4. Geringfügig höher sind die Latenzzeiten unter TCP. TCP über ATM (233 μs) zeigt hier nur geringe Vorteile gegenüber dem Ethernet (285 μs) bei jeweils 4-Bytes-Nachrichtenlänge. Dagegen sind die Werte für UDP/Ethernet um ein Vielfaches höher als für UDP/ATM. Da UDP beim Austausch von Nachrichten in der Praxis eine größere Rolle spielt als TCP, ist somit ein erheblicher Leistungszuwachs beim Einsatz von ATM möglich. Der Umstand, daß zur Übertragung einer 128 Bytes langen Nachricht drei bzw. vier ATM-Zellen gegenüber einer bei einer 1 Byte langen Nachricht nötig sind, wirkt sich bei den Latenzzeiten kaum aus. Die Latenzzeiten werden primär durch den Overhead des eingesetzten Protokolls beeinflußt. Die genauen Meßwerte für diesen Test finden sich in Tabelle 5.1.

Test 11 (ibode2/4): Siehe Abbildung 5.13. Auch hier sind die Latenzzeiten des Fore-API für beide AAL-Typen am kleinsten. Bis zu einer Nachrichtenlänge von 64 Bytes werden etwa 340 μs gemessen. Die Werte von UDP/ATM sind etwas besser als die von TCP/ATM. Bei den HP-Rechnern war es umgekehrt. Im Vergleich zum (unbelasteten) Ethernet schneidet ATM bei diesem Test aber nur geringfügig besser ab. Mit über

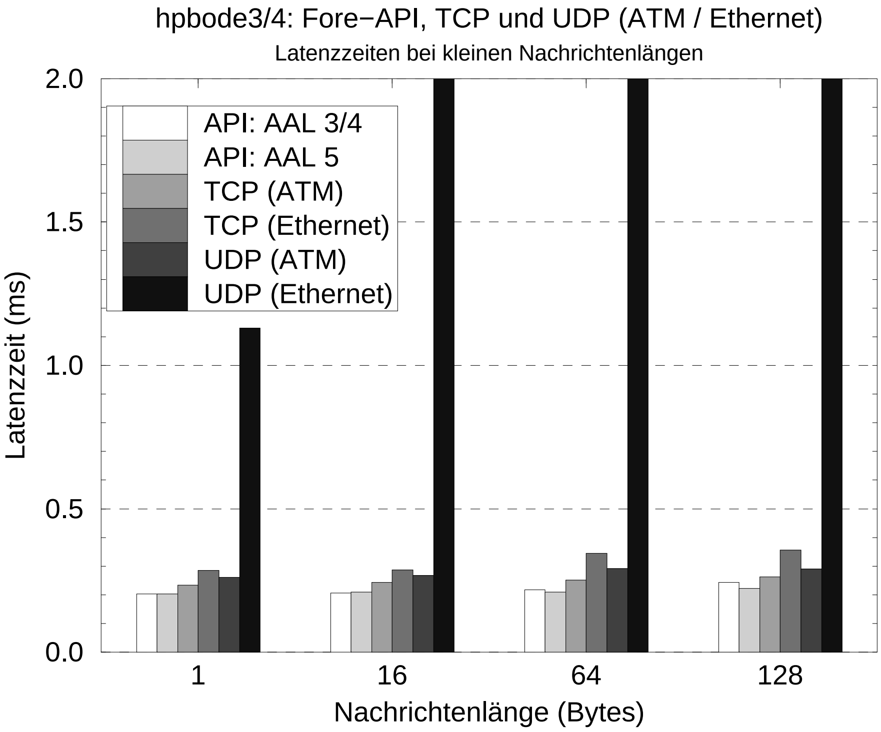

Abbildung 5.12: hpbode3/4: Latenzzeiten für kleine Nachrichtenlängen

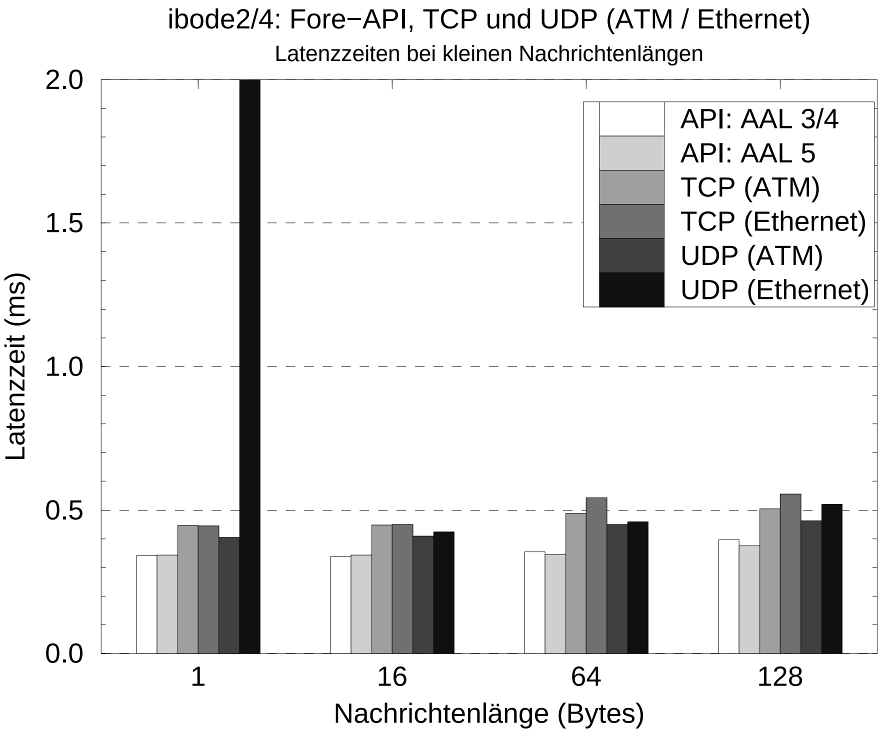

Abbildung 5.13: ibode2/4: Latenzzeiten für kleine Nachrichtenlängen

45 ms weicht der Wert von UDP/Ethernet vom übrigen Bild stark ab. Insgesamt sind die Latenzzeiten der Übertragungen über ATM bei IBM fast doppelt so groß als bei HP. Alle Meßwerte sind in Tabelle 5.2 zusammengefaßt.

In Test 12 und Test 13 wurden die Latenzzeiten von größeren Nachrichten gemessen. Die betrachteten Nachrichtenlängen sind 256, 512 Bytes, 1, 4 und 8 KByte. Bei Verwendung des Fore-API darf die Nachrichtenlänge die MTU nicht überschreiten.

Test 12 (hpbode3/4): Siehe Abbildung 5.14. Mit zunehmender Nachrichtenlänge werden die Latenzzeiten des Ethernet im Vergleich zu ATM immer schlechter. Bei größeren Nachrichten macht sich auch der größere Overhead von AAL 3/4 gegenüber AAL 5 bei Übertragung über das Fore-API bemerkbar. Interessant ist aber, daß ab einer Nachrichtenlänge von 1 KByte TCP/ATM und UDP/ATM bessere Latenzzeiten aufweisen als das Fore-API/AAL 5, obwohl dieses den kleineren Protokoll-Overhead haben müßte. Hier ist die Implementierung von *IP over ATM* von Fore besser als das eigene API. Die in Abb. 5.14 bei 8 KByte dargestellten Meßwerte des Fore-API beziehen sich auf die maximale Nachrichtenlänge 9188 Bytes (MTU). Für die genauen Werte siehe Tabelle 5.1.

Test 13 (ibode2/4): Siehe Abbildung 5.15. Das Fore-API zeigt das gleiche Verhalten bezüglich des AAL-Typs wie im Test 12 zwischen den HP-Rechnern. Aufgrund der kleineren MTU des Fore-API bei AIX, kann nicht überprüft werden, ob die Latenzzeiten großer Nachrichten (ab 4 KByte) bei IP/ATM besser sind als beim Fore-API. Die Meßwerte für das Fore-API bei 4-KByte-Nachrichtenlänge beziehen sich auf die MTU von 4092 Bytes. Während die HP-Latenzzeiten bei kleinen Nachrichten insgesamt besser als die der IBM-Rechner sind, verschwindet dieser Vorteil ab Größen von 8 KByte. Dieser Effekt resultiert aus den höheren Durchsatzwerten, die zwischen ibode2/4 gemessen wurden. In Tabelle 5.2 finden sich alle gemessenen Latenzzeiten.

Im letzten Abschnitt dieses Kapitels wird untersucht, ob die bisher gewonnenen Ergebnisse auf die verteilte Programmierumgebung PVM übertragbar sind. Hierzu wird ein PVM-Benchmark eingesetzt.

Nachrichtenlänge	1 B	16 B	64 B	128 B	256 B	512 B	1 KB	4 KB	8 KB
API: AAL 3/4	0.203	0.206	0.218	0.243	0.290	0.351	0.447	0.965	1.880[7]
API: AAL 5	0.203	0.210	0.210	0.222	0.267	0.301	0.408	0.692	1.340[7]
TCP (ATM)	0.233	0.243	0.251	0.262	0.310	0.337	0.384	0.758	1.131
TCP (Ethernet)	0.285	0.287	0.345	0.356	0.567	0.787	1.309	4.717	9.091
UDP (ATM)	0.261	0.268	0.292	0.290	0.330	0.352	0.420	0.787	1.316
UDP (Ethernet)	1.131	7.576	16.667	2.809	3.448	4.902	5.319	71.429	97.345

Tabelle 5.1: Latenzzeiten (ms) für hpbode3/4 (Test 10, 12)

[7] Der Wert bezieht sich auf die Nachrichtenlänge 9188 Bytes (MTU).
[8] Der Wert bezieht sich auf die Nachrichtenlänge 4092 Bytes (MTU).
[9] Kein Meßwert, da Nachrichtenlänge größer als MTU!

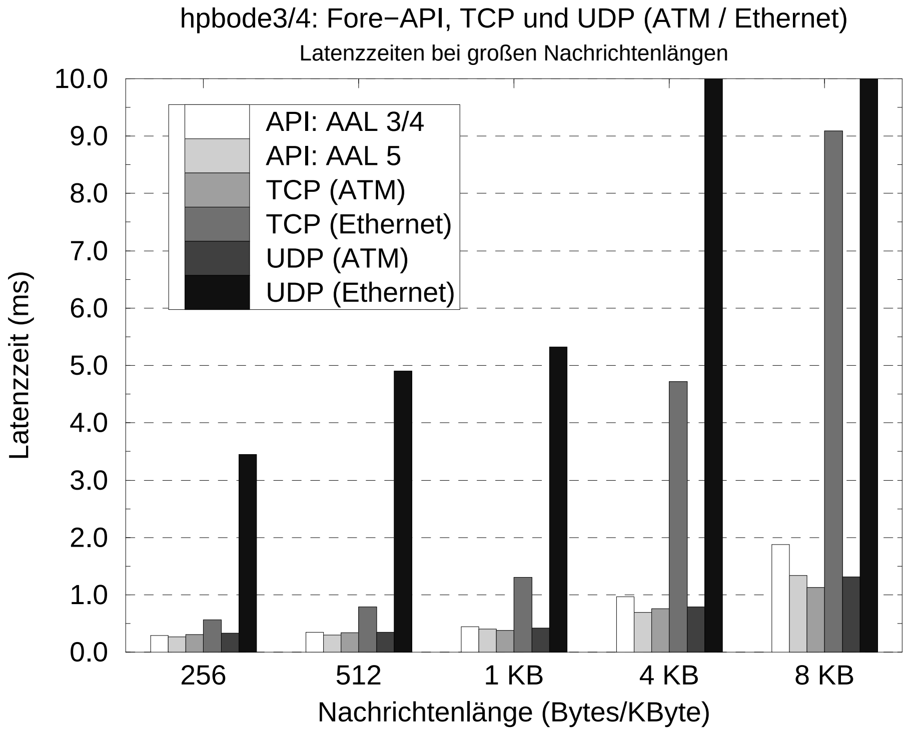

Abbildung 5.14: hpbode3/4: Latenzzeiten für große Nachrichtenlängen

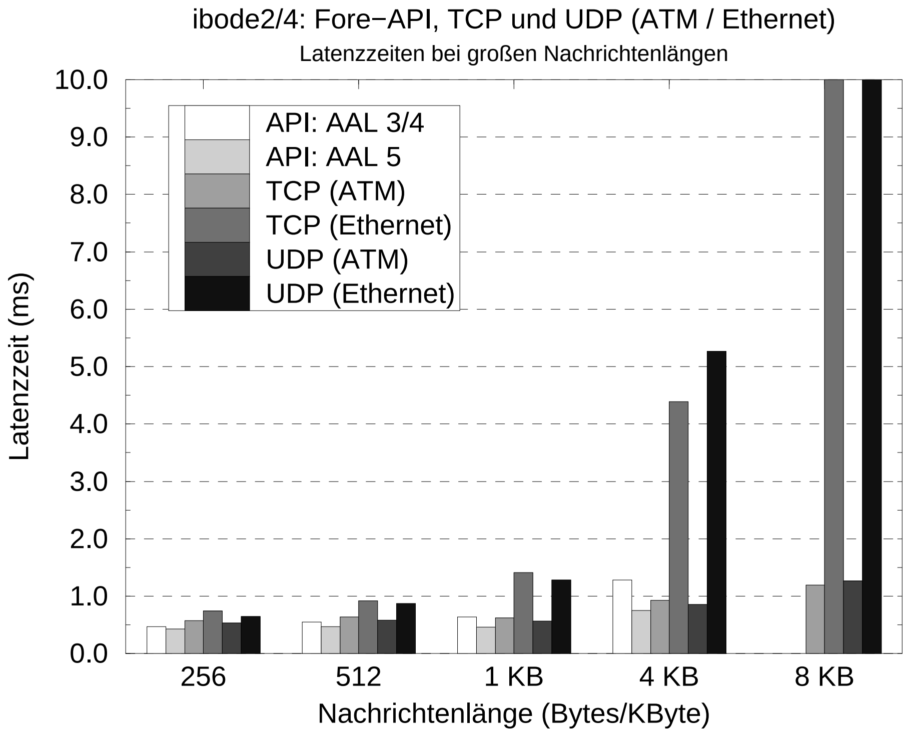

Abbildung 5.15: ibode2/4: Latenzzeiten für große Nachrichtenlängen

Nachrichtenlänge	1 B	16 B	64 B	128 B	256 B	512 B	1 KB	4 KB	8 KB
API: AAL 3/4	0.342	0.339	0.355	0.397	0.469	0.548	0.637	1.282[8]	-[9]
API: AAL 5	0.343	0.343	0.345	0.376	0.425	0.469	0.460	0.750[8]	-[9]
TCP (ATM)	0.447	0.448	0.488	0.504	0.577	0.637	0.623	0.929	1.190
TCP (Ethernet)	0.445	0.450	0.543	0.556	0.742	0.916	1.412	4.386	10.136
UDP (ATM)	0.405	0.409	0.450	0.462	0.534	0.579	0.562	0.856	1.269
UDP (Ethernet)	45.455	0.424	0.459	0.521	0.644	0.874	1.285	5.263	62.485

Tabelle 5.2: Latenzzeiten (ms) für ibode2/4 (Test 11, 13)

5.4 Der PVM-Benchmark

Bei dem PVM-Benchmark handelt es sich um ein Echo-Programm, welches von den PVM-Entwicklern Manchek und Geist geschrieben wurde. Am *Lewis Research Center* der NASA wurden damit Durchsatz- und Latenzzeitmessungen durchgeführt und die Ergebnisse mit denen von anderen parallelen Programmierumgebungen (z.B. MPI) und anderen Hochleistungsnetzen (z.B. FDDI) verglichen. Der Code für den PVM-Benchmark und die Meßergebnisse sind im World-Wide-Web[10] veröffentlicht.

