Entwurf und Implementierung von Dienstgütemessungen im ATM-Netz

Studienarbeit im Fach Informatik

vorgelegt von

Falko Dreßler

geboren am 2.12.1971 in Dresden

Angefertigt am

Institut für Mathematische Maschinen und Datenverarbeitung IV

Friedrich-Alexander-Universität Erlangen-Nürnberg

Betreuer: **Dipl. Inf. Martin Heyer (RRZE)**
Dr. Peter Holleczek (RRZE)
Prof. Dr. Fridolin Hofmann

Begin der Arbeit: 21.2.1996
Ende der Arbeit: 26.4.1996

Erklärung:

Ich versichere, daß ich diese Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe, und daß die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, den 25. April 1996

Falko Dreßler

Danksagung:

Hiermit möchte ich meiner Freundin Sylke für ihre schier unendliche Geduld danken, mit der sie mich bei meiner Arbeit unterstützt hat.

Außerdem danke ich allen Mitarbeitern des RRZE, vor allem aber Herrn Dr. Peter Hollezcek, für ihre tatkräftige Unterstützung, mit der sie mir seit vielen Jahren bei meiner Arbeit im RRZE, bei der ich sehr viel lernen konnte, was über den Standardlehrstoff an der Universität herausgeht, beiseite gestanden haben, und die mir eine Studienarbeit ermöglichten, die mir viel Freude bereitet und mich in meiner Ausbildung vorangetrieben hat.

Die folgenden Produkt- und Firmenbezeichnungen sind **Warenzeichen** der jeweiligen Firmen und Organisationen:

Sun, Sun Microsystems, Sun Workstation, SunATM, SunOS, Solaris, SPARC, SPARCstation / Sun Microsystems, Inc.

UNIX® / UNIX Systems Laboratories, Inc.

X Window System, X11 / Massachusetts Institute of Technologie

ForeRunner / Fore Systems

Hewlett Packard, HP / Hewlett Packard, Inc.

Inhaltsverzeichnis

1 Einleitung und Motivation

In der heutigen Zeit ist es von immer stärkerer Bedeutung, daß vor allem auch wissenschaftliche Einrichtungen, wie z.B. Universitäten, durch weltumspannende Computernetze, speziell das Internet, Daten und Informationen schnell und sicher austauschen können. In Deutschland hat der DFN-Verein (Verein zur Förderung eines Deutschen Forschungsnetzes e.V.) zu diesem Zweck 1989 das sogenannte Wissenschaftsnetz (WiN) aufgebaut.

Durch das ständig steigende Datenvolumen im WiN und die wachsende Anzahl von 2 MBit/s Anschlüssen wurde es notwendig, ein neues, schnelleres WiN aufzubauen. Aus diversen RTB (Regionales Testbed) Projekten, wie z.B. dem RTB Bayern, an dem auch das RRZE (Regionales Rechenzentrum Erlangen) der Universität Erlangen-Nürnberg beteiligt ist, entstand ein ganz Deutschland umspannendes Breitband-Wissenschaftsnetz, das B-WiN.

Dabei soll das B-WiN schrittweise immer weiter ausgebaut werden, wobei immer mehr Dienste aufgenommen werden (zuerst nur IP, später auch ATM, X.25, Sprache, etc.).

Als Übertragungstechnik wurde ATM (*Asynchronous Transfer Mode*) (siehe Kapitel 2 - ATM - Prinzipien und Überlegungen) auf 34 und 155 MBit/s Leitungen vorgesehen. Die Starttopologie ist in Abbildung 1.1 wiedergegeben.

Die DeTeSystem als Auftragnehmer des B-WiN garantiert die Einhaltung bestimmter Dienstgüteparameter. Dies sind im einzelnen:

1. Verfügbarkeit des Kernnetzes und der Nutzeranschlüsse
2. Zuverlässigkeit (MTBF für das Kernnetz, die Leitungen und die Nutzeranschlüsse)
3. Durchsatz (Gesamtdurchsatz im Kernnetz)
4. Zellverlustwahrscheinlichkeit im Gesamtnetz
5. Delay (abhängig vom ATM-Dienst, wie z.B. VBR, CBR)
6. Jitter (Variation des Zelldelays, abhängig vom ATM-Dienst, wie z.B. VBR, CBR)

Die Einhaltung der Punkte 1 bis 4 läßt sich allein durch Auswertung von Statistiken des Netzmanagementsystems einfach und genau sicherstellen. Die beiden letzten Punkte hingegen (Delay und Jitter) lassen sich auf diese Weise nicht überprüfen, so daß eine spezielles Meßprogramm benötigt wird. Ein solches zu entwickeln und zu implementieren, ist die Aufgabe dieser Arbeit.

Im Kapitel 2 (ATM - Prinzipien und Überlegungen) wird beschrieben, wie man bei der Definition des ATM-Protokolls vorgegangen ist und warum man bestimmte Vorgaben gemacht hat. Es wird gezeigt, wie ATM arbeitet, welche Funktionalitäten vorgesehen wurden und welche man aus Performancegründen weggelassen hat.

Kapitel 3 (ATM Standards) beschreibt die wichtigsten Standards, die entweder von der CCITT (Comité Consultatif Internationale Télégraphique et Téléphonique) oder vom ATM Forum entworfen bzw. festgesetzt wurden. Die Standards, die später für die Entwicklung des Programms benötigt wurden, werden entsprechend genauer dargelegt.

In Kapitel 4 (Die Programmierschnittstelle der SunATM™-155 Karte) wird auf das API (*Application Programmers Interface*), also der Schnittstelle für die Entwicklung von Anwendungen, der genutzten ATM Karte eingegangen. Dieses Interface bildet die Schnittstelle zwischen dem Programm und der Hardware.

Kapitel 5 (Zeitmessungen unter UNIX) zeigt die Möglichkeiten des genutzten Betriebssystems (Solaris 2.4) Zeiten im Mikrosekundenbereich auf Benutzerebene zu messen, und gibt einen Einblick in die dabei entstehenden Probleme.

In Kapitel 6 (TMT4ATM - Transmit delay Measurement Tool for ATM) wird schließlich das im Rahmen der Arbeit entstandene Programm (TMT4ATM - Transmit delay Measurement Tool for ATM) beschrieben. Es werden Denkansätze erklärt und die grundlegende Funktionalität geschildert. Außerdem wird ein Ausblick auf mögliche Verbesserungen gegeben.

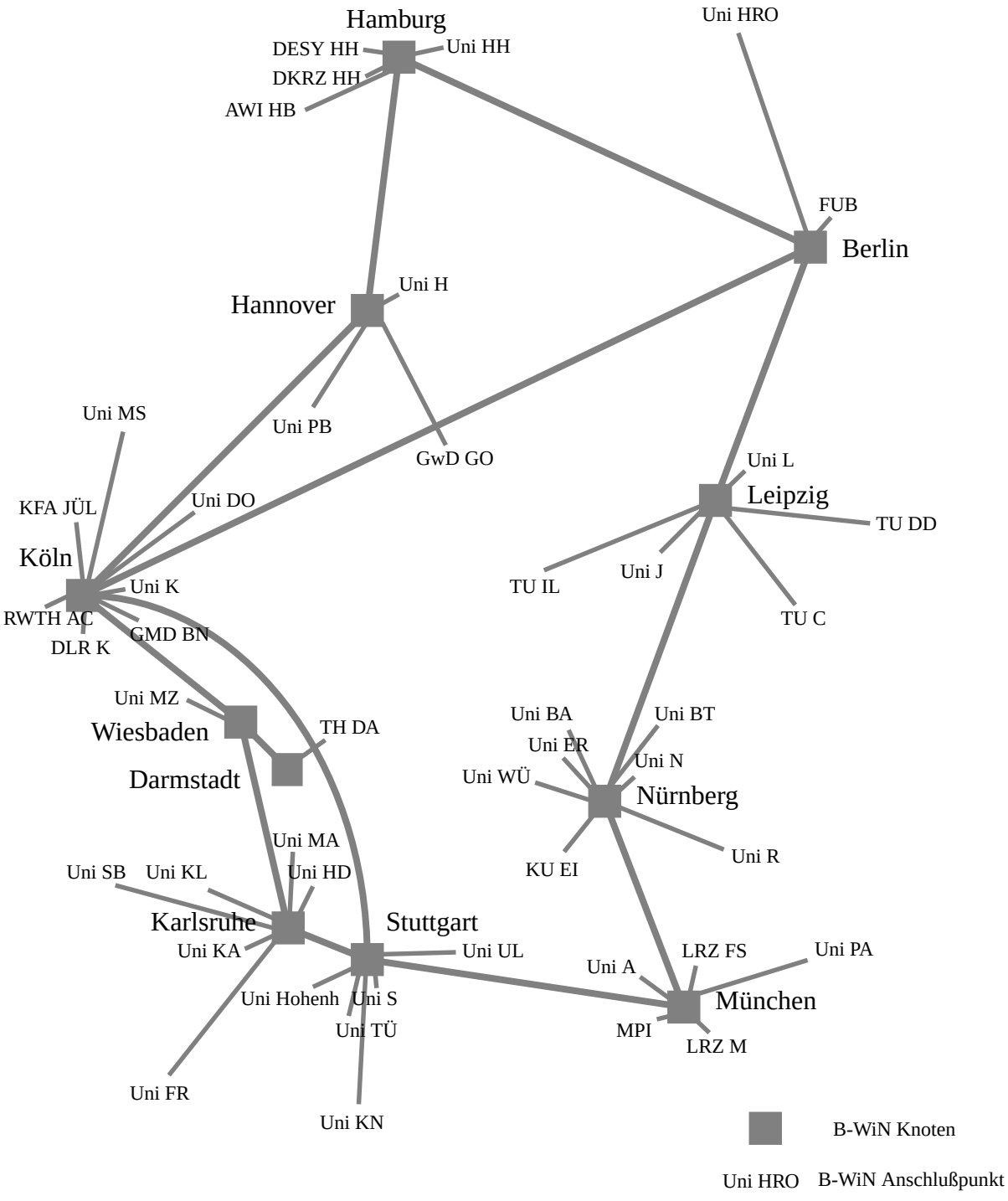

Abbildung 1.1 - Startkonfiguration des B-WiN (45 Anschlußpunkte)