5.4.1 Funktionsweise des PVM-Benchmarks

Der PVM-Benchmark besteht aus den drei Dateien `timing.c`, `timing_node.c` und `timing.h`. Den Quellcode für den Echo-Client enthält `timing.c`, den für den Echo-Server `timing_node.c`. In `timing.h` können Parameter für den Benchmark eingestellt werden. Der Client startet mittels *pvm_spawn()* den Server auf dem anderen Knoten der virtuellen Maschine. Anschließend startet er die Zeitmessung und sendet eine Nachricht der Länge 0 an den Server (*pvm_psend()*). Der Server empfängt die Nachricht und schickt sie sofort an den Client zurück. Nach dem Empfang stoppt der Client die Zeit. Dieser Vorgang wird n-mal wiederholt. Alle Antwortzeiten werden aufsummiert. Aus der Gesamtsumme wird die durchschnittliche Latenzzeit einer Nachricht und der durchschnittliche Durchsatz für diese Nachrichtenlänge berechnet. Danach werden n Nachrichten der Länge 1 Byte, 2 Bytes, 4 Bytes usw. ausgetauscht. Die maximale Nachrichtenlänge (`MAX`) und die Anzahl n der Wiederholungen (`REPS`) werden in der Datei `timing.h` spezifiziert.

Abhängig vom gewählten Modus für das Routing der Nachrichten innerhalb der virtuellen Maschine (siehe Kapitel 4.2), muß PVM große Nachrichten geeignet fragmentieren. Beim normalen Routing-Modus (*PvmRouteDefault*) wird die Kommunikation über UDP-Sockets abgewickelt. Die maximale UDP-Paketgröße für PVM ist durch die Konstante `UDPMAXLEN` in `$PVM_ROOT/src/global.h` festgelegt und hat den generischen Wert 4 KByte. Sie kann an die maximale Größe eines IP-Pakets, die vom Betriebssystem unterstützt wird, angepaßt werden. Beim Direktmodus für das Routing (*PvmRouteDirect*) tauschen die PVM-Prozesse Nachrichten über TCP-Sockets aus. PVM benutzt hierfür eine Socket-Puffergröße von 32 KByte. Dieser Wert ist in der Konstante `TTSOCKBUF` in der Quelldatei `$PVM_ROOT/src/lpvm.c` definiert. Außerdem setzt PVM die Socket-Option `TCP_NODELAY`.

Für die Messungen in dieser Arbeit wurden die Standardwerte für die UDP-Paketgröße und

[10] `http://www.lerc.nasa.gov/WWW/ACCL/PARALLEL/benchmark.html`

die TCP-Socket-Puffergröße verwendet, um vergleichbare Meßwerte zu erhalten. Experimente mit der UDP-Paketgröße (z.B. 8 KByte, 16 KByte) ergaben keine höheren Leistungswerte. Eine Anpassung der TCP-Socket-Puffergröße ist nur in homogenen Umgebungen sinnvoll, da der Maximalwert vom Betriebssystem (z.B. HP-UX: 56 KByte, AIX: 64 KByte) abhängig ist. Für den Benchmark wurde das Austauschformat der Nachrichten auf *PvmDataRaw* gesetzt, da nur innerhalb einer Architektur gemessen wurde. Außerdem werden die Meßwerte somit nicht durch zusätzlichen Overhead für die Konvertierung in das XDR-Format (*PvmDataDefault*) verfälscht. Nähere Informationen zu den Formaten finden sich in [11]. Für die Durchsatzmessungen wurde der Maximalwert für die Nachrichtenlänge auf 128 KByte eingestellt. Pro Meßwert wurden jeweils hundert Nachrichten ausgetauscht. Der Durchsatz wird vom PVM-Benchmark in der Einheit MByte/s[11] berechnet. Die Meßwerte sind somit vergleichsweise 5% höher als bei den Messungen mit nttcp[12].

5.4.2 Test 14 – 17: Durchsatzmessungen mit dem PVM-Benchmark

In den ersten beiden Tests wurde der Durchsatz für den Routing-Modus *PvmRouteDefault* zwischen hpbode3/4 bzw. ibode2/4 gemessen. Zum Einsatz kam die PVM-Version 3.3.7 für die Tests UDP/Ethernet und UDP/ATM und die in Kapitel 4.3 beschriebene PVM-Version der *Universität von Minnesota* für den Fore-API-Test. Die Diagramme für Test 14 und 15 (Abbildungen 5.16 und 5.17) zeigen den gemessenen Durchsatz bei Nachrichtenlängen zwischen 64 Bytes und 128 KByte.

Test 14 (hpbode3/4, PvmRouteDefault): Siehe Abbildung 5.16. Der Durchsatz unter ATM steigt ab Nachrichten der Länge 256 Bytes stark an. Aber erst ab einer Länge von 1 KByte zeigt die Übertragung über ATM wesentlich bessere Werte als über das weitgehend unbelastete Ethernet. Bei 128-KByte-Nachrichten beträgt der Durchsatz über das Fore-API 2.9 MByte/s. UDP über ATM liegt mit 2.5 MByte/s knapp darunter. Bei noch größeren Nachrichten wurde keine weitere Steigerung im Durchsatz beobachtet, wie auch Messungen in [13] gezeigt haben.

Test 15 (ibode2/4, PvmRouteDefault): Siehe Abbildung 5.17. Auch hier erreicht das Fore-API im Bereich von 4 KByte – 64 KByte etwas höhere Werte als UDP/ATM und wird erst bei Nachrichten der Länge 128 KByte eingeholt. Die Maximalwerte von 1.8 MByte/s für das Fore-API und 1.9 MByte/s für UDP/ATM liegen allerdings unter den Werten, die zwischen den HP-Rechnern gemessen wurden. Signifikant ist der Einbruch von UDP/ATM bei 4 KByte, der abgeschwächt auch bei UDP/Ethernet zu beobachten ist. Dies scheint mit der maximalen UDP-Paketlänge von 4 KByte bei PVM zusammenzuhängen. Ursache hierfür ist aber nicht die Kommunikation zwischen Prozeß und *pvmd* über die UNIX-Domain-Sockets, da der Einbruch sonst auch die Fore-API-Kurve betreffen müßte. Dieses Phänomen wurde unter HP-UX nicht beobachtet.

Die erreichten Werte, die gegenüber dem Ethernet nur eine Leistungssteigerung um den Faktor drei darstellen, bleiben hinter den Erwartungen zurück. Die Ursache hierfür liegt am Kommunikationspfad innerhalb von PVM, wenn der normale Routing-Modus eingesetzt wird.

[11] $1\ MByte := 10^6 Bytes$

[12] Dort: $1\ MByte := 2^{20} Bytes$

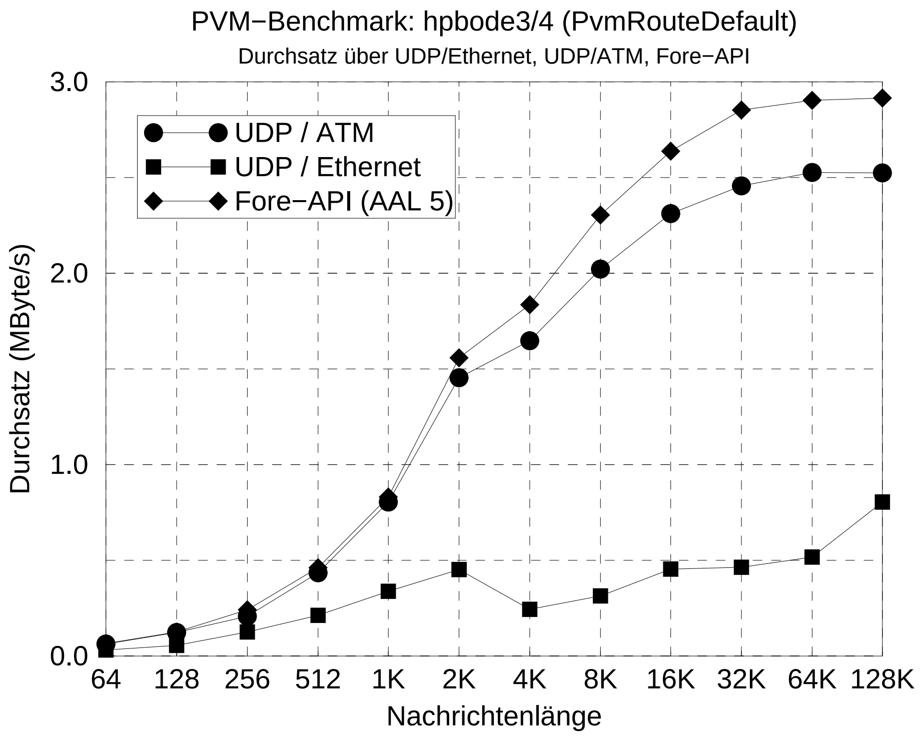

Abbildung 5.16: PVM-Benchmark (*PvmRouteDefault*): Durchsatzwerte für hpbode3/4

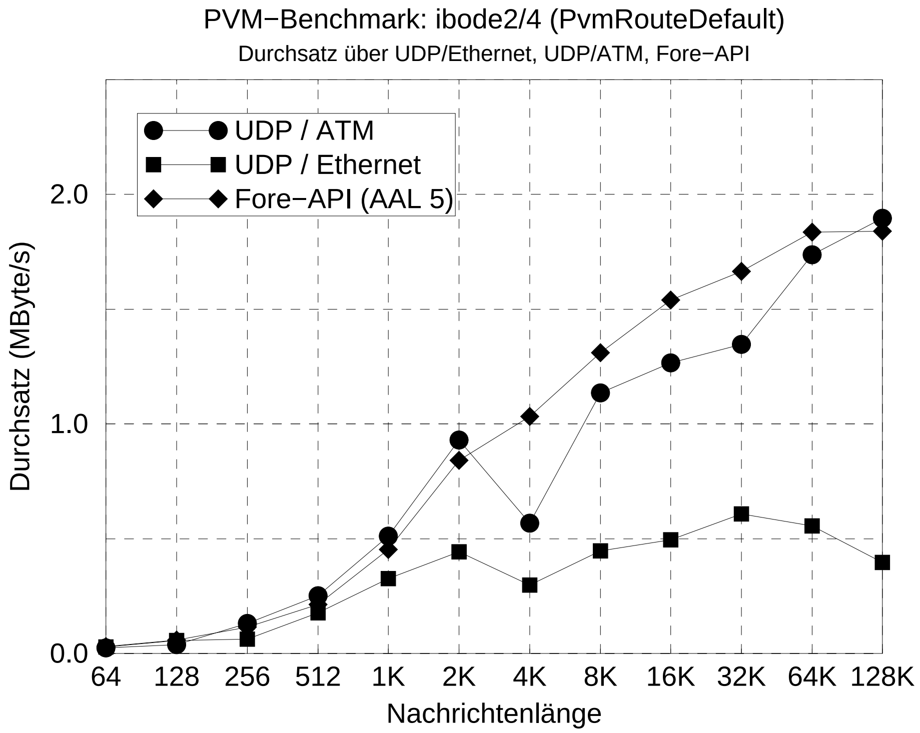

Abbildung 5.17: PVM-Benchmark (*PvmRouteDefault*): Durchsatzwerte für ibode2/4

Der Aufwand, der durch den Protokollwechsel bei der Hintereinanderschaltung von Prozeß-Dämon-Kommunikation (TCP, UNIX-Domain-Sockets) und Dämon-Dämon-Kommunikation (UDP) erzeugt wird, schluckt teilweise den Leistungsgewinn durch ATM. Wie die Tests 16 und 17 zeigen, wird ein wesentlich besserer Durchsatz erreicht, wenn das direkte Routing (*PvmRouteDirect*) benutzt wird. Leider war hier ein Vergleich *IP over ATM* und Fore-API nicht möglich, da mit der verwendeten PVM-ATM-Version der Austausch von Nachrichten über das Fore-API bei aktiviertem Direkt-Routing fehlschlug (vgl. Abschnitt 4.3). Die Diagramme für Test 16 und 17 sind in den Abbildungen 5.18 und 5.19 dargestellt.

Test 16 (hpbode3/4, PvmRouteDirect): Siehe Abbildung 5.18. Wiederum ist ab Nachrichtenlängen von 1 KByte über ATM eine deutliche Leistungssteigerung gegenüber dem Ethernet zu beobachten. Bei 128 KByte wird der Maximaldurchsatz von 6.2 MByte/s gegenüber nur 2.9 MByte/s bei *PvmRouteDefault* erreicht.

Test 17 (ibode2/4, PvmRouteDirect): Siehe Abbildung 5.19. Im Gegensatz zu Test 15 erzielen die IBM-Maschinen hier wieder größere Durchsatzwerte als die HP-Maschinen. Dies deckt sich mit den Ergebnissen der Messungen mit nttcp und netperf. Bei einer Nachrichtenlänge von 128 KByte beträgt der Durchsatz 7.4 MByte/s. Am LACE-Cluster des *Lewis Research Center – LERC* wurden Spitzenwerte von 9.2 MByte/s zwischen zwei IBM-Workstation (RS6000, Model 590s) gemessen[13]. Tabelle 5.3 enthält die Meßwerte für ATM von Test 16 und 17 sowie die des *LERC* für den *LACE-Cluster*.

Länge	64 B	128 B	256 B	512 B	1K	2K	4K	8K	16K	32K	64K	128K
hpbode3/4	0.11	0.23	0.39	0.79	1.36	2.63	3.73	4.84	5.75	5.87	6.06	6.18
ibode2/4	0.06	0.12	0.22	0.43	0.88	1.60	2.75	4.29	5.56	6.24	7.00	7.44
LACE	0.08	0.16	0.30	0.57	1.16	2.04	3.27	4.91	6.43	7.80	8.73	9.19

Tabelle 5.3: PVM-Benchmark: Durchsatzwerte (MByte/s) für hpbode3/4, ibode2/4 und LACE

Unter netperf betrug der maximale Durchsatz einer TCP-Stream-Verbindung bei einer Socket-Puffergröße von 32 KByte für hpbode3/4 9.5 MByte/s, innerhalb von PVM werden hier 6.2 MByte/s erreicht. Die Rechner ibode2/4 erzielen unter netperf einen maximalen Durchsatz von 12.4 MByte/s, innerhalb von PVM 7.4 MByte/s. Der zusätzliche Overhead von PVM kostet bei HP 35%, bei IBM sogar 40% der Durchsatzleistung. Der theoretisch höchstmögliche Durchsatz für *IP over ATM* liegt bei etwa 16 MByte/s. Die Tests zeigen, daß hiervon für die Nachrichtenübertragung zwischen zwei PVM-Prozessen nicht einmal die Hälfte übrig bleibt. Die Ergebnisse der Latenzzeitmessungen mit dem PVM-Benchmark enthält der folgende Abschnitt.