2 ATM - Prinzipien und Überlegungen

ATM - *Asynchronous Transfer Mode* - ist die offizielle Bezeichnung der CCITT für ein sogenanntes *Fast Packet Switching* Protokoll. Ein anderer oft gebrauchter Name für dieselbe Technik ist ATD (*Asynchronous Time Division*). *Fast packet switching* ist ein verbindungsorientierter Paketdienst mit minimalem Funktionsumfang im Netzwerk.

In diesem Kapitel werden generelle Überlegungen beim Entwurf des ATM Protokolls erläutert. So wird z.B. auf prinzipielle Definitionen, Performance Charakteristiken, die Festlegung der Größe der Informationsblöcke und die im Header der Pakete integrierte Funktionalität eingegangen.

Das Transportprotokoll für das zukünftige B-ISDN (*Broadband Integrated Services Digital Network*) sollte diverse Vorteile gegenüber bisherigen Protokollen wie z.B. X.25 aufweisen. Die CCITT hat sich auf Grund der Eigenschaften von ATM, das genau diese im folgenden beschriebenen Vorteile aufweist, für ATM als Transportdienst entschieden.

(1) Flexibilität und Zukunftssicherheit

Um nicht in relativ kurzer Zeit für neue, z.Z. nicht vorhersagbare Dienste wieder ein neues Netzwerk entwerfen zu müssen, sollte das Transportprotokoll beliebige Services mit den unterschiedlichsten Eigenschaften und Anforderungen erbringen können.

(2) Effiziente Nutzung der verfügbaren Ressourcen

Alle Ressourcen des Netzwerkes sollen für alle Dienste gleichermaßen nutzbar sein, so daß eine optimale statistische Verteilung dieser Ressourcen ermöglicht werden kann.

(3) Ein einziges universelles Netzwerk

Um Kosten für Leitungen und das Management möglichst niedrig zu halten, soll es nur noch ein allumfassendes Netzwerk geben. Auch werden dadurch die Kosten für die Entwicklung, Kontrolle und den Aufbau des Netzes minimiert. Da über ATM alle Dienste übertragen werden können, ist ATM als universeller Transportdienst nutzbar.

2.1 Prinzipielle Definitionen

Für ATM hat man folgende Eigenschaften festgelegt:

(1) Keine Fehlerkorrektur und keine Flußkontrolle im Datenteil auf Link-to-Link Basis

Bei fehlerhafter Übertragung eines Paketes oder bei Paketverlusten durch Überlastung des Netzes und/oder der Netzwerkknoten wird keinerlei Aktion zur Korrektur des dadurch entstehenden Fehlers (z.B. Aufforderung zur Neuübertragung des Paketes) unternommen. Dies kann aufgrund der sehr hohen Qualität der (optischen) Verbindungsleitungen in Kauf genommen werden. Auch auf Flußkontrolle wird z.Z. im ATM Netzwerk verzichtet, da man durch vernünftige Ressourcenverwaltung und Warteschlangendimensionierung die Anzahl von Überläufen in den Warteschlangen kontrollieren kann, die zu Paketverlusten führt. Werte einer Paketverlustwahrscheinlichkeit von 10^{-8} bis 10^{-12} sind akzeptabel. In ATM-Netzen muß die Fehlerkorrektur und Flußkontrolle also auf End-to-End Basis durchgeführt werden. Aufgrund der fehlenden Flußkontrolle auf Link-to-Link Basis, kommt es im Netz selbst zu einer enormen Geschwindigkeitssteigerung.

(2) Verbindungsorientierte Arbeitsweise

Um Daten über das ATM Netzwerk übertragen zu können, muß vorher eine logische/virtuelle Verbindung durch das Netzwerk aufgebaut werden. Während dieses Verbindungsaufbaues werden im Netz alle notwendigen Ressourcen reserviert. Ist diese Reservierung nicht möglich, wird der Verbindungsaufbauwunsch vom ATM Netz verweigert. Beim Verbindungsabbau werden alle belegten Ressourcen wieder freigegeben. Während der Verbindung können allerdings die angeforderten Ressourcen garantiert werden.

(3) Reduzierte Funktionalität im Header

Um eine schnelle Verarbeitung der Pakete in den Netzwerkswitches (bis zu einigen Gbit/s) zu garantieren, wurde die Funktionalität des Headers auf ein Minimum reduziert. Übrig blieb die wichtigste Funktion des Headers in einem verbindungsorientierten Netzwerk, die Identifikation der virtuellen Verbindung, um das richtige Weiterleiten der Pakete im Netz zu sichern.

(4) Relativ kleine Informationsblöcke (Datenpakete)

Zur Verringerung der internen Puffer der Switches und um die Verzögerungen bei deren Verwaltung zu verringern, wurde der Informationsblock so klein wie möglich gehalten. Kleine Puffer garantieren ein kleines Delay und einen kleinen Jitter (der Begriff Jitter steht in diesem Fall für die Varianz des Delays), wie es von Echtzeitanwendungen gefordert wird.

Die in (3) und (4) geforderten Funktionalitäten lassen sich am besten durch Hardware-Switching erreichen.

2.2 Performance Charakteristiken

Die wesentlichen Performance Charakteristiken sind die Zeittransparenz, d.h. die Güte von Delay und Jitter, und die semantische Transparenz, d.h. die Güte der Übertragung (Bitfehlerrate). Diese beiden Charakteristiken werden im folgenden beschrieben.

2.2.1 Zeittransparenz

Das Delay einer Netzwerkverbindung ist einer der wichtigsten Parameter des Netzes. Vor allem bei den modernen Echtzeitdiensten, wie z.B. Sprach- und Videoübertragung, ist ein niedriges Delay und vor allem ein niedriger Jitter von starker Bedeutung.

In ATM Netzwerken kommt es zu folgenden Delays, die jedes für sich minimiert werden müssen:

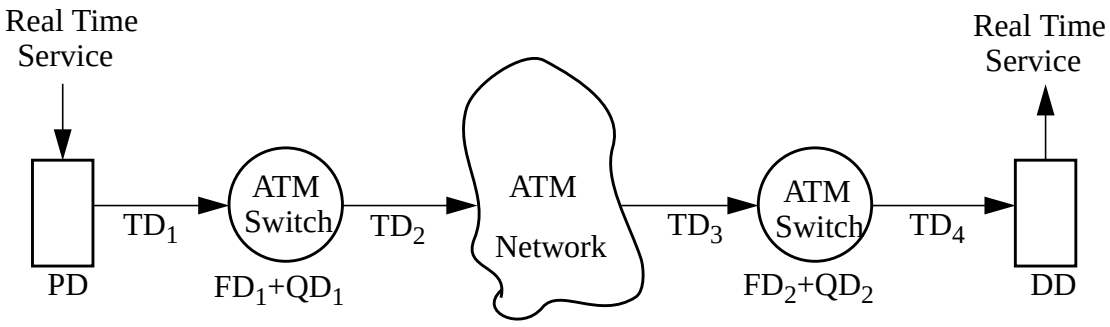

Abbildung 2.1 - Delays in einem ATM Netzwerk [Pry93]

PD..... Paketisierungsdelay (*Packetization Delay*)

TD..... Übertragungsdelay (*Transmission Delay*)

QD ..... Warteschlangendelay (*Queueing Delay*)

FD..... fixes Switching Delay (*Fixed Switching Delay*)

DD ..... Depaketisierungsdelay (*Depacketization Delay*)

(a) Übertragungsdelay (TD)

Damit ist die Verzögerung auf den physikalischen Leitungen gemeint. Auf großen Strecken wird das Übertragungsdelay auch bei besten Leitungen aus rein technischen Gründen relativ groß. Typische Werte für dieses Delay liegen bei 4 bis 5 µs pro km (abhängig vom benutzten physikalischen Medium).

(b) Paketisierungsdelay (PD)

Mit Paketisierung ist das Einpacken eines Bitstromes in zur Versendung über ein ATM Netzwerk geeignete Pakete gemeint. Durch entsprechend leistungsfähige Hardware kann man dieses Delay minimieren.

(c) fixes Switching Delay (FD)

In einem ATM Switch entsteht durch feste Übertragungswege und Schaltzeiten ein konstantes Delay, das man durch eine immer leistungsfähigere Hardware minimieren kann.

(d) Depaketisierungsdelay (DD)

Beim Empfänger entsteht durch das Zusammensetzen der Pakete zu einem kontinuierlichen Bitstrom, der auf einer höheren Ebene u.U. wieder in Pakete umgewandelt wird, wieder ein Delay. Wie beim Paketisierungsdelay kann dieses Delay durch eine besonders leistungsfähige Hardware (Hardware-Switching) minimiert werden.

(e) Warteschlangendelay (QD)

Im Netzwerkswitch wird ein variables Delay durch die Verwaltung von Warteschlangen und Puffern, in denen die Pakete vor der Weiterleitung zwischengespeichert werden, erzeugt. Dieses Delay kann man durch optimale Warteschlangenalgorithmen und Pufferverwaltungsstrategien und eine optimierte Warteschlangen-/Puffergröße minimieren.

2.2.2 Semantische Transparenz

Semantische Transparenz bedeutet die Fähigkeit eines Netzwerkes, Informationen korrekt von einer Quelle zu einem Ziel zu transportieren, u.U. mit einer limitierten, akzeptablen Anzahl von Fehlern.

Wie in jedem anderen *Packet Switching System* entstehen auch in einem ATM-Netzwerk Fehler, hervorgerufen durch Übertragungssysteme und durch Switches und Multiplexer. Allerdings sind diese Fehler aufgrund der fehlenden Fehlerkorrektur auf ATM-Ebene anders zu bewerten als in anderen herkömmlichen Netzwerken, wie z.B. X.25. Die Fehler lassen sich in drei Kategorien aufteilen:

(1) Verstümmelung des Informationsfeldes durch Übertragungsfehler

Bitfehler, die durch die Übertragung verursacht wurden, sind auf die Eigenschaften des physikalischen Übertragungsmediums zurückzuführen. Für ATM-Netzwerke werden hauptsächlich optische Medien, d.h. Glasfaserleitungen, benutzt. Diese haben bekanntermaßen eine besonders niedrige Bitfehlerwahrscheinlichkeit, da sie frei von elektromagnetischen Einflüssen sind. Da es für die physikalische Übertragung keine Rolle spielt, ob ein Bit im Informationsteil (Data) oder im Steuerteil (Header) eines Paketes liegt, ist die Fehlerrate durch Verstümmelung des Informationsfeldes folgendermaßen zu berechnen:

B_i ..... Fehlerrate durch Verstümmelung des Informationsteils

B ..... Bitfehlerrate des Mediums

h ..... Länge des Headers

i ..... Länge des Datenteils

$$B_i = \frac{i}{h+i} B \qquad \text{Gleichung 2.1}$$

Da die Fehlerwahrscheinlichkeit schon allein durch das Übertragungsmedium sehr klein ist, wird die semantische Transparenz gewährleistet.

(2) Paketverlust durch Fehler im Header

Es ist ein großer Unterschied, ob ein fehlerhaftes Bit im Informations- oder im Steuerteil eines Paketes auftaucht. Ein fehlerhafter Datenteil wird unverändert (da keine Fehlerkorrektur des Informationsteiles in ATM auf Link-to-Link Basis vorhanden ist) zum richtigen Empfänger weitergeleitet. Ein Fehler im Header allerdings führt zu einer Mißinterpretation der Steuerinformation, so daß das Paket im Switch/Multiplexer fehlerhaft weitergeleitet wird und dadurch entweder zerstört wird oder bei einem falschen Empfänger ankommt, so daß sogar zwei verschiedene Verbindungen durch einen einzigen Bitfehler in Mitleidenschaft gezogen werden können. Dadurch kann es zu einer Vervielfachung eines Fehlers kommen (Fehlermultiplikation).

Die Wahrscheinlichkeit, daß ein Fehler im Header auftaucht, ist:

B_h..... Fehlerrate durch Verstümmelung des Steuerteils

B ..... Bitfehlerrate des physikalischen Mediums

h..... Länge des Headers

i..... Länge des Informationsteil

$$B_h = \frac{h}{h+i}B$$ Gleichung 2.2

Wenn keine Fehlerkorrektur durchgeführt wird, kommt es z.B. bei vielen aufeinanderfolgenden Bitfehlern zu einer Fehlermultiplikation, da u.U. mehrere Verbindungen gleichzeitig von den Fehlern betroffen werden (im ungünstigsten Fall bekommt der richtige Empfänger nichts (Paketverlust), und ein zweiter Empfänger bekommt das nicht für ihn bestimmte Paket). Der Multiplikationseffekt der Fehler ist nicht unerheblich (siehe auch [Pry93]), eine vollständige Fehlerkorrektur aber aus Effizienzgründen nicht machbar. Ein einfacher aber sehr wirkungsvoller Ansatz zur Lösung des Problems ist ein adaptiver Fehlererkennungs- und -korrekturalgorithmus, der auch sehr einfach zu implementieren ist:

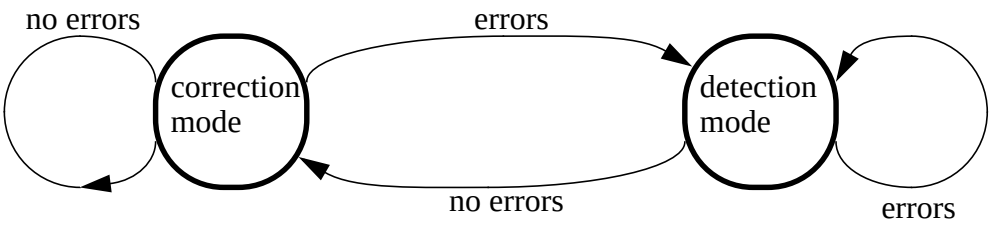

Abbildung 2.2 - Adaptiver Fehlererkennungs- und -korrekturalgorithmus [Pry93]

Im Falle von einzelnen Bitfehlern werden diese korrigiert, im Falle von Burstfehlern werden diese nur erkannt, da mit großer Wahrscheinlichkeit auch der Informationsteil des Paketes zerstört wurde, was eine Korrektur des Headers erübrigt. Treten keine Fehler auf, bleibt der Algorithmus im Korrekturmodus (*correction mode*) und wechselt erst nach dem Auftreten eines Fehlers in den Erkennungsmodus (*detection mode*). Wenn weitere Fehler folgen (Burstfehler), dann bleibt der Algorithmus solange im Erkennungsmodus, bis er beim Eintreffen des ersten korrekten Paketes zurück in den Korrekturmodus wechselt.

So kann durch einen einfachen, aber effektiven Algorithmus semantische Transparenz angenähert werden. Leider treten immer noch nicht korrigierbare Fehler auf (Burstfehler). Dieser Algorithmus wird heute in ATM angewandt.

(3) Paketverlust durch Warteschlangenüberläufe

Durch eine ideale Dimensionierung der Warteschlangen im Netzwerk kann der Paketverlust durch Warteschlangenüberläufe (*queue overflow*) minimiert werden. Um Audio- und Videoübertragungen ohne größere Fehler zu ermöglichen, muß die Paketverlustrate kleiner als 10^{-8} sein, da sonst die Qualität der Übertragung zu stark in Mitleidenschaft gezogen wird. Wenn die Warteschlangen so dimensioniert werden können, daß diese Voraussetzung erfüllt wird, kann semantische Transparenz garantiert werden. In einem verbindungsorientierten Netzwerk wie ATM, wo es der Kontrolle des Netzes unterliegt, den Aufbau einer Verbindung zuzulassen oder abzulehnen, wenn die Belastung des Netzes den Wert unter- bzw. überschreitet, für den die Warteschlangen dimensioniert wurden, ist dies theoretisch machbar.

2.3 Definition der Größe der Informationsblöcke

Bei der Definition der Größe der Informationsblöcke waren zwei Punkte zu klären. Dies ist zum einen die Frage, ob man eine feste oder eine variable Paketlänge verwendet, und zum zweiten, wie groß die Pakete eigentlich sein sollen (bei variabler Paketlänge steht die Frage nach einer minimalen bzw. maximalen Länge).

2.3.1 Variable kontra feste Paketlänge

Die wichtigste Entscheidung ist, ob nun eine variable Paketlänge oder eine feste Paketlänge verwendet werden soll. Es gibt sehr unterschiedliche Argumente für bzw. gegen beide Lösungen. Die wichtigsten und gleichzeitig die ausschlaggebenden sind die Effizienz in der Bandbreitenausnutzung des Übertragungskanals, die mögliche Switching-Performance (Geschwindigkeit des Switching kontra Komplexität) und das Delay.