5.4.3 Test 18 – 21: Latenzzeitmessungen mit dem PVM-Benchmark

Wie bei den Durchsatzmessungen wurde auch die Messung der Nachrichtenlatenzzeiten für die beiden Routing-Modi *PvmRouteDefault* und *PvmRouteDirect* durchgeführt. Zusätzlich wird die Messung der sog. *startup latency* und die Messung von Latenzzeiten von großen Nachrichten in getrennten Tests behandelt. In den Tests für die *startup latency* werden Nachrichten mit minimalen Längen von 0, 1, 4 und 16 Bytes betrachtet. Unter großen Nachrichten werden Längen von 128 Bytes, 1 KByte, 4 KByte und 16 KByte verstanden.

[13] `http://www.lerc.nasa.gov/WWW/ACCL/PARALLEL/benchmark.html`

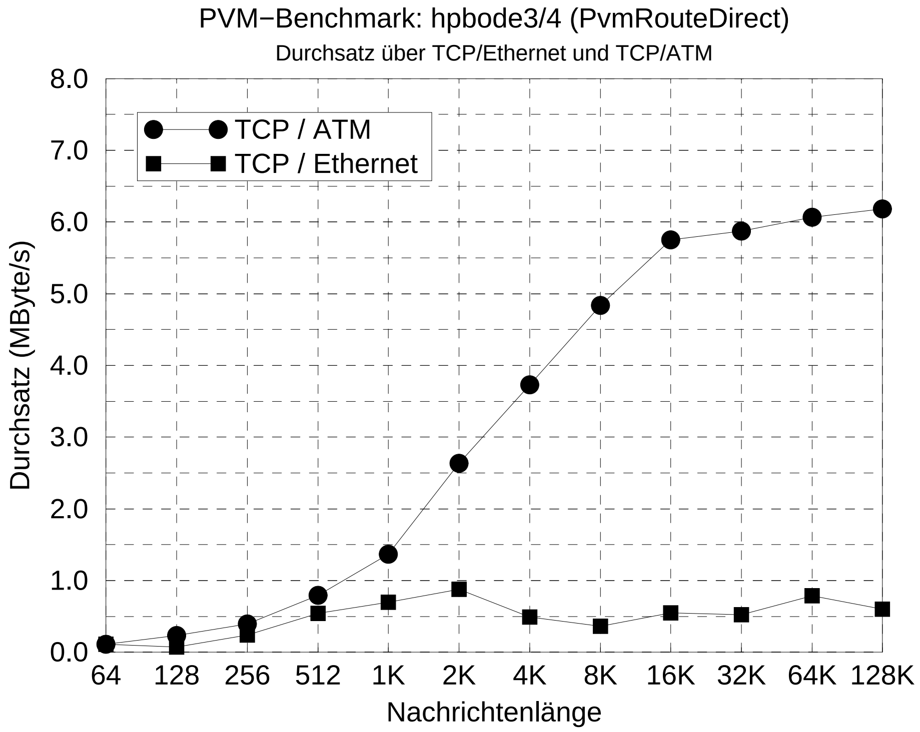

Abbildung 5.18: PVM-Benchmark (*PvmRouteDirect*): Durchsatzwerte für hpbode3/4

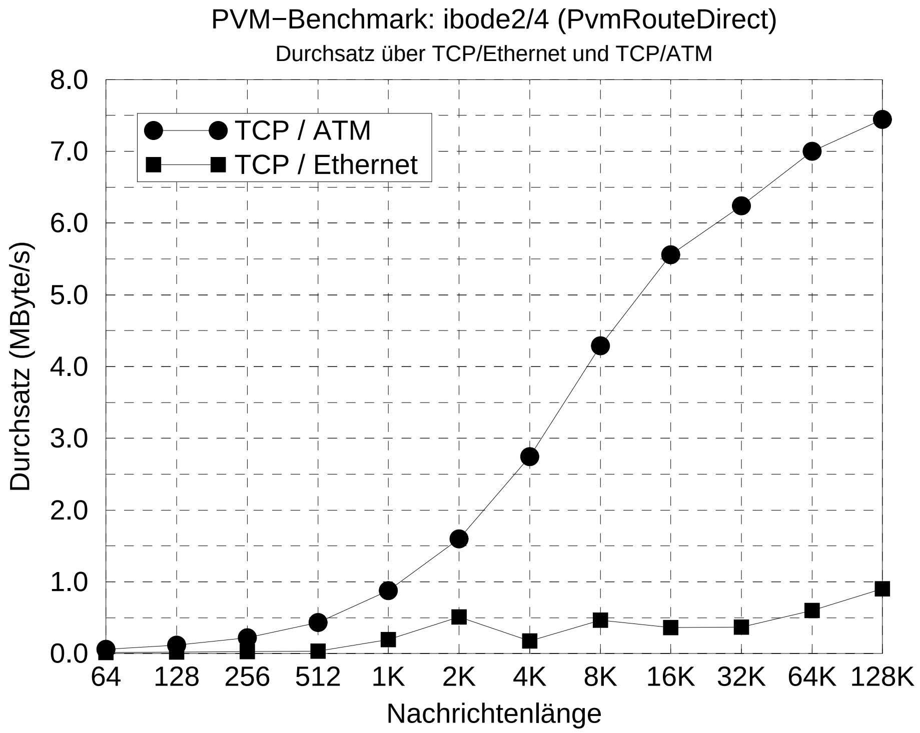

Abbildung 5.19: PVM-Benchmark (*PvmRouteDirect*): Durchsatzwerte für ibode2/4

Test 18 (PvmRouteDefault, startup latency): Siehe Abbildung 5.20. In diesem Test werden die Latenzzeiten, die zwischen hpbode3/4 und ibode2/4 über das Fore-API, bzw. über UDP/ATM und UDP/Ethernet gemessen wurden, miteinander verglichen. Einheitlich ist das Bild bei HP: Für alle vier Nachrichtenlängen liegen die über ATM (Fore-API und UDP) gemessenen Zeiten bei etwa 1 ms. Das Fore-API erzielt bei einer 0-Byte-Nachricht den Bestwert 914 μs. Die Übertragung über ATM ist somit gegenüber UDP/Ethernet etwa um den Faktor drei schneller. Bei IBM muß weiter differenziert werden. Fore-API und UDP/Ethernet liegen hier meist gleichauf bei etwa 2 ms. Das Ethernet hat hier insgesamt sogar die niedrigsten Werte. UDP/ATM zeigt mit Ausnahme der 1-Byte-Nachricht etwas schlechtere Werte und leistet sich bei 16 Bytes mit fast 3.5 ms einen bösen Ausrutscher. Auffällig ist auch, daß die Latenzzeiten von ibode2/4 über ATM mindestens doppelt so groß sind als die von hpbode3/4. Damit hat sich der bereits unter netperf aufgetretene Effekt (vgl. Tabelle 5.1 und 5.2) noch etwas verstärkt.

Test 19 (PvmRouteDefault, große Nachrichten): Siehe Abbildung 5.21. Erwartungsgemäß weist das Ethernet bei Nachrichten größer als 1 KByte sehr viel höhere Zeiten auf als ATM. Dies hängt mit der geringeren Bandbreite und der MTU von 1500 Bytes zusammen. Bei HP zeigen Fore-API und UDP/ATM wieder einheitlich gute Werte. Die IBM-Zeiten sind auch bei größeren Nachrichten sehr viel schlechter wenn auch nicht doppelt so hoch wie bei HP. Das Fore-API liefert hier meist bessere Werte als UDP/ATM. Allgemein ist zu beobachten, daß Nachrichten bis zur Länge von 1 KByte für einen bestimmten Übertragungsweg in etwa die gleiche Latenzzeit haben.

Noch größere Bedeutung haben die Latenzzeiten, wenn der Overhead durch die Protokollwechsel TCP – UDP – TCP von *PvmRouteDefault* beim direkten Routing *PvmRouteDirect* wegfällt. In Test 20 und Test 21 wurden die Latenzzeiten für diesen Modus gemessen.

Test 20 (PvmRouteDirect, startup latency): Siehe Abbildung 5.22. Beim direkten Routing ergeben sich für die *startup latency* annähernd identische Werte für ATM und Ethernet. HP verzeichnet Latenzzeiten um 500 μs, IBM um 1000 μs. Bei der unrealistischen Nachrichtenlänge von 0 Byte wird für das Ethernet zwischen ibode2/4 allerdings eine Latenzzeit von 4.5 ms gemessen.

Test 21 (PvmRouteDirect, große Nachrichten): Siehe Abbildung 5.23. Bei großen Nachrichten wird das Ethernet wieder weit abgeschlagen. Mit zunehmender Nachrichtenlänge werden auch die Latenzzeiten von ibode2/4 im Vergleich zu hpbode3/4 besser. Dies liegt an dem höheren Durchsatz, der zwischen ibode2/4 über TCP/ATM erreicht wird. Die genauen Meßwerte für Test 20 und 21 und wiederum für den *LACE-Cluster* finden sich in Tabelle 5.4.

Nachrichtenlänge	0 B	1 B	4 B	16 B	128 B	1K	4K	16K
hpbode3/4 (ATM)	0.466	0.530	0.530	0.533	0.545	0.750	1.098	2.850
hpbode3/4 (Eth.)	0.437	0.548	0.500	0.577	1.742	1.463	8.379	29.971
ibode2/4 (ATM)	0.858	1.045	1.058	1.087	1.102	1.164	1.491	2.949
ibode2/4 (Eth.)	4.522	1.036	1.010	1.124	6.840	5.342	23.173	45.112
LACE (ATM)	0.677	0.791	0.790	0.790	0.825	0.881	1.251	2.547

Tabelle 5.4: PVM-Benchmark: Latenzzeiten (ms) für hpbode3/4, ibode2/4 und LACE

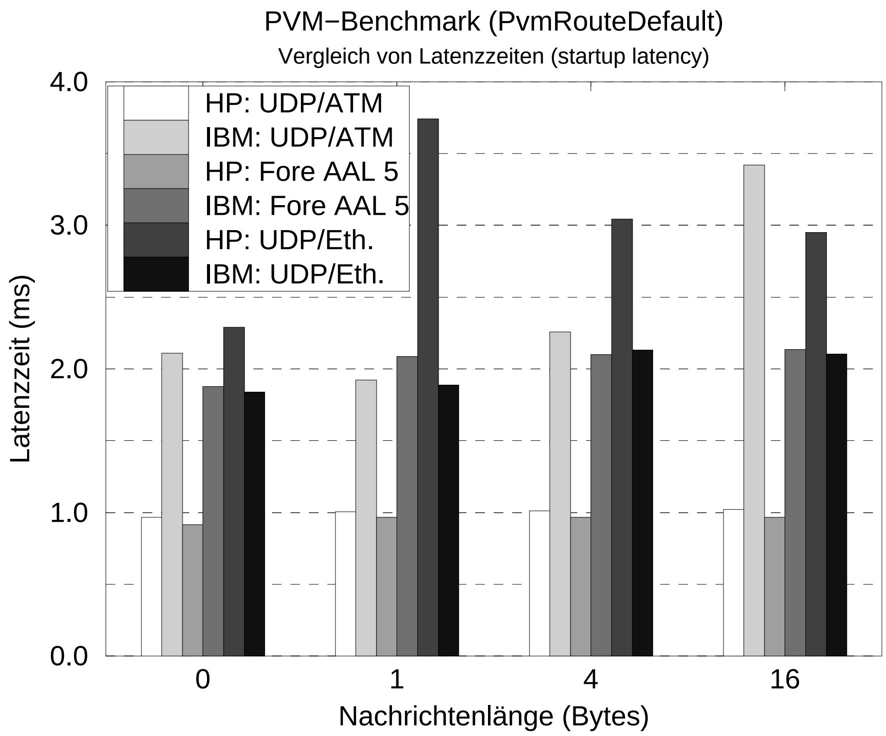

Abbildung 5.20: PVM-Benchmark (*PvmRouteDefault*): Latenzzeiten (*startup latency*)

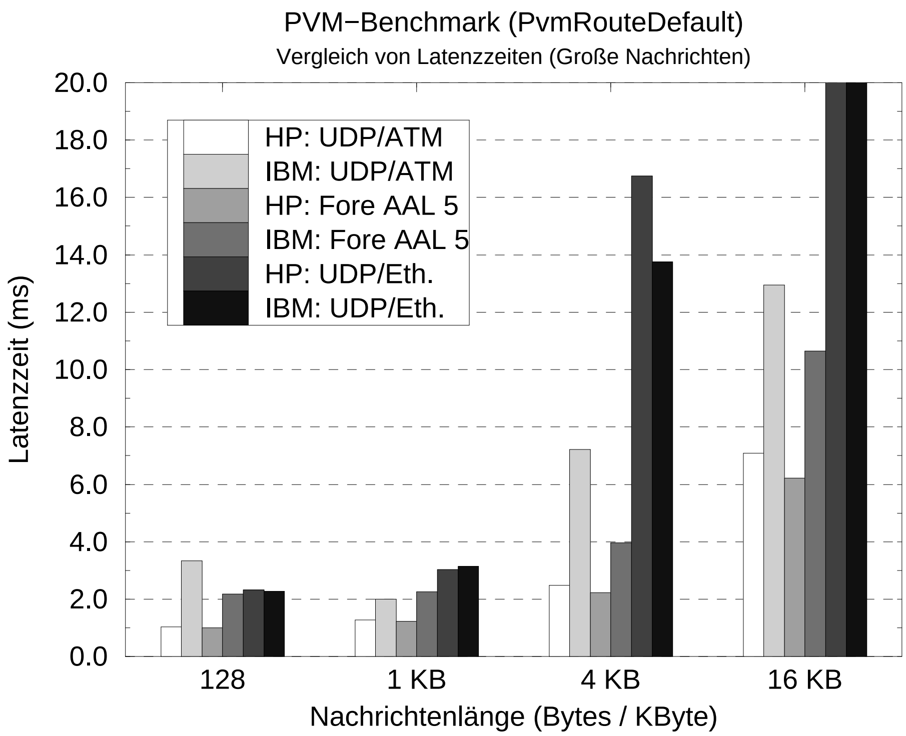

Abbildung 5.21: PVM-Benchmark (*PvmRouteDefault*): Latenzzeiten für große Nachrichten