2.3.1.1 Effizienz der Bandbreitenausnutzung

In einem *Packet Switching System*, in dem ein Overhead durch den Header entsteht, wird die Effizienz folgendermaßen berechnet:

η ..... Effizienz

I..... Anzahl von Informationsbytes

O ..... Anzahl von Overheadbytes

$$\eta = \frac{I}{I + O}$$ Gleichung 2.3

(a) Feste Paketlänge

Bei einer festen Paketlänge, wird die Effizienz folgendermaßen berechnet:

η_F..... Effizienz bei fester Paketlänge

L ..... Informationsbytes in einem Paket

H ..... Größe des Headers

X ..... Anzahl der zu übertragenden Informationsbytes

$$\eta_F = \frac{X}{\left|\frac{X}{L}\right|(L+H)} \qquad \text{Gleichung 2.4}$$

wobei |z| der kleinste ganzzahlige Wert ist, der größer oder gleich z ist.

Das heißt, daß die Effizienz der Bandbreitenausnutzung optimal ist für alle Informationsblöcke, deren Länge ein Vielfaches von L beträgt, d.h. im optimalen Fall ist die Effizienz:

η_{Fopt}..... Optimum der Effizienz bei fester Paketlänge

$$\eta_{Fopt} = \frac{L}{L+H} \qquad \text{Gleichung 2.5}$$

(b) Variable Paketlänge

Bei einer variablen Paketlänge ist der Overhead durch den Paketheader plus der nötigen Kennung, die die Paketlänge spezifiziert, gegeben. So berechnet man die Effizienz wie folgt:

η_V ..... Effizienz bei variabler Paketlänge

h_V..... spezifischer Overhead durch variable Paketlänge

$$\eta_V = \frac{X}{X+H+h_v} \qquad \text{Gleichung 2.6}$$

(c) Vergleich der beiden Ansätze

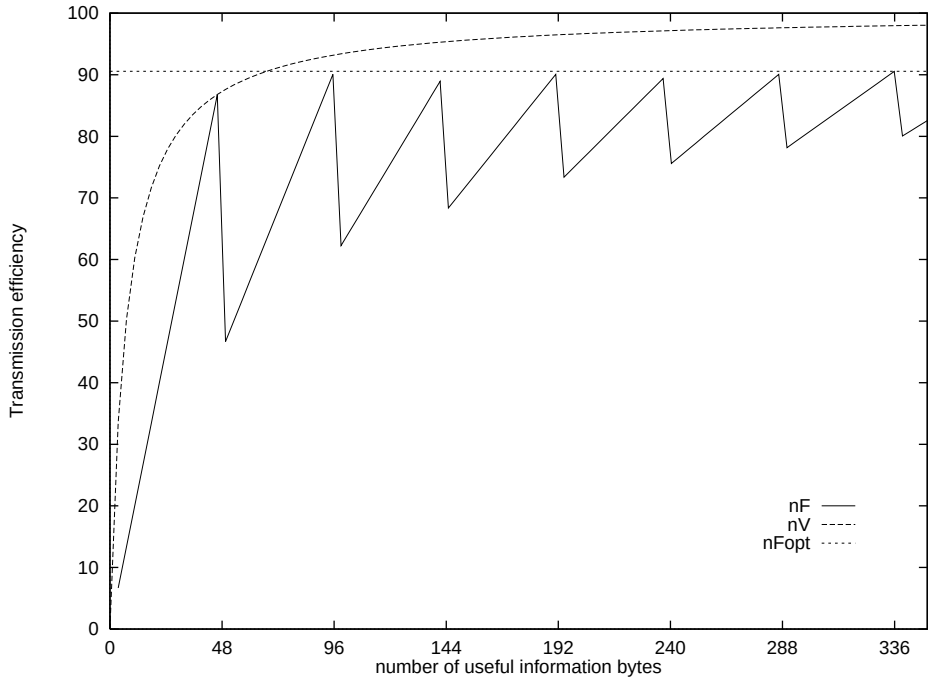

Abbildung 2.3 - Übertragungsoverhead bei variabler und fester Paketlänge [Pry93]

Da in einem Breitbandnetz hauptsächlich Dienste benutzt werden, bei denen große Datenmengen in mehr oder weniger kontinuierlichen Datenströmen transportiert werden sollen (z.B. Videoübertragungen), ist die Effizienz der Ausnutzung der zur Verfügung stehenden Bandbreite annähernd optimal (Abbildung 2.3), egal ob nun eine variable oder eine feste Länge der Pakete benutzt wird.

2.3.1.2 Geschwindigkeit des Switching und Komplexität

Die Komplexität der Implementierung eines Paketswitchers für feste oder variable Paketlänge hängt von den Funktionen, die jeweils ausgeführt werden müssen, ab. Die dabei wichtigsten Faktoren sind die Geschwindigkeit der Operationen und der Speicherbedarf der Warteschlangenverwaltung:

(a) Geschwindigkeit der Operationen

Der wichtigste Faktor für die Geschwindigkeit der Operationen ist die Auswertung des Headers. Angenommen, die Funktionalität des Headers wäre für feste und variable Paketlänge die gleiche, dann hätte man z.B. 2,8 µs (bei 48+5 Byte Paketen und 155 MBit/s) Zeit (siehe [Pry93]), den Header auszuwerten. Bei einer variablen Paketlänge muß dieses auch im “worst case”, d.h.

für das kürzeste Paket, funktionieren. Die Geschwindigkeitsanforderungen an einen Netzwerkswitch wären dann erheblich höher (man hätte nur noch 533 ns bei einem 5+5 Bytes Paket bei 155 MBit/s).

Der nächste für die Geschwindigkeit der Operationen zu beachtende Punkt ist die Verwaltung der Warteschlangen. Bei einer festen Paketgröße ist das Allozieren von Speicherblöcken und deren Freigabe sehr einfach und darum sehr schnell zu implementieren. Bei variabler Paketlänge wird diese Verwaltung sehr komplex, da diverse zeitaufwendige Algorithmen eingebaut werden müssen (*Find best fit*, *Find first fit*, *Garbage collection*, etc.).

(b) Speicherbedarf der Warteschlangenverwaltung

Bei Paketen einer festen Länge hängt der Speicherbedarf von der Auslastung und der akzeptierten Paketverlustrate ab. Das läßt sich einfach modellieren und aus diesem Modell berechnen. Bei Paketen variabler Länge ist dies sehr viel komplizierter, da der Bedarf in diesem Fall von der Paketlängenverteilung abhängt. Die einfachste Regel für die Dimensionierung der Warteschlangen ist der „worst case“, d.h. das längste mögliche Paket. Mit dieser Lösung sind die Speicherbedürfnisse sehr viel größer als bei einer festen Paketlänge.

2.3.1.3 Delay

Um das Delay bei der Übertragung so niedrig wie möglich zu halten, ist die Paketgröße zu minimieren. Vor allem für Echtzeitübertragungen, wie Sprachübertragung, kann das Delay bei zu großer Paketlänge zu groß werden. So kommt es z.B. bei Audioübertragungen bei zu hohen Delays zu Echoeffekten, die die Qualität der Übertragung extrem einschränken.

2.3.1.4 Ergebnisse der Gegenüberstellung

Die wichtigste Anwendung eines Breitbandnetzes ist die Übertragung von Sprache, Video und sonstigen Daten, wie z.B. Filetransfers. Der Vorteil einer variablen Paketlänge ist sehr viel kleiner als die Vorteile einer festen Paketlänge, was die Komplexität und Geschwindigkeit des Switching und der Warteschlangenverwaltung betrifft. Aus diesem Grund haben sich die Experten bei der CCITT 1988 für eine feste Paketlänge für ATM entschieden und gleichzeitig den Begriff “Zelle” eingeführt.

2.3.2 Größe einer ATM Zelle

Die wesentlichen Aspekte bei der Festlegung der Größe einer ATM Zelle sind die Übertragungseffizienz, das Delay und die Komplexität der Implementierung. Die wichtigsten Punkte zur Effizienz wurden schon in Kapitel 2.3.1.1 (Effizienz der Bandbreitenausnutzung) und zum Delay in Kapitel 2.3.1.3 (Delay) erwähnt.

Die eigentliche Komplexität der Implementierung besteht in der Verwaltung und dem Speicherbedarf der Warteschlangen in den Switches. Es ist einfach zu sehen, daß der Speicherbedarf ansteigt, je größer die Zelle ist. Auf der anderen Seite muß natürlich auch der Header ausgewertet werden, was um so mehr Leistung von den Netzwerkswitches verlangt, je kürzer ein Paket ist.

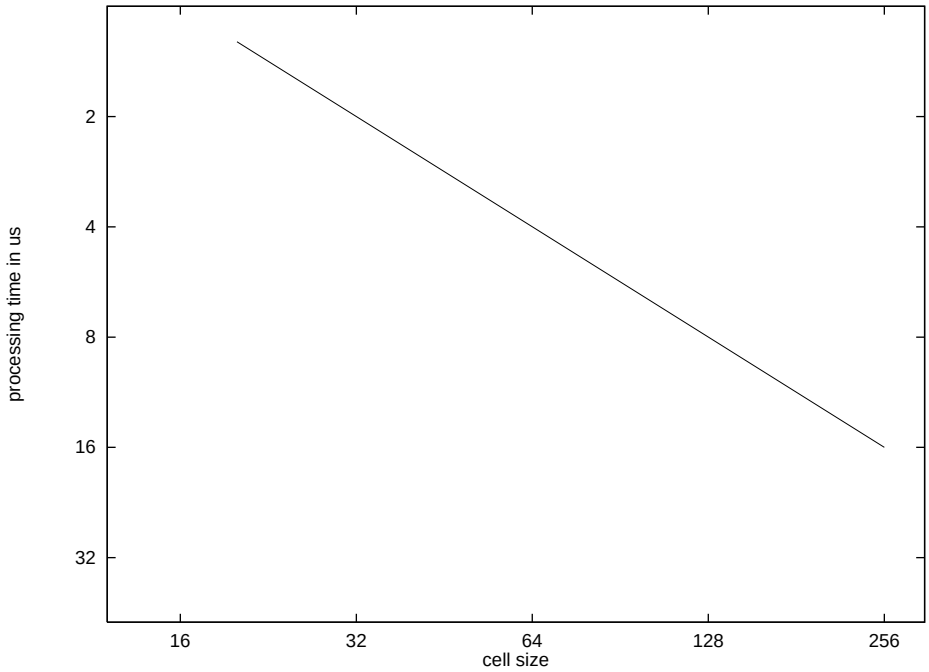

Abbildung 2.4 - Leistung des Switches vs Zellgröße [Pry93]

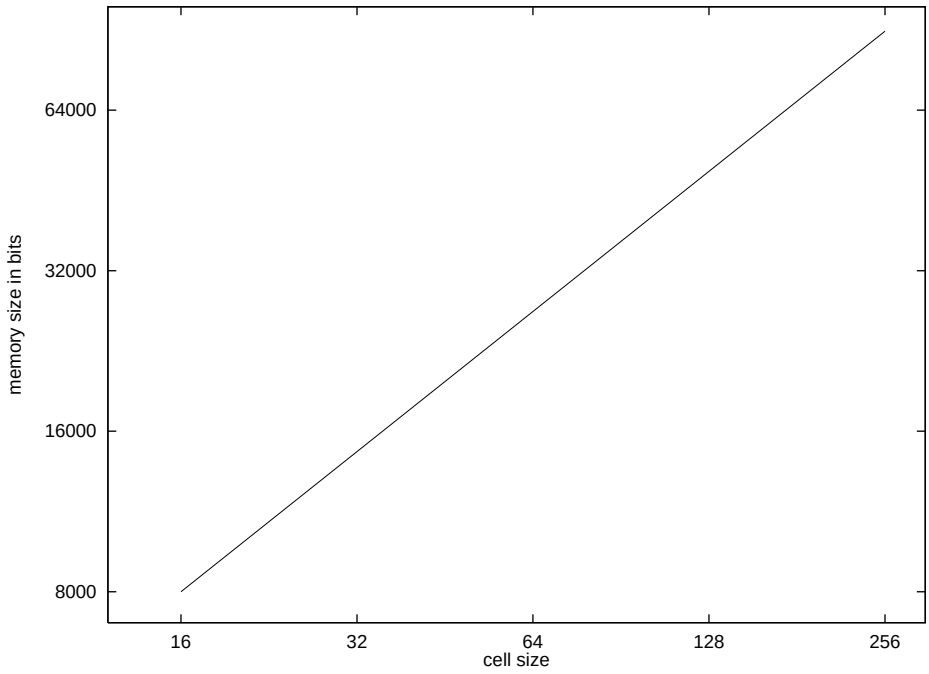

Abbildung 2.5 - Speicherbedarf vs Zellgröße [Pry93]

1989 entschied sich die CCITT 1989 für eine Zellgröße von 53 Bytes.

2.4 Headerfunktionalität

Im folgenden werden Funktionalitäten beschrieben, die durch Parameter im Header erbracht werden müssen, im Hinblick auf die verbindungsorientierte Arbeitsweise von ATM, Dienstgütevoraussetzungen verschiedener Dienste und die Wartung des Netzes.

2.4.1 Virtuelle Verbindungen

Die Funktionalität im Header wurde bei ATM auf ein absolutes Minimum reduziert. Einträge wie Quell- und Zieladressen und Sequenznummern, die man in verbindungslosen Netzwerken braucht, sucht man hier vergeblich, da sie nicht benötigt werden. Jede virtuelle Verbindung wird durch eine Nummer identifiziert, welche nur eine lokale Bedeutung auf einem Stück der Verbindung hat. Weiterhin existiert keinerlei Fehlerkontrolle für den Informationsteil auf Link-to-Link oder auf End-to-End Basis. Auf diese Funktionen kann man aufgrund der hohen Qualität des Übertragungsmediums verzichten.

Die wichtigste verbleibende Funktion des Headers ist die Identifikation der virtuellen Verbindung, was durch zwei Parameter geschieht: den VCI (*Virtual Channel Identifier*) und den VPI (*Virtual Path Identifier*).

2.4.2 Virtuelle Kanäle

Virtuelle Kanäle werden durch den VCI-Eintrag im Header spezifiziert. In den kommenden Breitbandnetzen, die Hunderte von MBit/s transportieren können, wobei ein virtueller Kanal u.U. lediglich einige kBit/s benutzt, wird es einige zehntausend virtuelle Kanäle simultan nebeneinander geben. Darum wird ein VCI Feld von 16 Bit benötigt, d.h. es sind 65536 virtuelle Kanäle gleichzeitig möglich. Der VCI wird beim Verbindungsaufbau zugeordnet und hat nur eine lokale Bedeutung. Über das Netz hinweg wird er jeweils im Switch übersetzt. Wird eine Verbindung geschlossen, so werden alle von dieser Verbindung genutzten VCIs wieder frei und können erneut vergeben werden.

Ein interessanter Vorteil dieses Prinzips ist die Möglichkeit der Nutzung von mehreren VCIs für einen Dienst. So kann z.B. bei der Bildtelefonie Spache, Bild und Daten auf je einem eigenen VCI mit unterschiedlichen Dienstgütemerkmalen transportiert werden. Außerdem können so einfach Komponenten zu- oder weggeschaltet werden. Auch alle Signalling-Informationen (siehe Kapitel 3.1.4 - Signalling) werden über einen eigenen VCI übertragen.

2.4.3 Virtuelle Pfade

Weiterhin ist es im künftigen B-ISDN vorgesehen feste Wege im Netz zu definieren, die eine große Anzahl von gleichzeitigen Verbindungen transportieren können. Dieses Konzept ist auch als virtueller Pfad oder als virtuelles Netzwerk bekannt und wird genutzt, um die verfügbaren Netzwerkressourcen effizient und einfach zu verwalten. Um dieses virtuelle Netzwerk zu unterstützen wurde ein weiteres Feld in den Header einer ATM Zelle eingebaut: der VPI.

Das VPI Feld ist 8 oder 12 Bit lang, so daß 256 bzw. 4096 virtuelle Pfade mit je bis zu 65536 virtuellen Kanälen erzeugt werden können.

2.4.4 Prioritäten

Eine weitere Funktion im Header ist die Unterstützung von Prioritäten. Diese dienen im Falle einer Netzüberlastung dazu, daß nur auf niedrig priorisierten Verbindungen Datenverluste auftreten, während andere, “wichtigere” Verbindungen ohne Beeinträchtigung weiterlaufen.

Es existieren zwei Arten von Prioritäten: Zeit- und semantische Priorität. In einem System mit Zeitprioritäten können sich manche Zellen länger aufhalten, als andere. In einem System mit semantischer Priorität hingegen haben manche Zellen eine höhere Wahrscheinlichkeit, verloren zu gehen, d.h. wir haben Zellen/Verbindungen mit unterschiedlicher semantischer Transparenz.

Prioritäten können sowohl für Verbindungen (auf VCI oder VPI Basis) oder auch nur für einzelne Zellen vergeben werden. In ATM wurde dem Header einer ATM-Zelle ein weiteres Feld zur Steuerung der Prioritäten angelegt.

2.4.5 Überwachung und Wartung

Zur Überwachung und Wartung (*Maintenance*) des Netzwerkes und zur Feststellung der Geschwindigkeit von ATM Verbindungen wurden einige weitere Bits in den Header eingefügt, um zwischen normalem Datenverkehr und Wartungsverkehr zu unterscheiden. Diese Methode wurde PTI (*Payload Type Identification*) genannt, da der Payload Anteil einer Zelle (also der Informationsteil) verschiedene Arten von Information beinhalten kann (Nutzdaten, Maintenance, etc.). Dadurch ist es möglich, spezielle Zellen in eine virtuelle Verbindung einzuspeisen, die wie normale Zellen durch das Netz gelenkt werden, aber z.B. Qualitätsmessungen für die Geräte zwischen Einspeisung der Zellen in das Netz und Herauslösung dieser Zellen aus dem Netz dienen.

Abhängig von den von ATM unterstützten Wartungsfunktionen sind 0 bis 2 Maintenance Bits im Header definiert.

2.4.6 Mehrfacher Zugriff

Um mehrere Terminals (Benutzer) mit einer physikalischen Schnittstelle zu verbinden (Mehrfacher Zugriff - *Multiple Access*), gilt es, ein point-to-multipoint Protokoll zu definieren, das die gleichzeitige Nutzung ein und desselben Links ermöglicht. Um diese Funktionalität abzubilden, werden wieder einige Bits im Header benötigt. Allerdings gibt es auch einige point-to-multipoint Algorithmen, die dies ohne zusätzliche Bits können, wie z.B. *Register-Insertion Mechanism*. Die Anzahl der benötigten Bits hängt im wesentlichen von den von der MAC (*Medium Access Control*) Schicht zur Verfügung gestellten Mechanismen ab und liegt zwischen 0 und 8 Bits.