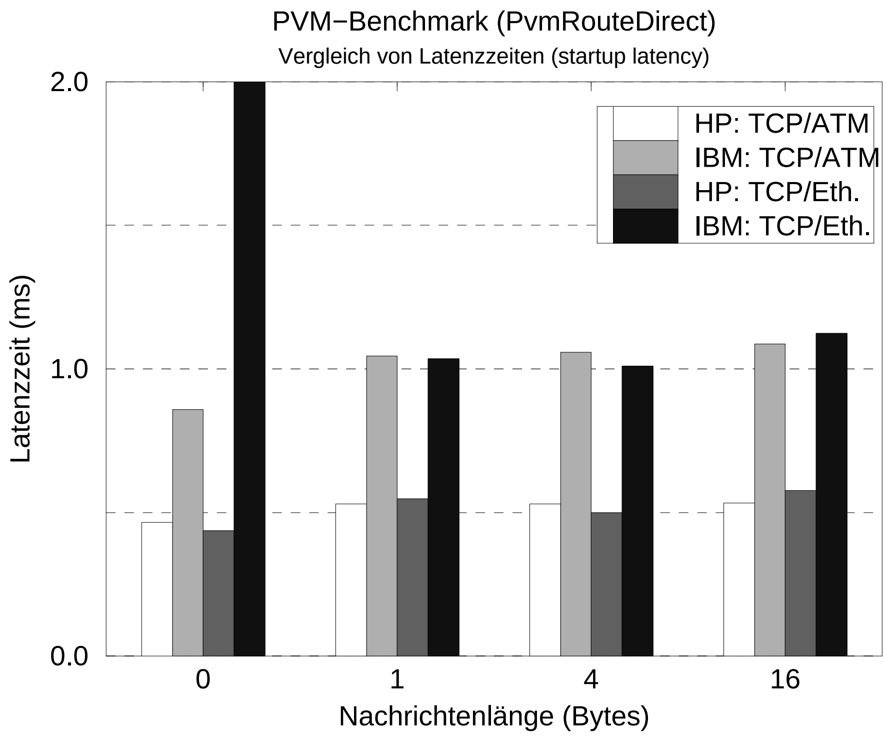

Abbildung 5.22: PVM-Benchmark (*PvmRouteDirect*): Latenzzeiten (*startup latency*)

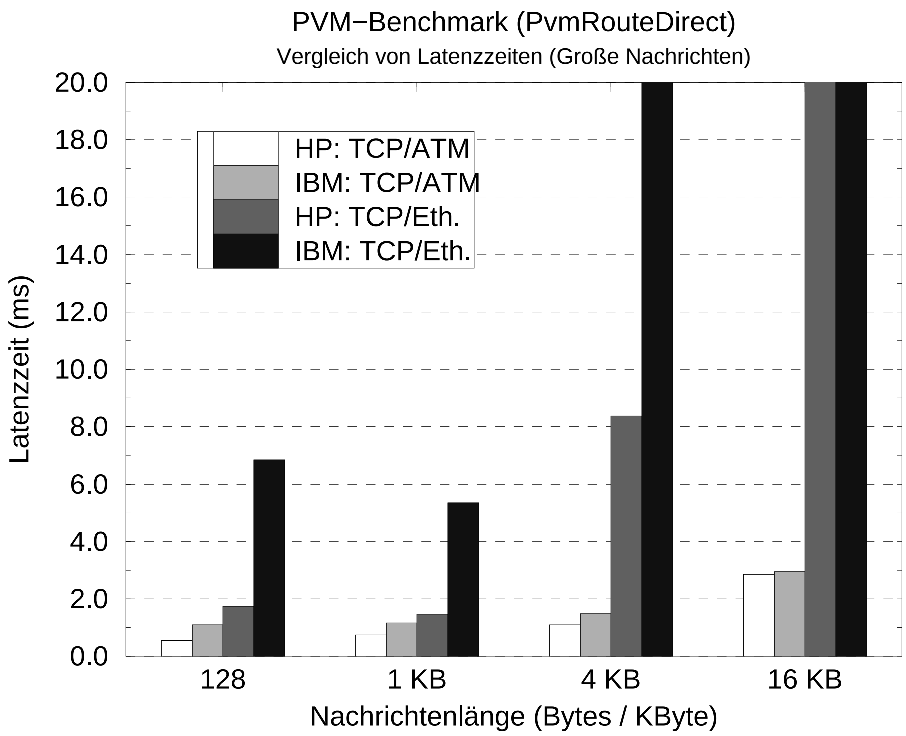

Abbildung 5.23: PVM-Benchmark (*PvmRouteDirect*): Latenzzeiten für große Nachrichten

Vergleicht man die Werte für die *startup latency* innerhalb PVM (*PvmRouteDirect)* mit denen von netperf (Tabellen 5.1, 5.2), so sind erstere etwa doppelt so hoch. Die niedrigsten Latenzzeiten wurden unter netperf allerdings mit dem Fore-API erzielt. Auch unter *PvmRouteDefault* hat das Fore-API eine gute Leistung gezeigt. Die Unterstützung des Fore-API von PVM macht daher durchaus Sinn, da hiermit die limitierte Skalierbarkeit aufgrund der beschränkten Anzahl von File-Deskriptoren für TCP aufgehoben wäre. Da das Fore-API proprietär ist, kann es natürlich nur in einem Fore-ATM-Cluster eingesetzt werden.

6

Optimierungsmöglichkeiten für TCP/UDP über ATM

Die TCP/IP-Protokoll-Suite wurde in den siebziger Jahren entwickelt. Damals waren Netztechnologien wie Token-Ring mit 4 MBit/s und Ethernet mit 10 MBit/s Stand der Technik. Es hat sich herausgestellt, daß TCP/IP aufgrund des Overheads der Protokollschichtung nicht optimal für Hochgeschwindigkeitsnetze wie ATM mit sehr hohen Bandbreiten geeignet ist.

TCP nutzt nicht die Möglichkeiten von ATM bei den *Quality of Service* Parametern wie z.B. Reservierung von Bandbreite für bestimmte Verbindungen. Andererseits enthält TCP als Transportschicht Funktionen, die für das Weiterleiten von Paketen in traditionellen routerbasierten Netzen (*Store-and-Forward*-Architektur) erforderlich sind. Da Nachrichten in einem ATM-Netz auf Zellebene über eine logische Ende-zu-Ende-Verbindung von den ATM-Switches transportiert werden, sind Funktionen wie Reihenfolgesicherung überflüssig, da Zellen immer über den gleichen Pfad laufen und somit automatisch in der richtigen Reihenfolge beim Empfänger eintreffen.

Auch die Fehlerkorrektur von TCP, die im Fehlerfall ein gesamtes TCP-Segment erneut überträgt, ist für ATM ungeeignet. Hier könnte eine Transportschicht Fehler aufgrund von verlorenen Zellen direkt erkennen und nur diese erneut beim Sender anfordern.

Aus diesen Gründen entwickeln Hersteller von ATM-Netzkomponenten Kommunikationsschnittstellen wie das Fore-API, damit Anwendungen unter Umgehung des Protokoll-Overhead von TCP/IP effizient über ATM-Netze kommunizieren können. Tatsache ist aber, daß sich TCP/IP weltweit für die Datenkommunikation über lokale Netze und Weitverkehrsnetze und in heterogenen Systemumgebungen durchgesetzt hat. Auch parallele Programmierumgebungen und Bibliotheken für Nachrichtenaustausch (*message passing*) wie PVM oder MPI benutzen IP als Kommunikationsmechanismus, da der Austausch von Nachrichten zwischen völlig unterschiedlichen Systemarchitekturen möglich sein soll. Trotz Leistungseinbußen werden daher selten effizientere bzw. proprietäre Schnittstellen genutzt.

An Standards für *IP over ATM* arbeiten mehrere Organisationen wie z.B. das ATM-Forum. Aufgabe der Komponentenhersteller ist die möglichst leistungsfähige und effiziente Implementierung von Treibern für *IP over ATM*. Daneben gibt es aber Möglichkeiten als Anwender bzw. Systemadministrator über bestimmte Parameter TCP/IP für ATM zu optimieren. Einige davon sind bereits bei den Leistungsmessungen erwähnt worden. Eine Zusammenfassung

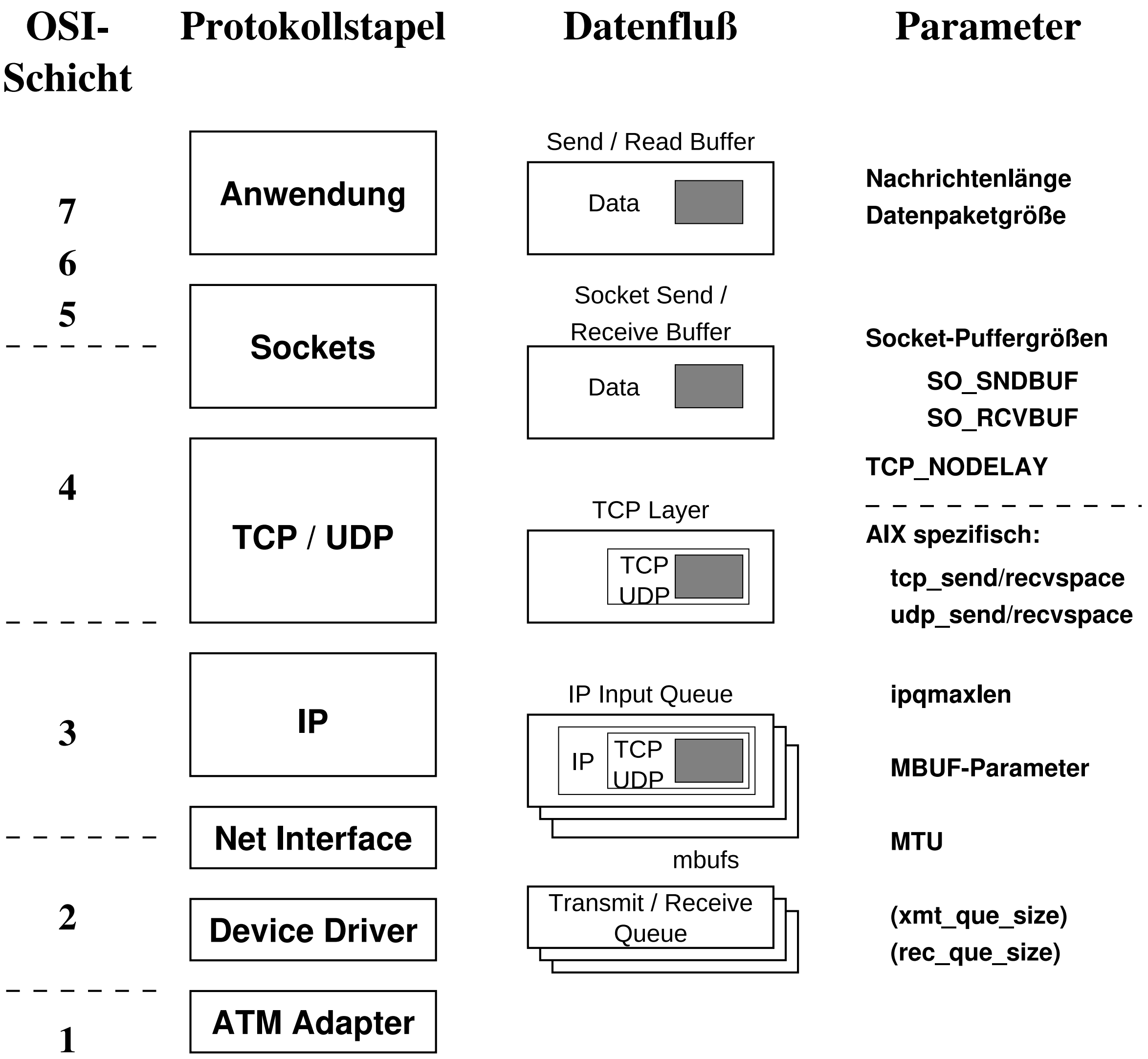

Abbildung 6.1: Übersicht über den Datenfluß bei IP über ATM

und einen Ausblick auf die dynamische Pufferverwaltung unter AIX beinhalten die folgenden Abschnitte. Vorher wird der Weg eines Datenpakets durch den Protokollstapel beschrieben.

6.1 Paketdatenfluß

Die Abbildung 6.1 stellt eine grobe Übersicht über den Fluß eines Datenpakets durch den Protokollstapel von der Anwendung bis zur ATM-Adapterkarte dar. Zusätzlich enthält sie die Einordnung in das OSI-Schichtenmodell und einen Überblick über Parameter, die auf den einzelnen Schichten eingestellt werden können. Das Senden und Empfangen von Daten wird getrennt betrachtet.

Senden: Eine Anwendung, die Datenkommunikation über TCP/IP betreiben will, nutzt dazu die BSD-Socket-Schnittstelle[1]. Beim Öffnen eines Sockets wird festgelegt, ob TCP oder UDP als Transportprotokoll verwendet werden soll. Schreibt eine Anwendung auf einen Socket, werden Daten aus dem Sendepuffer im Benutzerspeicher (*user space*) in den Socket-Sendepuffer im Kernel-Speicherbereich kopiert. Die Socket-Puffer werden im Kernel durch Datenstrukturen sog. *mbufs*, die fest im Hauptspeicher gehalten werden, realisiert. Einzelne *mbufs* besitzen eine Größe von 256 Bytes, *mbuf clusters* sind 4096 Bytes groß. Abhängig von der zu übertragenden Datenmenge wird der TCP/UDP-Schicht ein Zeiger auf eine verkettete Liste von *mbufs* oder Clusters übergeben. Die TCP/UDP-Schicht hängt an die Liste einen weiteren *mbuf* mit dem TCP/UDP-Header an. Ein Zeiger auf diese Liste wird wiederum an die IP-Schicht weitergereicht. Die IP-Schicht segmentiert die Daten. Die maximale Segmentgröße für TCP (*Maximum Segment Size – MSS*) ist die MTU der Netzschnittstelle abzüglich der Größe des TCP- und IP-Headers (jeweils 20 Bytes). Bei ATM beträgt die MSS somit $9188 - 40 = 9148$ Bytes. Anschließend wird das Segment in ein IP-Paket mit einem IP-Header verpackt. Das Paket wird an das entsprechende Netz-Interface weitergegeben. Das Netz-Interface überprüft, ob das Paket ein für den Device-Treiber gültiges Format besitzt und schreibt das Paket an den Device-Treiber. Dort wird es in die Sendewarteschlange (*transmit queue*) gestellt und vom Adapter mittels DMA abgeholt. Der Adapter zerlegt die Pakete in ATM-Zellen und sendet diese an das ATM-Netz.

Empfangen: Den umgekehrten Weg nehmen die vom Adapter empfangen Daten. Der Adapter setzt die Zellinhalte wieder zu Paketen zusammen und übergibt sie der Empfangswarteschlange des Device-Treibers. Der Treiber verpackt die Daten in einer *mbuf*- oder Clusterkette. Handelt es sich um IP-Pakete werden diese von der Interface-Schicht in die Warteschlange der IP-Schicht (*IP input queue*) gestellt. Die durch einen Interrupt aufgerufene IP-Schicht entfernt den IP-Header und übergibt das Paket der TCP- oder UDP-Instanz. In der TCP/UDP-Schicht wird das Paket in den zugehörigen Socket-Empfangspuffer geschrieben, wo die Daten von der Anwendung in den Benutzerspeicher (*application read buffer*) gelesen werden können. Die *mbufs* werden in Abschnitt 6.3 genauer besprochen.

Ein Tuning von Optionen und Parametern der verschiedenen Schichten kann die Durchsatzleistung von *IP over ATM* verbessern. Im folgenden werden die Parameter der einzelnen Schichten diskutiert.