2.4.7 Header-Fehlerkorrektur

Wie in Kapitel 2.2.2 beschrieben, ist der interessanteste Algorithmus zur Fehlererkennung und -korrektur ein adaptiver Algorithmus, der einzelne Fehler korrigieren kann und gehäuft auftretende Fehler zumindest erkennt. Um den Header zu schützen, ist es angebracht, ein auf dem Hamming Code basierendes Verfahren zu benutzen. In diesem Fall wurde der BCH Code (Bose-Chadhuri-Hocquenghem) gewählt. Dieser Algorithmus kann Bitfehler wie aus folgender Tabelle ersichtlich korrigieren (Daten aus [Pry93]):

n **Gesamtanzahl von Bits**	**k** **nutzbare Bits**	**t** **korrigierbare Bits**
31	26 21 16	1 2 3
63	57 51 45	1 2 3
127	120 113 106	1 2 3

Tabelle 2.1 - BCH Codes

Es ergeben sich folgende Wahrscheinlichkeiten, mit einer bestimmten Anzahl von Kodierungsbits eine bestimmte Anzahl von Bits zu schützen (Daten aus [Pry93]):

Codebits **zu schützende Bits**	6	7	8
32	48%	74%	89%
40	36%	68%	84%
48	23%	62%	81%

Tabelle 2.2 - mögliche Fehlererkennung für eine Ein-Bit-Fehlerkorrektur

Wie man sieht, ist eine Länge von 8 Bit für die Fehlerkontrolle im Header ein annehmbarer Wert. Unter diesen Voraussetzungen und unter Beachtung der Anforderungen an die ATM-Headergröße von 2-5 Byte hat die CCITT folgende Header standardisiert:

Funktion	**benötigte Bits**	**CCITT NNI/UNI**
virtueller Kanal (VCI)	8-16	16
virtueller Pfad (VPI)	8-12	12/8
Prioritäten	0-4	1
Maintenance/Payload Typ	0-2	2
Point-to-Multipoint	0-8	0/4
Headerfehlerkontrolle (HEC)	0-8	8
Reserviert	0-6	1
Gesamtanzahl der Bits	16-56	40

Tabelle 2.3 - Headerfunktionalität und benötigte Größe

3 ATM Standards

In diesem Kapitel werden die grundlegenden von der CCITT und vom ATM Forum entworfenen bzw. von der CCITT standardisierten Prinzipien von ATM, wie z.B. die Adressierung, gezeigt. Außerdem wird auf das ATM Schichtenmodell, mit besonderer Berücksichtigung der für diese Arbeit benötigten Schichten, eingegangen.

3.1 Prinzipien

In diesem Abschnitt folgt eine kurze Aufzählung der Prinzipien von ATM, wobei das Signalling (der Mechanismus zum Aufbauen einer Verbindung durch das ATM Netzwerk, durch Aushandeln der Verbindungsparameter und Aufbau der entsprechenden VCs) detaillierter beschrieben wird.

3.1.1 Informationsübertragung

ATM ist ein paketorientierter Übertragungsmodus basierend auf einem asynchronen Zeitmultiplexverfahren. Jede ATM-Zelle hat eine feste Länge und besteht aus einem Header und einem Informationsteil. Über ein ATM-Netzwerk kann Information jeder Art (Sprache, Video, Nutzdaten, etc.) transparent transportiert werden, es wird jedoch keinerlei Fehlerkorrektur zur Verfügung gestellt.

3.1.2 Routing

ATM arbeitet verbindungsorientiert. Die VPIs/VCIs sind für die gesamte Zeit einer Verbindung festgelegt und werden übersetzt, wenn eine Zelle von einem System in ein anderes “geswitched” wird. Signalling und andere Benutzerinformationen werden über spezielle virtuelle Kanäle transportiert.

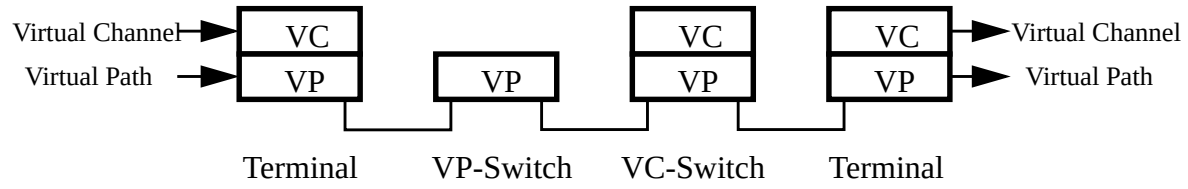

Abbildung 3.1 - VC- und VP-Verbindungen [Pry93]

Es existieren zwei verschiedene Arten virtueller Verbindungen: Verbindungen über virtuelle Kanäle - VCCs (*Virtual Channel Connections*) und solche über virtuelle Pfade - VPCs (*Virtual Path Connections*). Dabei können mehrere VCCs über einen VPC gehen. Das Switching der Zellen richtet sich immer zuerst nach dem VP und erst dann nach dem VC, wobei es auch reine VP-Switches geben kann (siehe Abbildung 3.1)

3.1.3 Ressourcen

ATM arbeitet verbindungsorientiert. Für Datenübertragungen werden entweder VPCs (*Virtual Path Connection*), die wiederum mehrere VCCs (*Virtual Channel Connection*) transportieren können, oder VCCs aufgebaut. Dies macht es nötig, daß für diese Verbindung ein VPI und im Falle eines VCC noch ein VCI alloziert werden muß. Diese nennt man auch *ATM Cell Identifier.* Zu diesen Identifikatoren gehört auch noch der PTI (*Payload Type Identifier*). Die *ATM Cell Identifier*, welche z.T. in der ATM-Schicht vorbesetzt werden, dienen als Voraussetzung für die Kommunikation über das Netzwerk und ermöglichen ein Netzmanagement.

Eine weitere, für jede virtuelle Verbindung reservierbare Ressource ist die Bandbreite. Es werden Möglichkeiten zur Sicherung der Ressourcen beim Multiplexing von mehreren Verbindungen mit variabler Zellrate (VBR - *Variable Bit Rate*) bereitgestellt. Die z.Z. einzige von der CCITT standardisierte Möglichkeit dazu ist die *Peak Cell Rate* (PCR), womit eine Kontrolle des Bandbreitenbedarfs einer VBR-Verbindung ermöglicht wird. Weitere Möglichkeiten wurden und werden im ATM Forum besprochen und für eine Standardisierung seitens der CCITT vorbereitet.

Durch verschiedene Parameter ist die *Quality of Service* (QoS) gegeben. Dazu gehören Werte wie die Zellverlustrate, das Delay und der Jitter. Diese Werte sind z.B. über die reservierte Bandbreite oder die Zellverlustpriorität (CLP - *Cell Loss Priority*) steuerbar.

3.1.4 Signalling

Das Aushandeln der Verbindungsparameter (VPI/VCI, Durchsatz, QoS) zwischen Benutzer und Netzwerk erfolgt über einen separaten virtuellen Kanal durch das sogenannte Signalling. Das Signalling Protokoll ist in der CCITT Empfehlung Q.93B festgelegt.

3.1.4.1 ATM-Adressierung

Im Vergleich zu anderen Netzwerken ist eine ATM-Adresse mit 20 Byte ziemlich lang. IP benutzt z.B. nur 4 Byte. Es gibt 3 verschiedene Möglichkeiten, eine private ATM Adresse zu kodieren:

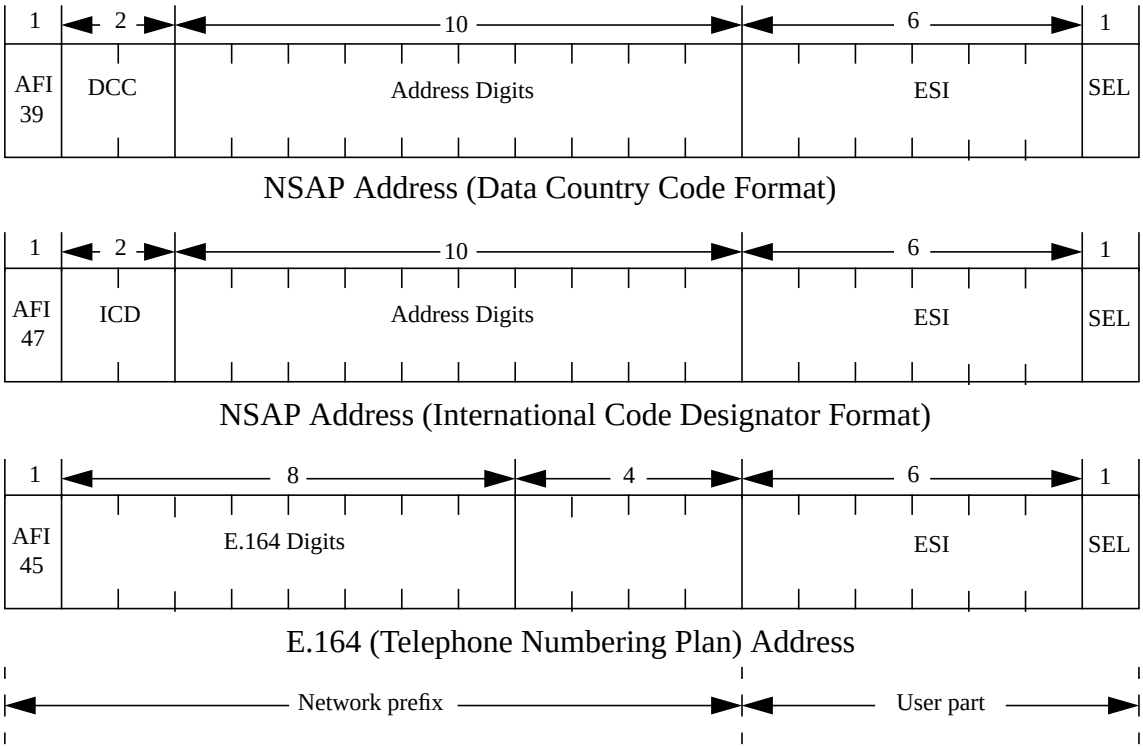

Abbildung 3.2 - ATM-Adressierungsschemata [BSTS93-1]

Die ersten 13 Byte der Adresse bezeichnen das Netzwerk, wobei der *Authority and Format Identifier* (AFI) im ersten Byte angibt, welches Adressierungsschema benutzt wird. Alle 3 Formate benutzen dieselbe Kodierung im Trailer. Der *End System Identifier* (ESI) kennzeichnet eindeutig das Endsystem. Typischerweise kann hier die 48 Bit lange MAC-Adresse benutzt werden. Das letzte Byte ist ein Selektorfeld, welches nicht für das Routing verwendet wird, aber eine bestimmte Anwendung auf dem Endsystem identifizieren kann.

Die NSAP-Adressen (*Network Service Access Point*) sind hierarchisch organisiert, um das Routing zu vereinfachen. Dieses ähnelt dem Subnetzkonzept in IP-Netzwerken. Führende Stellen in der NSAP-Adresse referenzieren logische Subnetze, und bei ankommenden Adressen wird in Routingtabellen ein durch ein Längenfeld spezifizierter Anteil des NSAP verglichen, um eine passende Direktive zu finden.

Wenn das ATM Netz gestartet wird, muß jede Station seine ATM-Adresse kennen. Für lokale Netze hat das ATM-Forum einige Prozeduren über das UNI (*User Network Interface*) für Endstationen definiert, um ihre Netzwerkadresse vom Netz zu erfahren. Das *Interim Local Management Interface* (ILMI) Protokoll wird benutzt, um Konfigurations- und Managementinformationen durch das UNI über einen speziellen virtuellen Kanal (VPI=0, VCI=16) zu übertragen. Dieses Protokoll benutzt das *Simple Network Management Protocol* (SNMP) mit einer, das

ATM-UNI beschreibenden *Management Information Base* (MIB). Dabei wird erwartet, daß jedes Endsystem (ATM Karte, ATM-LAN Bridge, etc.) seinen ESI kennt. Der lokale ATM-Switch muß die restlichen Adreßkomponenten für alle angeschlossenen ATM-Systeme kennen.

Der Adreßregistrierungsprozeß wird jedesmal gestartet, wenn das Netzwerk initialisiert, eine Station angeschlossen oder eine Station vom Netz abgehängt wird.

3.1.4.2 Verbindungsaufbau

Eine Verbindung wird durch eine SETUP-Message, die durch das *User Network Interface* geschickt wird, initiiert. Diese Message beinhaltet folgende die Verbindung beschreibende Elemente:

- *ATM User Cell Rate* - spezifiziert die gewünschte Bandbreite in Zellen pro Sekunde
- *Broadband Bearer Capability* - spezifiziert den verbindungsorientierten Trägerservice, wie z.B. Class X (siehe Kapitel 3.2.3.1 - Klassen und Typen der AAL)
- *Broadband Low Layer Information* - spezifiziert Link- und Netzwerkprotokollinformationen, wie LLC/SNAP für gebridgete LAN Protokolle
- *Called Party Number* - ATM-Zieladresse
- *Called Party Subaddress* - private ATM-Adresse (NSAP) des Ziels, falls nur E.164-Adressierung unterstützt wird
- *Calling Party Number* - ATM-Quelladresse
- *Calling Party Subaddress* - private ATM-Adresse (NSAP) der Quelle
- *Quality of Service* - spezifiziert die gewünschte QoS-Klasse der Verbindung
- *Transit Network Selection* - identifiziert das gewünschte Trägernetzwerk für die Verbindung (z.B. AT&T, MCI, Sprint)

Wenn die SETUP-Message vom Netzwerk empfangen wurde, wird mit einer CALL PROCEEDING-Message geantwortet, die das Endsystem informiert, welcher VC (VPI/VCI) benutzt wird. Wenn das SETUP den Empfänger erreicht, hat dieser alle, die gewünschte Verbindung beschreibenden Elemente und die vom Netzwerk gewählten VPI/VCI Werte zur Verfügung, um zu entscheiden, ob er die Verbindung annehmen will oder nicht. Wird eine Verbindung gewünscht, antwortet der Empfänger mit einer CONNECT-Message.

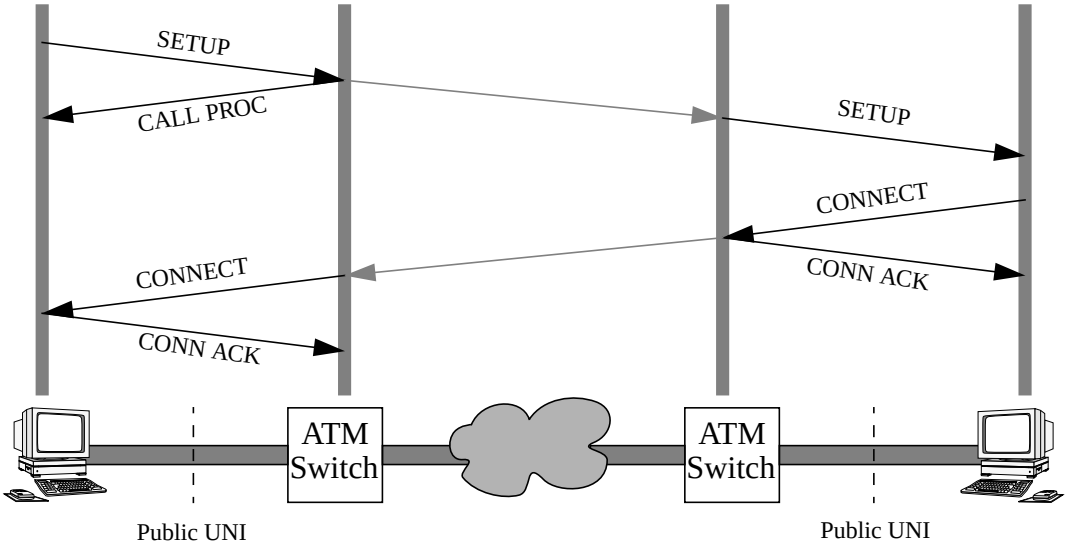

Abbildung 3.3 - Nachrichtenaustausch beim Verbindungsaufbau

3.1.4.3 Ablehnung des Verbindungsaufbauwunsches

Es kann natürlich sein, daß eine Verbindung nicht wie angefordert aufgebaut werden kann. Dies ist u.a. der Fall, wenn die Zieladresse nicht gefunden werden konnte oder wenn nicht genügend freie Ressourcen für die gewünschte Bandbreite und/oder die QoS verfügbar sind.

Die Verbindung kann an verschiedenen Punkten im Netz abgelehnt werden. Auf Benutzerebene kann ein Switch oder der Empfänger des SETUP dieses mit einer RELEASE COMPLETE-Message beantworten, wenn er merkt, daß die Verbindung nicht aufgebaut werden kann. Dabei wird der Grund, warum die Verbindung abgelehnt wurde, im RELEASE COMPLETE-Paket angegeben.

Im Netzwerk kann es z.B. zu Routingproblemen kommen. Wenn ein Switch eine solche Situation erkennt, lehnt er die Verbindung ebenfalls mit einem RELEASE COMPLETE ab.

3.1.4.4 Verbindungsabbau

Eine stehende Verbindung kann von beiden Seiten abgebaut werden. Gründe dafür sind z.B. das Beenden einer Applikation oder Fehlersituationen im Netzwerk. Wird die Verbindung von einem Benutzer abgebaut, so sendet dieser eine RELEASE-Nachricht an den Kommunikationspartner, die dieser mit einer RELEASE COMPLETE-Message beantwortet. Die RELEASE-Nachrichten beinhalten einen Parameter, der den Grund des Verbindungsabbaus erklärt.