6.2 Optimierungsparameter der Protokollschichten

Anwendungsschicht

Auf Anwendungsebene bestimmt die Nachrichtenlänge die Größe des Sende- und Empfangspuffers im Benutzerspeicherbereich. Wie die Tests gezeigt haben, wird der höchste Durchsatz meist bei großen Nachrichten erreicht, da dann der Overhead des TCP/IP-Protokollstapels bezogen auf eine Nachricht klein ist. Natürlich kann die Nachrichtenlänge die Größe des Socket-Puffers nicht überschreiten.

[1] Die BSD-Socket-Schnittstelle wird in [22] detailliert beschrieben.

Unter AIX sollte die Nachrichtenlänge ein Vielfaches von 4 KByte betragen, da dies die Größe eines *mbuf-Cluster* ist. Kleinere Nachrichten als 4 KByte senken den Durchsatz unter TCP. Bei Verwendung von UDP ist es sinnvoll, daß die Nachrichtenlänge die MTU des ATM-Interface nicht überschreitet. Die Nachricht kann dann ohne Segmentierung als einzelnes UDP-Paket dem Device-Treiber übergeben werden.

Socket- und TCP/UDP-Schicht

Bei kleinen Nachrichten, die über TCP-Stream-Sockets versendet werden sollen, empfiehlt sich das Setzen der Socket-Option `TCP_NODELAY` mittels *setsockopt()*, damit die TCP-Schicht nicht das Senden der kleinen Segmente verzögert. Hiermit werden auch bekannte Timing-Anomalien von TCP verhindert (vgl. [17], Abschnitt 3.2).

Die Messungen in dieser Arbeit haben gezeigt, daß die Socket-Puffergrößen (`SO_SNDBUF` und `SO_RCVBUF`) erheblichen Einfluß auf den erreichbaren Durchsatz haben. Die Default-Werte 8 KByte bei HP-UX und 16 KByte bei AIX sind für ATM-Netze zu klein. In [18] wird eine minimale Größe von 45 KByte (allerdings für SunOS) empfohlen. Unter HP-UX und AIX wird der höchste Durchsatz jeweils bei den maximalen Puffergrößen von 56 KByte respektive 64 KByte erzielt. Bei PVM beträgt der Standardwert für TCP 32 KByte.

Es gibt weitere Regeln für die Festlegung der Puffergrößen:

- Die Größe der Socket-Puffer soll ein Vielfaches der Nachrichtenlänge betragen.
- Die Puffergröße legt auch die Größe des TCP-Fensters für die Flußkontrolle gemäß RFC 1122 fest. Damit dieses Verfahren in der Praxis funktioniert, *muß* der Sendepuffer mindestens die doppelte Größe des maximalen TCP-Segments (MSS) haben (vgl. [18]).
- In [4] ist eine Anomalie beschrieben, bei der der Durchsatz einer TCP-Verbindung über ATM zusammenbricht, wenn der Empfangspuffer in bestimmten Bereichen größer als der Sendepuffer ist. Dies hängt ebenfalls mit den Algorithmen für die Flußkontrolle von TCP und zur Vermeidung des *Silly Window Syndrome* (*Nagle's algorithm*) zusammen. Diese Anomalie wird verhindert, wenn der Socket-Sendepuffer mindestens dreimal so groß wie die MSS ist. In diesem Fall ist die Größe des Socket-Empfangspuffers des Endsystems am anderen Ende der TCP-Verbindung unerheblich.

Unter HP-UX ist der Standardwert für die Socket-Puffergrößen als Konstante im Kernel festgelegt. AIX hingegen läßt eine dynamische Konfiguration der Kernel-Variablen durch das Kommando `no` zu. Durch folgenden Befehl wird die Standardgröße des TCP-Socket-Sendepuffers auf 64 KByte gesetzt:

```
no -o tcp_sendspace=65536
```

Analog für `tcp_recvspace`, `udp_sendspace` und `udp_recvspace`. Wird die Option `rfc1323=1` mit `no` gesetzt, sind auch größere Socket-Puffer als 64 KByte möglich. Zu beachten ist, daß die Default-Werte für alle Netz-Schnittstellen eines Systems gelten.

Die dynamische Konfiguration von Parametern unterhalb der TCP/UDP-Schicht ist ausschließlich unter AIX möglich. Unter HP-UX Version 10 werden diese Parameter automatisch an die Leistungsfähigkeit der Netz-Interfaces eines Systems angepaßt.

IP-Schicht

Auf der IP-Schicht kann als einziger Parameter die Länge der Warteschlange für empfangene IP-Pakete (*IP input queue*) konfiguriert werden. Der Standardwert für `ipqmaxlen` ist 50. Wenn diese Grenze erreicht wird, kann die IP-Schicht keine weiteren Pakete vom Netz-Interface akzeptieren und verwirft die Pakete. Dies ist möglicherweise der Grund für die hohen Paketverluste bei den UDP-Durchsatzmessungen (vgl. Test 3). Wird der Wert von `ipqmaxlen` vergrößert, kann starker UDP-Verkehr über den ATM-Adapter empfangen werden. Dadurch steigt allerdings der Bedarf an Rechenzeit der IP-Schicht, da mehr Pakete in der Warteschlange von IP bearbeiten werden müssen, was negative Auswirkungen auf andere Prozesse im System hat. Außerdem ist nicht garantiert, daß die IP-Schicht sehr hohe IP-Verkehrsraten schnell genug verarbeiten kann. Die Warteschlange sollte daher nur vergrößert werden, wenn es tatsächlich zum Überlauf[2] kommt und wenn das System genug CPU-Reserven besitzt. Folgendes Kommando verdoppelt die Länge der Warteschlange:

```
no -o ipqmaxlen=100
```

Interface-Schicht

Auf der Interface-Schicht ist die maximale Transporteinheit MTU einstellbar. Der Wert 9188 Bytes für *Classical IP over ATM* wird im RFC 1577 [16] definiert und von den Fore-Gerätetreibern als Default benutzt. Es ist nicht sinnvoll, diesen Wert zu verändern. Kleinere Werte würden den möglichen Durchsatz verringern. Die MTU muß für alle Endsysteme, die direkt miteinander verbunden sind, gleich groß sein.

Gerätetreiber-Schicht (Device Driver)

Auf dieser Schicht ist es bei einigen Netzadaptern möglich, die Größe der Warteschlangen für das Senden und Empfangen von Übertragungsrahmen (`xmt_que_size` und `rec_que_size`) zu konfigurieren. Die ATM-Gerätetreiber von Fore lassen dies allerdings nicht zu.

6.3 AIX: Verwaltung des mbuf-Pools

Die Datenstruktur für Puffer, die AIX im Kernel benutzt, um Daten von einer Anwendung zum Device-Treiber einer Netzschnittstelle und umgekehrt zu transportieren, wird *mbuf* genannt. Das System verwaltet einen Pool dieser Puffer, der von allen Kommunikationsprotokollen (z.B. TCP/IP und SNA) genutzt wird. Der Pool wird im Hauptspeicher (RAM) des Rechners fest angelegt, d.h. Teile des Pools werden niemals in den Sekundärspeicher ausgelagert. Im mbuf-Pool werden Socket-Puffer, IP-Pakete, Header, Routing-Informationen und andere Daten des Kommunikationssubsystems gespeichert. Die Größe des Pools bzw. die Anzahl der verfügbaren Puffer hat Einfluß auf die Leistungsfähigkeit einer Netzschnittstelle. Im folgenden werden Parameter besprochen mit denen die Pufferverwaltung und somit die Leistungsfähigkeit optimiert werden kann. Die Beschreibung ist nur für die AIX-Version 3.2 gültig. Obwohl das Prinzip der Pufferverwaltung in AIX 4 gleichgeblieben ist, verfügt diese Version über automatische Mechanismen zur Optimierung.

In Abbildung 6.2 ist das Schema des mbuf-Pools abgebildet. Zur Verfügung stehen zwei verschiedene Puffergrößen. Ein kleiner Puffer (*mbuf*) ist 256 Bytes groß, ein *mbuf-Cluster*

[2] Die Vorgehensweise zum Auslesen des *overflow counter* findet sich in [14] und [20].

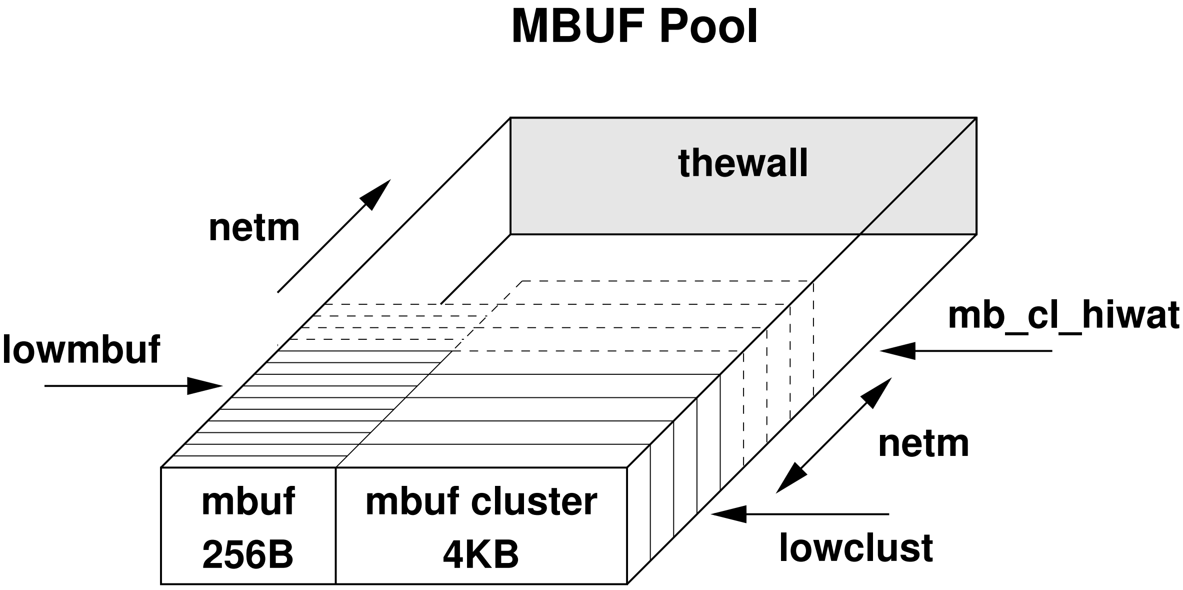

Abbildung 6.2: Parameter für die Verwaltung des mbuf-Pools

faßt 4 KByte. Tatsächlich setzen sich die Cluster aus einer Gruppe von *mbufs* zusammen. Kleine Datenpakete und Header werden in kleinen Puffern, große Datenpakete in Clustern abgelegt. Sowohl *mbufs* wie auch Cluster können zu Listen verkettet werden. Cluster enthalten ausschließlich Daten und werden stets mit einem *mbuf* verkettet, der beispielsweise den Paket-Header beinhaltet.

Um die Verwaltung des Pools kümmert sich der Kernel-Prozeß *netm*. Seine Aufgabe ist es, dafür zu sorgen, daß stets eine Mindestanzahl freie *mbufs* und Cluster vorhanden sind. Das bedeutet, daß sich die Größe des Pools und damit der benötigte Hauptspeicherplatz dynamisch mit der Netzlast verändert. Die Default-Werte für die Minimalanzahl freier *mbufs* und Cluster sind für eine mittlere Netzlast von 100 – 500 Paketen pro Sekunde ausgelegt. Zur Optimierung der Leistungsfähigkeit einer Netzschnittstelle muß der Administrator die Anzahl der verfügbaren Puffer an die durchschnittliche Netzlast anpassen. Die maximale Größe des Pools und einige Schwellwerte können mit dem Kommando `no` zur Laufzeit des Systems verändert werden.

Die Parameter `lowmbuf` und `lowclust` bestimmen die Mindestanzahl freier *mbufs* bzw. Cluster. Sinkt die Anzahl freier Puffer unter eine dieser Grenzen, fordert *netm* Speicherplatz für weitere Puffer von der Hauptspeicherverwaltung des Systems an. Beim nächsten Lauf von *netm* werden die zusätzlichen Puffer zur Verfügung gestellt. Die Gesamtgröße des Pools darf die Grenze, die durch den Parameter `thewall` festgelegt ist, nicht überschreiten. Hat die Größe des Pools `thewall` erreicht, können keine weiteren Puffer mehr angelegt werden und es kommt zu Paketverlusten. Der Default-Wert für `thewall` beträgt 2 MByte. Die Default-Werte für `lowmbuf` und `lowclust` hängen von der Systemkonfiguration (z.B. Anzahl der Netzadapter im Rechner) ab. Übersteigt die Anzahl der freien Cluster im Pool den Wert `mb_cl_hiwat`, löst *netm* einige Cluster auf und gibt den Speicher an die Speicherverwaltung zurück. Die Anzahl der freien Cluster schwankt also zwischen `lowclust` und `mb_cl_hiwat`.

Veränderungen an den Parametern müssen sorgfältig geplant werden, da falsch eingestellte

Werte die Leistung des ganzen Systems erheblich senken kann. Wird der Wert für `thewall` zu hoch gewählt, kann das Auslagern von Speicherseiten zunehmen, da anderen Prozessen weniger Hauptspeicher im RAM zur Verfügung steht. Ein zu kleiner Wert kann andererseits zu Paketverlusten aufgrund fehlender Puffer führen. Falsch eingestellte Schwellwerte für *mbufs* und Cluster erfordern häufiges Verändern der Pufferanzahl durch *netm*. Da *netm* eine konstant hohe Priorität besitzt, kann dadurch viel CPU-Zeit verbraucht werden (*thrashing*). Umso weniger *netm* anpassen muß, umso effizienter ist die Pufferverwaltung.

Ein Tuning des mbuf-Pools ist also nur notwendig, wenn über das System eine hohe Netzlast abgewickelt wird bzw. wenn Probleme wie Paketverluste auftreten. Die Last gemessen in IP-Paketen pro Sekunde an einem Netzinterface kann mit dem Kommando

```
netstat -I <device> <intervall>
```

abgefragt werden. Das Kommando

```
netstat -m
```

gibt detaillierte Informationen über die momentane Auslastung des mbuf-Pools aus. Eine Änderung der mbuf-Parameter ist dann erforderlich, wenn Speicheranforderungen verweigert wurden (*requests for memory denied*) bzw. wenn die Anzahl der benutzen Cluster nahe an der Maximalanzahl ist.

Für das Tuning der einzelnen Parameter gelten folgende Anhaltspunkte:

- Fällt die Hauptnetzlast auf UDP soll die Anzahl der *mbufs* mindestens doppelt so groß wie die durchschnittliche Paketrate sein.
- Der Wert für `lowmbuf` soll mindestens doppelt so groß wie der für `lowclust` sein, da mit jedem Cluster mindestens ein *mbuf* verbunden ist.
- Werden die Werte für `lowmbuf` bzw. `lowclust` verändert, muß auch `thewall` entsprechend angepaßt werden.
- Nach Vergrößerung von `thewall` sollte mit dem Kommando `vmstat` überprüft werden, ob vermehrtes Auslagern von Seiten (*paging*) auftritt. Dann muß der Hauptspeicher des Systems eventuell vergrößert werden.
- Der Wert für `mb_cl_hiwat` soll mindestens doppelt so groß wie `lowclust` sein, um *thrashing* von *netm* zu verhindern.

Die AIX-Pufferverwaltung und das Tuning der Parameter sind in [14] und [20] detailliert beschrieben. Anhang D enthält die Werte, die von IBM für die Benutzung des IBM *TurboWays* ATM-Adapters vorgeschlagen werden.

7

Zusammenfassung

Aufgabe dieser Arbeit war es, eine Leistungsanalyse des ATM-Netzes am LRR-TUM durchzuführen. Über einen Fore-ATM-Switch ist dort ein heterogener Cluster von HP- und IBM-Rechnern miteinander vernetzt. Von Hochleistungsnetzen wie ATM wird erwartet, daß sie die Effizienz von parallelen Programmierumgebungen erheblich steigern können. Daher sollte gemessen werden, welche Leistungssteigerung beim Nachrichtenaustausch innerhalb der parallelen Programmierumgebung PVM beim Einsatz von ATM gegenüber Ethernet erreicht wird.

In der Analyse wurden die beiden Leistungsgrößen Durchsatz und Latenzzeit untersucht. Diese Größen wurden für die IP-basierten Protokolle TCP und UDP sowie für die proprietäre Kommunikationsschnittstelle Fore-API ermittelt. Zur Messung wurden die frei verfügbaren Benchmarks nttcp und netperf sowie der PVM-Benchmark benutzt. Das Programm nttcp wurde um die Unterstützung des Fore-API erweitert.

Der Durchsatz über TCP beträgt zwischen den HP-Rechnern höchstens 10.1 MByte/s und zwischen den IBM-Rechnern 15.7 MByte/s. Diese Werte werden aber nur bei den Maximalgrößen für die Socket-Puffer erreicht. Unter UDP wird mit nttcp ein Durchsatz von höchstens 9.5 MByte/s für HP und 16.1 MByte/s für IBM gemessen. Aufgrund der fehlenden Flußkontrolle bei UDP kommt es zu erheblichen Paketverlusten beim Empfänger, wenn die Senderate nicht kontrolliert wird. Über das Fore-API (AAL 5) können Daten zwischen den HP-Maschinen mit einem Durchsatz von 10.9 MByte/s und zwischen den IBM-Maschinen mit 11.9 MByte/s übertragen werden. Die Fore-Treiber für *IP over ATM* sind für AIX besser als für HP-UX. Das Fore-API enttäuscht trotz des geringeren Protokoll-Overheads.

Die Latenzzeiten beim Austausch von kleinen Nachrichten über das ATM-Netz sind bei HP um den Faktor zwei kleiner als bei IBM. Hier zeigt das Fore-API geringfügig bessere Werte als *IP over ATM*. Ein unbelastetes Ethernet kann aber durchaus noch mithalten. Die noch junge ATM-Technik ist bezüglich Latenzzeiten noch nicht so optimiert wie das etablierte Ethernet.

Innerhalb von PVM ist die durch ATM erzielbare Leistungssteigerung vom Modus für das Routing abhängig. Der PVM-Benchmark mißt mit *PvmRouteDefault* nur einen moderaten Durchsatz von 2 -3 MByte/s für IP über ATM bzw. für das Fore-API. Mit dem Direktmodus *PvmRouteDirect* wird ein Durchsatz von 6.2 MByte/s (HP) und 7.4 MByte/s (IBM) für TCP/ATM erreicht. Der Kommunikations-Overhead von PVM senkt den Durchsatz auf etwa die Hälfte des Werts der mit den reinen Benchmarks nttcp und netperf gemessenen Werte.

Die Latenzzeiten kleiner Nachrichten sind mit *PvmRouteDirect* für ATM annähernd gleich groß wie für das Ethernet, zwischen HP-Maschinen allerdings nur halb so groß als zwischen IBM-Maschinen.

TCP/IP ist aufgrund seiner aufwendigen Schichtung und seiner Konzeption für routerbasierte Netze nur eingeschränkt für ATM geeignet. Eine Schnittstelle für Anwendungen zum direkten Aufsatz auf der AAL-Schicht von ATM in Verbindung mit einer Transportschicht, deren Funktionen auf die Zellvermittlungstechnik von ATM zugeschnitten sind, könnte die Übertragungsleistung gegenüber TCP/IP verbessern. Dem Fore-API fehlen Transportschichtfunktionen wie Flußkontrolle und Fehlerkorrektur. Außerdem ist es aufgrund der Proprietät keine echte Alternative zu TCP/IP. Auf diesem Bereich ist aber das ATM-Forum dabei, einen Standard zu entwickeln. Um mit TCP/IP über ATM sehr gute Leistungen zu erzielen, ist es nötig, Parameter auf unterschiedlichen Schichten optimal einzustellen.

A

Aufrufparameter von nttcp

In diesem Anhang sind alle in den Messungen verwendeten Aufrufparameter von `nttcp` zusammengefaßt. Für diese Arbeit wurde `nttcp` um die Unterstützung der Kommunikationsschnittstelle Fore-API (vgl. Kapitel 3) erweitert. Bei analoger Aufrufsyntax kann somit der Durchsatz von Datenverbindungen über das Fore-API mit AAL 3/4 und AAL 5 gemessen werden.

Aufruf Sender (*Client*): `nttcp -t [-options] host [ < in ]`

Aufruf Empfänger (*Server*): `nttcp -r [-options] [ > out ]`

Parameter und Optionen:

-t Konfiguriert `nttcp` als Sender.

-r Konfiguriert `nttcp` als Empfänger.

-n## Anzahl der zu sendenden Nachrichten (*source buffer*)

-l## Länge der Nachrichten (*length of source buffer*) in Bytes. Default: 8192.

-w## Setzt die Größe des Socket-Sende-/Empfangspuffers, Einheit KByte.

-D Setzt `TCP_NODELAY` Socket-Option.

-u Kommunikation über UDP-Datagramme anstatt TCP.

-g## Verzögerungswert zwischen der Aussendung von UDP-Paketen (*inter packet delay*). Kann auch für das Fore-API benutzt werden. Da es bei UDP und beim Fore-API keine Flußkontrolle gibt, kann so die Senderate des Senders gedrosselt werden, falls es zu Paketverlusten kommt.

-a Kommunikation über das Fore-API anstatt über TCP/IP.

-4 Fore-API AAL 3/4.

-5 Fore-API AAL 5 (default).

B

Testkonfigurationen: nttcp

Test 1: TCP: Durchsatz bei variabler Nachrichtenlänge

Ethernet:
Sender hpbode3: `nttcp -t -n<anzahl> -l<laenge> -D -w32 hpbode4`
Empfänger hpbode4: `nttcp -r -l<laenge> -w32`

ATM:
Sender hpbode3: `nttcp -t -n<anzahl> -l<laenge> -D -w32 hpbode4.atm`
Empfänger hpbode4: `nttcp -r -l<laenge> -w32`

Nachrichtenlänge:

- Startwert: 512 Bytes
- Endwert: 32 KByte
- Multiplikator: 2

Die Anzahl n Nachrichten wurde für jeden Meßwert so berechnet, daß das Produkt aus Anzahl n und Länge l 16 MByte ergab. Analog der Aufruf für ibode2/4.

Test 2: TCP: Durchsatz bei variabler Socket-Puffergröße

ATM:
Sender hpbode3: `nttcp -t -n2048 -l8192 -D -w<groesse> hpbode4.atm`
Empfänger hpbode4: `nttcp -r -l8192 -w<groesse>`

Socket-Puffergröße:

- Startwert: 4 KByte
- Endwert: 56 KByte (HP-UX), 64 KByte (AIX)
- Inkrement: 4 KByte

Analog für ibode2/4.

Test 3: UDP: Durchsatz bei variabler Nachrichtenlänge

Ethernet:

Sender hpbode3: nttcp -t -u -n<anzahl> -l<laenge> hpbode4
Empfänger hpbode4: nttcp -r -u -l<laenge>

ATM:

Sender hpbode3: nttcp -t -u -n<anzahl> -l<laenge> -g<delay> hpbode4.atm
Empfänger hpbode4: nttcp -r -u -l<laenge>

Nachrichtenlänge:

- Startwert: 512 Bytes
- Endwert: 8 KByte
- Inkrement: 1 KByte (ab 1 KByte)

Die Anzahl n Nachrichten wurde für jeden Meßwert so berechnet, daß das Produkt aus Anzahl n und Länge l 16 MByte ergab. Der Wert für *delay* wurde gerade so hoch gewählt, daß mindestens 95% der UDP-Pakete den Empfänger-nttcp erreichten. Analog für ibode2/4.

Test 4: Fore-API (AAL 3/4, 5): Durchsatz bei variabler Nachrichtenlänge

Sender hpbode3: nttcp -t -a -[4|5] -n<anzahl> -l<laenge> -g<delay> hpbode4.atm
Empfänger hpbode4: nttcp -r -a -l<laenge>

Nachrichtenlänge:

- Startwert: 512 Bytes
- Endwert: 4092 Bytes (ibode2/4), 9188 Bytes (hpbode3/4)
- Inkrement: 512 Bytes (ibode2/4), 1024 Bytes (hpbode3/4)

Die Anzahl n Nachrichten wurde für jeden Meßwert so berechnet, daß das Produkt aus Anzahl n und Länge l 16 MByte ergab. Der Wert für *delay* wurde gerade so hoch gewählt, daß mindestens 95% der Nachrichten den Empfänger-nttcp erreichten.

C

Testkonfigurationen: netperf

Test 5, 6: TCP/ATM: Durchsatz bei variabler Nachrichtenlänge und verschiedenen Socket-Sende- und Empfangspuffergrößen

Client hpbode3:

```
netperf -l 30 -H hpbode4.atm -t TCP_STREAM -- \
        -D -m <laenge> -s <SOCKET_SIZE> -S <SOCKET_SIZE>
```

Server hpbode4: `netserver &`

Verwendetes Skript: `tcp_stream_script`

Testdauer: 30 Sekunden
Nachrichtenlängen: 512 Bytes, 1KB, 2KB, 4KB, 8KB, 16KB, 32KB, 64KB
Socket-Puffergrößen (Sender/Empfänger): 8KB, 16KB, 32KB, 56KB (HP) / 64KB (IBM)

Analog für ibode2/4.

Test 7: Fore-API (AAL 3/4, 5): Empfängerdurchsatz bei variabler Nachrichtenlänge

Client hpbode3:

```
netperf -l 30 -H hpbode4.atm -t FORE_STREAM --\
        -a [4/5] -m<laenge> -M<laenge> -D /dev/fa0
```

Server hpbode4: `netserver &`

Verwendetes Skript: `fore_range_script` (modifiziertes TCP-Skript)

Testdauer: 30 Sekunden
Nachrichtenlängen:

- Startwert: 256 Bytes
- Endwert (MTU): 4092 Bytes (AIX), 9188 Bytes (HP-UX)
- Inkrement: 256 Bytes

Analog für ibode2/4.

Test 8, 9: Fore-API (AAL 5): Sender- und Empfängerdurchsatz

Client hpbode3:

```
netperf -l 30 -H hpbode4.atm -t FORE_STREAM --\
        -a 5 -m<laenge> -M<laenge> -D /dev/fa0
```

Server hpbode4: `netserver &`

Verwendetes Skript: `fore_range_script` (modifiziertes TCP-Skript)

Testdauer: 20 Sekunden
Nachrichtenlängen:

- Startwert: 16 Bytes (HP-UX), 8 Bytes (AIX)
- Endwert (MTU): 9184 Bytes (HP-UX), 4088 Bytes (AIX)
- Inkrement: 16 Bytes (HP-UX), 8 Bytes (AIX)

Analog für ibode2/4.

Test 10 – 13: Latenzzeitmessungen

Fore-API (AAL 3/4, 5):

Client hpbode3:

```
netperf -l 30 -H hpbode4.atm -t FORE_RR --\
        -r <laenge> -a [4/5] -D /dev/fa0
```

Server hpbode4: `netserver &`

Verwendetes Skript: `fore_rr_script` (modifiziertes TCP-Skript)

TCP ATM/Ethernet:

Client hpbode3:

```
netperf -l 30 -H hpbode4[.atm] -t TCP_RR --\
        -r <laenge> -s 0 -S 0 -D
```

Server hpbode4: `netserver &`

Verwendetes Skript: `tcp_rr_script`

Für die Socket-Puffer waren die Default-Größen eingestellt. Die `TCP_NODELAY`-Option war gesetzt.

UDP ATM/Ethernet:

Client hpbode3:

```
netperf -l 30 -H hpbode4[.atm] -t UDP_RR --\
        -r <laenge> -s 0 -S 0
```

Server hpbode4: `netserver &`

Verwendetes Skript: `udp_rr_script`

Für die Socket-Puffer waren die Default-Größen eingestellt.

Alle Messungen:

Testdauer: 30 Sekunden
Nachrichtenlängen:

- Test 10, 11: 1, 16, 64, 128 Bytes
- Test 12, 13: 256, 512 Bytes, 1, 2, 4, 8 KByte

Beim Fore-API darf die Nachrichtenlänge die MTU nicht überschreiten.

Analog für ibode2/4.

D

Konfiguration des IBM TurboWays ATM-Adapter

Folgende Zeilen werden in die Datei `/etc/rc.net` eingetragen:

```
/usr/sbin/no -o sb_max=6000000   # 6 MB; Limit an mbufs pro Socket
/usr/sbin/no -o thewall=6000     # 6 MB; Max. Groesse des mbuf-Pool
/usr/sbin/no -o lowclust=200     # min. 200 freie Cluster
/usr/sbin/no -o mb_cl_hiwat=1400 # max. 1400 freie Cluster
/usr/sbin/no -o lowmbuf=300      # min. 300 freie mbufs
/usr/sbin/no -o rfc1323=1        # Socket-Puffer > 64KB moeglich
/usr/sbin/no -o tcp_sendspace=262144   # Default: 256KB
/usr/sbin/no -o tcp_recvspace=524288   # Default: 512KB
/usr/sbin/no -o udp_sendspace=65536    # Default:  64KB
/usr/sbin/no -o udp_recvspace=524288   # Default: 512KB
```

Literatur

[1] ATM Forum: *ATM User Network Interface Specifikation 3.0*[1], Prentice Hall, 1993

[2] Elmar Bartel: *How fast is ATM?*[2], Institut für Informatik, Technische Universität München, 1995

[3] Shue-Ling Chang, David H.C. Du, Jenwei Hsieh, Mengjou Lin, Rose P. Tsang: *Enhanced PVM Communication over a High-Speed Local Area Network*[3], Distributed MultiMedia Center & Computer Science Department, University of Minnesota, 1995

[4] Douglas E. Comer, John C. Lin: *TCP Buffering and Performance over an ATM Network*, Internetworking: Research and Experience, Vol. 6 (1995), p. 1-13

[5] Sudheer Dharanikota, Kurt Maly, C. M. Overstreet: *Performance evaluation of TCP(UDP)/IP over ATM networks*[4], Computer Science Department, Old Dominion University, Norfolk, 1995

[6] Patrick W. Dowd, Todd M. Carozzi, Frank A. Pellegrino, Amy Xin Chen: *Native ATM Application Programmer Interface. Testbed for Cluster-based Computing*, Proceedings of the 10th Int. Parallel Processing Symposium (IPPS), Hawaii, April 1996

[7] FORE Systems, Inc. (Hrsg.): *ForeRunner ASX-200 ATM Switch User's Manual*, FORE Systems, Inc., Warrendale, 1994

[8] FORE Systems, Inc. (Hrsg.): *ForeRunner HPA-200 Eisa Bus ATM Adapter User's Manual*, FORE Systems, Inc., Warrendale, 1994

[9] FORE Systems, Inc. (Hrsg.): *ForeRunner MCA-200 ATM Microchannel Adapter User's Manual*, FORE Systems, Inc., Warrendale, 1994

[10] Yves A. Fouquet, Richard A. Schneemann, David E. Cypher, Alan Mink: *ATM Performance Measurement: Throughput Bottlenecks and Technology Barriers*[5], National Institute of Standards and Technology, Gaithersburg, Maryland, 1994

[11] G. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderam: *PVM 3 User's Guide and Reference Manual*, Oak Ridge National Laboratory, 1993

[12] Walter J. Goralski: *Introduction to ATM networking*, McGraw-Hill, Inc., New York, 1995

[1] WWW: `http://www.atmforum.com/atmforum/approved-specs.html`
[2] WWW: `http://www.leo.org/~bartel/howfast.html`
[3] WWW: `ftp://ftp.cs.umn.edu/users/du/pvm-atm/www.html`
[4] WWW: `ftp://ftp.cs.odu.edu/pub/waters/techreports/TR_94_23.ps.Z`
[5] WWW: `http://cmr.ncsl.nist.gov/multikron/articles.html`

[13] Athanasios Gouvedaris: *Messung der Leistungsfähigkeit von E/A-Systemen und Netzwerken in einer Ethernet-, ATM- und SP2-Umgebung*, Diplomarbeit, Institut für Informatik, Technische Universität München, 1996

[14] IBM: *System Management Guide: Communications and Networks*, AIX Version 3.2

[15] Othmar Kyas: *ATM-Netzwerke: Aufbau, Funktion, Performance*, 2. Auflage, DATACOM Buchverlag GmbH, Bergheim, 1995

[16] M. Laubach: *Classical IP and ARP over ATM*, Networking Group Request for Comment RFC 1577, Hewlett-Packard Laboratories, 1994

[17] Mengjou Lin, Jenwei Hsieh, David H. C. Du, Joseph P. Thomas, James A. MacDonald: *Distributed Network Computing over Local ATM Networks*[6], University of Minnesota, 1995

[18] T. Luckenbach, R. Ruppelt, F. Schulz: *Performance Experiments within Local ATM Networks*[7], GMD-FOKUS Berlin, 1995

[19] HP: *Netperf: A Network Performance Benchmark*, Manual for Revision 2.1, Information Networks Division, Hewlett-Packard Company, February 15, 1996

[20] Andreas Siegert: *Real Life TCP/IP, TCP/IP configuration and trouble shooting, Part 2: tuning and trouble shooting*, Folienset zur EMEA POWER Academy 1996, IBM Deutschland Informationssysteme GmbH, 1995

[21] Georg Stellner: *Methoden zur Sicherungspunkterzeugung in parallelen und verteilten Systemen*. LRR-TUM Research Report Series. Shaker Verlag, Aachen, Oktober 1996.

[22] W. Richard Stevens: *Programmieren von UNIX-Netzen*, Carl Hanser Verlag, München, 1992

[23] Andrew S. Tanenbaum: *Computer-Netzwerke*, Wolfram's Fachverlag, 1. Auflage 1990

[24] H. Zhou, G. A. Geist: *Faster Message Passing in PVM*[8], Oak Ridge National Laboratory, 1995

[6] WWW: `ftp://ftp.cs.umn.edu/users/du/pvm-atm/www.html`

[7] WWW: `http://www.fokus.gmd.de/atc/books.html`

[8] WWW: `http://www.epm.ornl.gov/~zhou/patm.ps`

SFB 342: Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

bisher erschienen :

Reihe A

342/1/90 A	Robert Gold, Walter Vogler: Quality Criteria for Partial Order Semantics of Place/Transition-Nets, Januar 1990
342/2/90 A	Reinhard Fößmeier: Die Rolle der Lastverteilung bei der numerischen Parallelprogrammierung, Februar 1990
342/3/90 A	Klaus-Jörn Lange, Peter Rossmanith: Two Results on Unambiguous Circuits, Februar 1990
342/4/90 A	Michael Griebel: Zur Lösung von Finite-Differenzen- und Finite-Element-Gleichungen mittels der Hierarchischen Transformations-Mehrgitter-Methode
342/5/90 A	Reinhold Letz, Johann Schumann, Stephan Bayerl, Wolfgang Bibel: SETHEO: A High-Performance Theorem Prover
342/6/90 A	Johann Schumann, Reinhold Letz: PARTHEO: A High Performance Parallel Theorem Prover
342/7/90 A	Johann Schumann, Norbert Trapp, Martin van der Koelen: SETHEO/PARTHEO Users Manual
342/8/90 A	Christian Suttner, Wolfgang Ertel: Using Connectionist Networks for Guiding the Search of a Theorem Prover
342/9/90 A	Hans-Jörg Beier, Thomas Bemmerl, Arndt Bode, Hubert Ertl, Olav Hansen, Josef Haunerdinger, Paul Hofstetter, Jaroslav Kremenek, Robert Lindhof, Thomas Ludwig, Peter Luksch, Thomas Treml: TOPSYS, Tools for Parallel Systems (Artikelsammlung)
342/10/90 A	Walter Vogler: Bisimulation and Action Refinement
342/11/90 A	Jörg Desel, Javier Esparza: Reachability in Reversible Free- Choice Systems
342/12/90 A	Rob van Glabbeek, Ursula Goltz: Equivalences and Refinement
342/13/90 A	Rob van Glabbeek: The Linear Time - Branching Time Spectrum
342/14/90 A	Johannes Bauer, Thomas Bemmerl, Thomas Treml: Leistungsanalyse von verteilten Beobachtungs- und Bewertungswerkzeugen
342/15/90 A	Peter Rossmanith: The Owner Concept for PRAMs
342/16/90 A	G. Böckle, S. Trosch: A Simulator for VLIW-Architectures
342/17/90 A	P. Slavkovsky, U. Rüde: Schnellere Berechnung klassischer Matrix-Multiplikationen

Reihe A

342/18/90 A Christoph Zenger: SPARSE GRIDS

342/19/90 A Michael Griebel, Michael Schneider, Christoph Zenger: A combination technique for the solution of sparse grid problems

342/20/90 A Michael Griebel: A Parallelizable and Vectorizable Multi- Level-Algorithm on Sparse Grids

342/21/90 A V. Diekert, E. Ochmanski, K. Reinhardt: On confluent semi-commutations-decidability and complexity results

342/22/90 A Manfred Broy, Claus Dendorfer: Functional Modelling of Operating System Structures by Timed Higher Order Stream Processing Functions

342/23/90 A Rob van Glabbeek, Ursula Goltz: A Deadlock-sensitive Congruence for Action Refinement

342/24/90 A Manfred Broy: On the Design and Verification of a Simple Distributed Spanning Tree Algorithm

342/25/90 A Thomas Bemmerl, Arndt Bode, Peter Braun, Olav Hansen, Peter Luksch, Roland Wismüller: TOPSYS - Tools for Parallel Systems (User's Overview and User's Manuals)

342/26/90 A Thomas Bemmerl, Arndt Bode, Thomas Ludwig, Stefan Tritscher: MMK - Multiprocessor Multitasking Kernel (User's Guide and User's Reference Manual)

342/27/90 A Wolfgang Ertel: Random Competition: A Simple, but Efficient Method for Parallelizing Inference Systems

342/28/90 A Rob van Glabbeek, Frits Vaandrager: Modular Specification of Process Algebras

342/29/90 A Rob van Glabbeek, Peter Weijland: Branching Time and Abstraction in Bisimulation Semantics

342/30/90 A Michael Griebel: Parallel Multigrid Methods on Sparse Grids

342/31/90 A Rolf Niedermeier, Peter Rossmanith: Unambiguous Simulations of Auxiliary Pushdown Automata and Circuits

342/32/90 A Inga Niepel, Peter Rossmanith: Uniform Circuits and Exclusive Read PRAMs

342/33/90 A Dr. Hermann Hellwagner: A Survey of Virtually Shared Memory Schemes

342/1/91 A Walter Vogler: Is Partial Order Semantics Necessary for Action Refinement?

342/2/91 A Manfred Broy, Frank Dederichs, Claus Dendorfer, Rainer Weber: Characterizing the Behaviour of Reactive Systems by Trace Sets

342/3/91 A Ulrich Furbach, Christian Suttner, Bertram Fronhöfer: Massively Parallel Inference Systems

342/4/91 A Rudolf Bayer: Non-deterministic Computing, Transactions and Recursive Atomicity

Reihe A

342/5/91 A	Robert Gold: Dataflow semantics for Petri nets
342/6/91 A	A. Heise; C. Dimitrovici: Transformation und Komposition von P/T-Netzen unter Erhaltung wesentlicher Eigenschaften
342/7/91 A	Walter Vogler: Asynchronous Communication of Petri Nets and the Refinement of Transitions
342/8/91 A	Walter Vogler: Generalized OM-Bisimulation
342/9/91 A	Christoph Zenger, Klaus Hallatschek: Fouriertransformation auf dünnen Gittern mit hierarchischen Basen
342/10/91 A	Erwin Loibl, Hans Obermaier, Markus Pawlowski: Towards Parallelism in a Relational Database System
342/11/91 A	Michael Werner: Implementierung von Algorithmen zur Kompaktifizierung von Programmen für VLIW-Architekturen
342/12/91 A	Reiner Müller: Implementierung von Algorithmen zur Optimierung von Schleifen mit Hilfe von Software-Pipelining Techniken
342/13/91 A	Sally Baker, Hans-Jörg Beier, Thomas Bemmerl, Arndt Bode, Hubert Ertl, Udo Graf, Olav Hansen, Josef Haunerdinger, Paul Hofstetter, Rainer Knödlseder, Jaroslav Kremenek, Siegfried Langenbuch, Robert Lindhof, Thomas Ludwig, Peter Luksch, Roy Milner, Bernhard Ries, Thomas Treml: TOPSYS - Tools for Parallel Systems (Artikelsammlung); 2., erweiterte Auflage
342/14/91 A	Michael Griebel: The combination technique for the sparse grid solution of PDE's on multiprocessor machines
342/15/91 A	Thomas F. Gritzner, Manfred Broy: A Link Between Process Algebras and Abstract Relation Algebras?
342/16/91 A	Thomas Bemmerl, Arndt Bode, Peter Braun, Olav Hansen, Thomas Treml, Roland Wismüller: The Design and Implementation of TOPSYS
342/17/91 A	Ulrich Furbach: Answers for disjunctive logic programs
342/18/91 A	Ulrich Furbach: Splitting as a source of parallelism in disjunctive logic programs
342/19/91 A	Gerhard W. Zumbusch: Adaptive parallele Multilevel-Methoden zur Lösung elliptischer Randwertprobleme
342/20/91 A	M. Jobmann, J. Schumann: Modelling and Performance Analysis of a Parallel Theorem Prover
342/21/91 A	Hans-Joachim Bungartz: An Adaptive Poisson Solver Using Hierarchical Bases and Sparse Grids
342/22/91 A	Wolfgang Ertel, Theodor Gemenis, Johann M. Ph. Schumann, Christian B. Suttner, Rainer Weber, Zongyan Qiu: Formalisms and Languages for Specifying Parallel Inference Systems

Reihe A

342/23/91 A	Astrid Kiehn: Local and Global Causes
342/24/91 A	Johann M.Ph. Schumann: Parallelization of Inference Systems by using an Abstract Machine
342/25/91 A	Eike Jessen: Speedup Analysis by Hierarchical Load Decomposition
342/26/91 A	Thomas F. Gritzner: A Simple Toy Example of a Distributed System: On the Design of a Connecting Switch
342/27/91 A	Thomas Schnekenburger, Andreas Weininger, Michael Friedrich: Introduction to the Parallel and Distributed Programming Language ParMod-C
342/28/91 A	Claus Dendorfer: Funktionale Modellierung eines Postsystems
342/29/91 A	Michael Griebel: Multilevel algorithms considered as iterative methods on indefinite systems
342/30/91 A	W. Reisig: Parallel Composition of Liveness
342/31/91 A	Thomas Bemmerl, Christian Kasperbauer, Martin Mairandres, Bernhard Ries: Programming Tools for Distributed Multiprocessor Computing Environments
342/32/91 A	Frank Leßke: On constructive specifications of abstract data types using temporal logic
342/1/92 A	L. Kanal, C.B. Suttner (Editors): Informal Proceedings of the Workshop on Parallel Processing for AI
342/2/92 A	Manfred Broy, Frank Dederichs, Claus Dendorfer, Max Fuchs, Thomas F. Gritzner, Rainer Weber: The Design of Distributed Systems - An Introduction to FOCUS
342/2-2/92 A	Manfred Broy, Frank Dederichs, Claus Dendorfer, Max Fuchs, Thomas F. Gritzner, Rainer Weber: The Design of Distributed Systems - An Introduction to FOCUS - Revised Version (erschienen im Januar 1993)
342/3/92 A	Manfred Broy, Frank Dederichs, Claus Dendorfer, Max Fuchs, Thomas F. Gritzner, Rainer Weber: Summary of Case Studies in FOCUS - a Design Method for Distributed Systems
342/4/92 A	Claus Dendorfer, Rainer Weber: Development and Implementation of a Communication Protocol - An Exercise in FOCUS
342/5/92 A	Michael Friedrich: Sprachmittel und Werkzeuge zur Unterstüt- zung paralleler und verteilter Programmierung
342/6/92 A	Thomas F. Gritzner: The Action Graph Model as a Link between Abstract Relation Algebras and Process-Algebraic Specifications
342/7/92 A	Sergei Gorlatch: Parallel Program Development for a Recursive Numerical Algorithm: a Case Study
342/8/92 A	Henning Spruth, Georg Sigl, Frank Johannes: Parallel Algorithms for Slicing Based Final Placement

Reihe A

342/9/92 A	Herbert Bauer, Christian Sporrer, Thomas Krodel: On Distributed Logic Simulation Using Time Warp
342/10/92 A	H. Bungartz, M. Griebel, U. Rüde: Extrapolation, Combination and Sparse Grid Techniques for Elliptic Boundary Value Problems
342/11/92 A	M. Griebel, W. Huber, U. Rüde, T. Störtkuhl: The Combination Technique for Parallel Sparse-Grid-Preconditioning and -Solution of PDEs on Multiprocessor Machines and Workstation Networks
342/12/92 A	Rolf Niedermeier, Peter Rossmanith: Optimal Parallel Algorithms for Computing Recursively Defined Functions
342/13/92 A	Rainer Weber: Eine Methodik für die formale Anforderungsspezifkation verteilter Systeme
342/14/92 A	Michael Griebel: Grid– and point–oriented multilevel algorithms
342/15/92 A	M. Griebel, C. Zenger, S. Zimmer: Improved multilevel algorithms for full and sparse grid problems
342/16/92 A	J. Desel, D. Gomm, E. Kindler, B. Paech, R. Walter: Bausteine eines kompositionalen Beweiskalküls für netzmodellierte Systeme
342/17/92 A	Frank Dederichs: Transformation verteilter Systeme: Von applikativen zu prozeduralen Darstellungen
342/18/92 A	Andreas Listl, Markus Pawlowski: Parallel Cache Management of a RDBMS
342/19/92 A	Erwin Loibl, Markus Pawlowski, Christian Roth: PART: A Parallel Relational Toolbox as Basis for the Optimization and Interpretation of Parallel Queries
342/20/92 A	Jörg Desel, Wolfgang Reisig: The Synthesis Problem of Petri Nets
342/21/92 A	Robert Balder, Christoph Zenger: The d-dimensional Helmholtz equation on sparse Grids
342/22/92 A	Ilko Michler: Neuronale Netzwerk-Paradigmen zum Erlernen von Heuristiken
342/23/92 A	Wolfgang Reisig: Elements of a Temporal Logic. Coping with Concurrency
342/24/92 A	T. Störtkuhl, Chr. Zenger, S. Zimmer: An asymptotic solution for the singularity at the angular point of the lid driven cavity
342/25/92 A	Ekkart Kindler: Invariants, Compositionality and Substitution
342/26/92 A	Thomas Bonk, Ulrich Rüde: Performance Analysis and Optimization of Numerically Intensive Programs
342/1/93 A	M. Griebel, V. Thurner: The Efficient Solution of Fluid Dynamics Problems by the Combination Technique
342/2/93 A	Ketil Stølen, Frank Dederichs, Rainer Weber: Assumption / Commitment Rules for Networks of Asynchronously Communicating Agents

Reihe A

342/3/93 A	Thomas Schnekenburger: A Definition of Efficiency of Parallel Programs in Multi-Tasking Environments
342/4/93 A	Hans-Joachim Bungartz, Michael Griebel, Dierk Röschke, Christoph Zenger: A Proof of Convergence for the Combination Technique for the Laplace Equation Using Tools of Symbolic Computation
342/5/93 A	Manfred Kunde, Rolf Niedermeier, Peter Rossmanith: Faster Sorting and Routing on Grids with Diagonals
342/6/93 A	Michael Griebel, Peter Oswald: Remarks on the Abstract Theory of Additive and Multiplicative Schwarz Algorithms
342/7/93 A	Christian Sporrer, Herbert Bauer: Corolla Partitioning for Distributed Logic Simulation of VLSI Circuits
342/8/93 A	Herbert Bauer, Christian Sporrer: Reducing Rollback Overhead in Time-Warp Based Distributed Simulation with Optimized Incremental State Saving
342/9/93 A	Peter Slavkovsky: The Visibility Problem for Single-Valued Surface (z = f(x,y)): The Analysis and the Parallelization of Algorithms
342/10/93 A	Ulrich Rüde: Multilevel, Extrapolation, and Sparse Grid Methods
342/11/93 A	Hans Regler, Ulrich Rüde: Layout Optimization with Algebraic Multigrid Methods
342/12/93 A	Dieter Barnard, Angelika Mader: Model Checking for the Modal Mu-Calculus using Gauß Elimination
342/13/93 A	Christoph Pflaum, Ulrich Rüde: Gauß' Adaptive Relaxation for the Multilevel Solution of Partial Differential Equations on Sparse Grids
342/14/93 A	Christoph Pflaum: Convergence of the Combination Technique for the Finite Element Solution of Poisson's Equation
342/15/93 A	Michael Luby, Wolfgang Ertel: Optimal Parallelization of Las Vegas Algorithms
342/16/93 A	Hans-Joachim Bungartz, Michael Griebel, Dierk Röschke, Christoph Zenger: Pointwise Convergence of the Combination Technique for Laplace's Equation
342/17/93 A	Georg Stellner, Matthias Schumann, Stefan Lamberts, Thomas Ludwig, Arndt Bode, Martin Kiehl und Rainer Mehlhorn: Developing Multicomputer Applications on Networks of Workstations Using NXLib
342/18/93 A	Max Fuchs, Ketil Stølen: Development of a Distributed Min/Max Component
342/19/93 A	Johann K. Obermaier: Recovery and Transaction Management in Write-optimized Database Systems
342/20/93 A	Sergej Gorlatch: Deriving Efficient Parallel Programs by Systemating Coarsing Specification Parallelism

Reihe A

342/01/94 A	Reiner Hüttl, Michael Schneider: Parallel Adaptive Numerical Simulation
342/02/94 A	Henning Spruth, Frank Johannes: Parallel Routing of VLSI Circuits Based on Net Independency
342/03/94 A	Henning Spruth, Frank Johannes, Kurt Antreich: PHIroute: A Parallel Hierarchical Sea-of-Gates Router
342/04/94 A	Martin Kiehl, Rainer Mehlhorn, Matthias Schumann: Parallel Multiple Shooting for Optimal Control Problems Under NX/2
342/05/94 A	Christian Suttner, Christoph Goller, Peter Krauss, Klaus-Jörn Lange, Ludwig Thomas, Thomas Schnekenburger: Heuristic Optimization of Parallel Computations
342/06/94 A	Andreas Listl: Using Subpages for Cache Coherency Control in Parallel Database Systems
342/07/94 A	Manfred Broy, Ketil Stølen: Specification and Refinement of Finite Dataflow Networks - a Relational Approach
342/08/94 A	Katharina Spies: Funktionale Spezifikation eines Kommunikationsprotokolls
342/09/94 A	Peter A. Krauss: Applying a New Search Space Partitioning Method to Parallel Test Generation for Sequential Circuits
342/10/94 A	Manfred Broy: A Functional Rephrasing of the Assumption/Commitment Specification Style
342/11/94 A	Eckhardt Holz, Ketil Stølen: An Attempt to Embed a Restricted Version of SDL as a Target Language in Focus
342/12/94 A	Christoph Pflaum: A Multi-Level-Algorithm for the Finite-Element-Solution of General Second Order Elliptic Differential Equations on Adaptive Sparse Grids
342/13/94 A	Manfred Broy, Max Fuchs, Thomas F. Gritzner, Bernhard Schätz, Katharina Spies, Ketil Stølen: Summary of Case Studies in FOCUS - a Design Method for Distributed Systems
342/14/94 A	Maximilian Fuchs: Technologieabhängigkeit von Spezifikationen digitaler Hardware
342/15/94 A	M. Griebel, P. Oswald: Tensor Product Type Subspace Splittings And Multilevel Iterative Methods For Anisotropic Problems
342/16/94 A	Gheorghe Ştefănescu: Algebra of Flownomials
342/17/94 A	Ketil Stølen: A Refinement Relation Supporting the Transition from Unbounded to Bounded Communication Buffers
342/18/94 A	Michael Griebel, Tilman Neuhoeffer: A Domain-Oriented Multilevel Algorithm-Implementation and Parallelization
342/19/94 A	Michael Griebel, Walter Huber: Turbulence Simulation on Sparse Grids Using the Combination Method

Reihe A

342/20/94 A Johann Schumann: Using the Theorem Prover SETHEO for verifying the development of a Communication Protocol in FOCUS - A Case Study -
342/01/95 A Hans-Joachim Bungartz: Higher Order Finite Elements on Sparse Grids
342/02/95 A Tao Zhang, Seonglim Kang, Lester R. Lipsky: The Performance of Parallel Computers: Order Statistics and Amdahl's Law
342/03/95 A Lester R. Lipsky, Appie van de Liefvoort: Transformation of the Kronecker Product of Identical Servers to a Reduced Product Space
342/04/95 A Pierre Fiorini, Lester R. Lipsky, Wen-Jung Hsin, Appie van de Liefvoort: Auto-Correlation of Lag-k For Customers Departing From Semi-Markov Processes
342/05/95 A Sascha Hilgenfeldt, Robert Balder, Christoph Zenger: Sparse Grids: Applications to Multi-dimensional Schrödinger Problems
342/06/95 A Maximilian Fuchs: Formal Design of a Model-N Counter
342/07/95 A Hans-Joachim Bungartz, Stefan Schulte: Coupled Problems in Microsystem Technology
342/08/95 A Alexander Pfaffinger: Parallel Communication on Workstation Networks with Complex Topologies
342/09/95 A Ketil Stølen: Assumption/Commitment Rules for Data-flow Networks - with an Emphasis on Completeness
342/10/95 A Ketil Stølen, Max Fuchs: A Formal Method for Hardware/Software Co-Design
342/11/95 A Thomas Schnekenburger: The ALDY Load Distribution System
342/12/95 A Javier Esparza, Stefan Römer, Walter Vogler: An Improvement of McMillan's Unfolding Algorithm
342/13/95 A Stephan Melzer, Javier Esparza: Checking System Properties via Integer Programming
342/14/95 A Radu Grosu, Ketil Stølen: A Denotational Model for Mobile Point-to-Point Dataflow Networks
342/15/95 A Andrei Kovalyov, Javier Esparza: A Polynomial Algorithm to Compute the Concurrency Relation of Free-Choice Signal Transition Graphs
342/16/95 A Bernhard Schätz, Katharina Spies: Formale Syntax zur logischen Kernsprache der Focus-Entwicklungsmethodik
342/17/95 A Georg Stellner: Using CoCheck on a Network of Workstations
342/18/95 A Arndt Bode, Thomas Ludwig, Vaidy Sunderam, Roland Wismüller: Workshop on PVM, MPI, Tools and Applications
342/19/95 A Thomas Schnekenburger: Integration of Load Distribution into ParMod-C
342/20/95 A Ketil Stølen: Refinement Principles Supporting the Transition from Asynchronous to Synchronous Communication

Reihe A

342/21/95 A	Andreas Listl, Giannis Bozas: Performance Gains Using Subpages for Cache Coherency Control
342/22/95 A	Volker Heun, Ernst W. Mayr: Embedding Graphs with Bounded Treewidth into Optimal Hypercubes
342/23/95 A	Petr Jančar, Javier Esparza: Deciding Finiteness of Petri Nets up to Bisimulation
342/24/95 A	M. Jung, U. Rüde: Implicit Extrapolation Methods for Variable Coefficient Problems
342/01/96 A	Michael Griebel, Tilman Neunhoeffer, Hans Regler: Algebraic Multigrid Methods for the Solution of the Navier-Stokes Equations in Complicated Geometries
342/02/96 A	Thomas Grauschopf, Michael Griebel, Hans Regler: Additive Multilevel-Preconditioners based on Bilinear Interpolation, Matrix Dependent Geometric Coarsening and Algebraic-Multigrid Coarsening for Second Order Elliptic PDEs
342/03/96 A	Volker Heun, Ernst W. Mayr: Optimal Dynamic Edge-Disjoint Embeddings of Complete Binary Trees into Hypercubes
342/04/96 A	Thomas Huckle: Efficient Computation of Sparse Approximate Inverses
342/05/96 A	Thomas Ludwig, Roland Wismüller, Vaidy Sunderam, Arndt Bode: OMIS — On-line Monitoring Interface Specification
342/06/96 A	Ekkart Kindler: A Compositional Partial Order Semantics for Petri Net Components
342/07/96 A	Richard Mayr: Some Results on Basic Parallel Processes
342/08/96 A	Ralph Radermacher, Frank Weimer: INSEL Syntax-Bericht
342/09/96 A	P.P. Spies, C. Eckert, M. Lange, D. Marek, R. Radermacher, F. Weimer, H.-M. Windisch: Sprachkonzepte zur Konstruktion verteilter Systeme
342/10/96 A	Stefan Lamberts, Thomas Ludwig, Christian Röder, Arndt Bode: PFS-Lib – A File System for Parallel Programming Environments
342/11/96 A	Manfred Broy, Gheorghe Ştefănescu: The Algebra of Stream Processing Functions
342/12/96 A	Javier Esparza: Reachability in Live and Safe Free-Choice Petri Nets is NP-complete
342/13/96 A	Radu Grosu, Ketil Stølen: A Denotational Model for Mobile Many-to-Many Data-flow Networks
342/14/96 A	Giannis Bozas, Michael Jaedicke, Andreas Listl, Bernhard Mitschang, Angelika Reiser, Stephan Zimmermann: On Transforming a Sequential SQL-DBMS into a Parallel One: First Results and Experiences of the MIDAS Project

Reihe A

342/15/96 A	Richard Mayr: A Tableau System for Model Checking Petri Nets with a Fragment of the Linear Time μ -Calculus
342/16/96 A	Ursula Hinkel, Katharina Spies: Anleitung zur Spezifikation von mobilen, dynamischen Focus-Netzen
342/17/96 A	Richard Mayr: Model Checking PA-Processes
342/18/96 A	Michaela Huhn: Put your Model Checker on Diet: Verification on Local States
342/01/97 A	Tobias Müller, Stefan Lamberts, Ursula Maier, Georg Stellner: Evaluierung der Leistungsfähigkeit eines ATM-Netzes mit parallelen Programmierbibliotheken

SFB 342 : Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

Reihe B

342/1/90 B	Wolfgang Reisig: Petri Nets and Algebraic Specifications
342/2/90 B	Jörg Desel: On Abstraction of Nets
342/3/90 B	Jörg Desel: Reduction and Design of Well-behaved Free-choice Systems
342/4/90 B	Franz Abstreiter, Michael Friedrich, Hans-Jürgen Plewan: Das Werkzeug runtime zur Beobachtung verteilter und paralleler Programme
342/1/91 B	Barbara Paech1: Concurrency as a Modality
342/2/91 B	Birgit Kandler, Markus Pawlowski: SAM: Eine Sortier- Toolbox - Anwenderbeschreibung
342/3/91 B	Erwin Loibl, Hans Obermaier, Markus Pawlowski: 2. Workshop über Parallelisierung von Datenbanksystemen
342/4/91 B	Werner Pohlmann: A Limitation of Distributed Simulation Methods
342/5/91 B	Dominik Gomm, Ekkart Kindler: A Weakly Coherent Virtually Shared Memory Scheme: Formal Specification and Analysis
342/6/91 B	Dominik Gomm, Ekkart Kindler: Causality Based Specification and Correctness Proof of a Virtually Shared Memory Scheme
342/7/91 B	W. Reisig: Concurrent Temporal Logic
342/1/92 B	Malte Grosse, Christian B. Suttner: A Parallel Algorithm for Set-of-Support
	Christian B. Suttner: Parallel Computation of Multiple Sets-of-Support
342/2/92 B	Arndt Bode, Hartmut Wedekind: Parallelrechner: Theorie, Hardware, Software, Anwendungen
342/1/93 B	Max Fuchs: Funktionale Spezifikation einer Geschwindigkeitsregelung
342/2/93 B	Ekkart Kindler: Sicherheits- und Lebendigkeitseigenschaften: Ein Literaturüberblick
342/1/94 B	Andreas Listl; Thomas Schnekenburger; Michael Friedrich: Zum Entwurf eines Prototypen für MIDAS