3.1.4.5 Interoperabilität

Das Signalling-Protokoll ist ein sehr komplexes Protokoll. Vor allem die Interaktionen im Netzwerk haben eine hohe Komplexität, die sich im Entwurf und in der Implementierung von Signalling-Systemen widerspiegelt. Die größten Probleme entstehen beim Aufbau eines Testnetzes, wenn die Geräte nicht sofort richtig miteinander kommunizieren. Interoperabilitätsprobleme resultieren hauptsächlich aus sich unterscheidenden Implementationen verschiedener Hersteller. Auch ist das Verständnis des Standards nicht immer das gleiche, da die meisten Spezifikationen in englischer Sprache, statt in einer exakteren Form, wie z.B. “Statusdiagramme” geschrieben sind. Weiterhin kann es Fehler in der Implementierung geben, wie z.B.:

- Es werden keine Bestätigungspakete verschickt, wodurch eine Seite für immer auf eine Nachricht wartet, die nie ankommen wird.
- Protokolltimer sind zu niedrig eingestellt, weshalb die sendende Seite einen Timeout erhält, bevor die Antwort ankommt.
- Die Sequenznummern in den Bestätigungspaketen sind falsch, wodurch die falschen Nachrichten im Speicher gelöscht werden.

Netzprobleme sind meistens auf falsch konfigurierte Switches und Endsysteme zurückzuführen (z.B. wenn zwei Switches dieselbe NSAP-Adresse gegeben wurde oder zwei Endsysteme dieselben ESIs haben).

3.2 Schichtenmodell

Das OSI[1]-Modell der ISO[2] ist ein sehr bekanntes Modell für alle Arten von Kommunikationssystemen. Eine ähnliche logische, hierarchische Struktur wurde auch für ATM genutzt, auch wenn die CCITT keine Verbindungen zwischen dem ATM- und dem OSI-Modell aufzeichnet. Das ATM-Modell hat die in Abbildung 3.4 gezeigte Struktur.

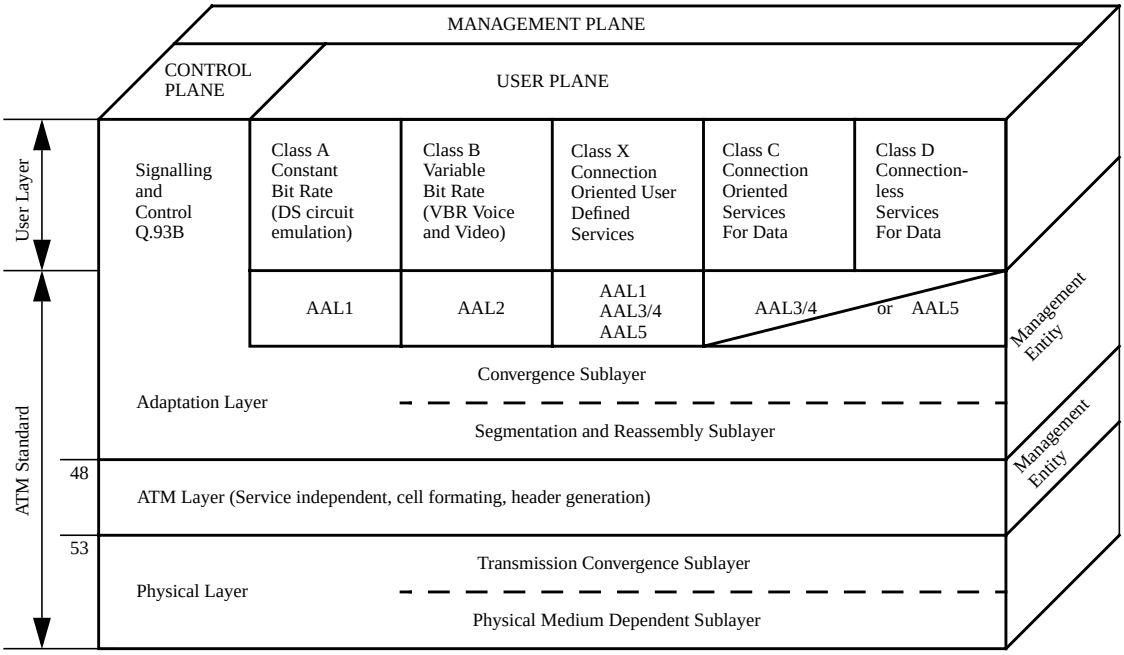

Abbildung 3.4 - ATM-Modell [Teke94]

3.2.1 Physikalische Schicht

Die physikalische Schicht besteht aus zwei Teilschichten: der *Physical Medium Sublayer* und der *Transmission Convergence Sublayer*:

(1) Physical Medium Sublayer

Dieser Teil der physikalischen Schicht ist für die korrekte Übertragung und den Empfang von Bits über ein geeignetes physikalisches Medium verantwortlich. Auf der untersten Ebene ist diese Funktion abhängig vom verwendeten Medium (optisch, elektrisch, etc.). Weiterhin ist die-

1. *Open Systems Interconnection*
2. *International Standards Organisation*

se Schicht für das *bit timing*, d.h. das richtige Takten der Datenbits, und einen geeigneten *Line Coding Algorithm*, wie z.B. den Manchestercode, zuständig. Ein Beispiel für diese Schicht ist CCITT-Empfehlung G.703[3].

(2) Transmission Convergence Sublayer

Diese Unterschicht hat hauptsächlich die folgenden 5 Aufgaben:

- Generierung und Wiederherstellung der Übertragungsrahmen (*transmission frame generation/recovery*)
- Anpassung dieser Rahmen (*transmission frame adaptation*)
- Erkennung der Zellgrenzen (*cell delineation*)
- Generierung und Überprüfung des HEC[4]-Feldes (*HEC header sequence generation/verification*)
- Einfügen und Unterdrücken von zusätzlichen Zellen zum Einhalten einer bestimmten Zellrate (*cell rate uncoupling*)

3.2.2 ATM-Schicht

3.2.2.1 Aufgaben

Die ATM Schicht ist vollständig abhängig vom physikalischen Medium, welches zum Transport der Zellen benutzt wird. Die folgenden Funktionen sind in dieser Schicht definiert:

- Multiplexen und Demultiplexen von Zellen verschiedener Verbindungen (identifiziert durch unterschiedliche VPI/VCI Werte) zu einem einzelnen Strom von Zellen auf der physikalischen Schicht.
- Übersetzen der Zellidentifikatoren beim Switchen von Zellen von einem physikalischen Link auf einen anderen. Dabei wird entweder nur der VPI geändert, oder sowohl VPI als auch VCI.
- Generieren/Extrahieren des Zellheaders nachdem/bevor die Zellen von dem/an den *ATM adaptation layer* weitergeleitet worden sind/werden.
- Implementieren eines Flußkontrollmechanismus auf dem *User Network Interface* (UNI) durch die GFC[5]-Bits im Header

---

3. G.703 - *Physical/electrical Characteristics of Hierarchical Digital Interfaces*
4. *Header Error Control*
5. *Generic Flow Control*

3.2.2.2 Header-Struktur

Nach der CCITT-Empfehlung besteht eine ATM-Zelle aus einem 48 Byte langen Informationsteil und einem 5 Byte langen Header. Auf dem *User Network Interface* hat der Header die in Abbildung 3.5 gezeigte Struktur. In der gleichen Abbildung wird auch die Headerstruktur auf dem *Network-Node Interface* (NNI) gezeigt. Diese beiden Formate sind bis auf das GFC-Feld auf dem UNI, welches auf dem NNI durch 4 weitere VPI-Bits ersetzt wird, identisch.

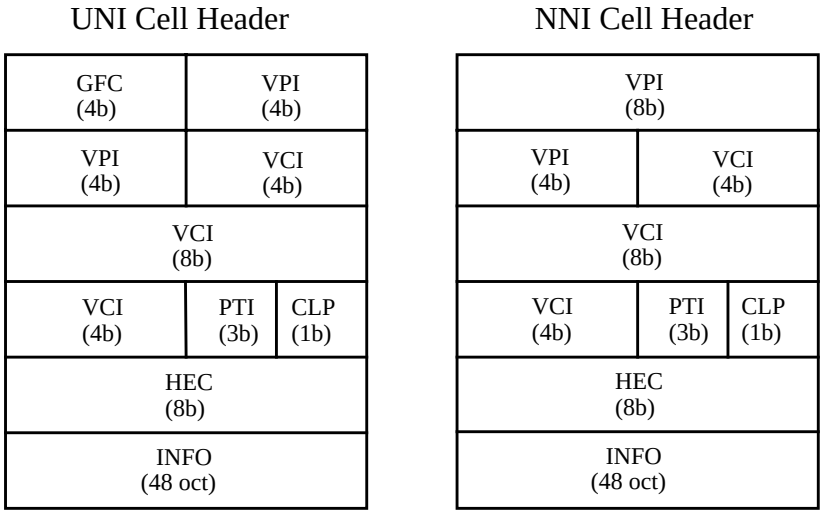

Abbildung 3.5 - ATM-Zell-Struktur [Pry93]

Dabei gibt es einige vorbesetzte Headerwerte, die für spezielle Zellen reserviert sind. Im einzelnen werden die folgenden Zellen unterschieden (Tabellen aus [Forum94]):

Use	**Value**[1, 2, 3, 4]			
	Octet 1	**Octet 2**	**Octet 3**	**Octet 4**
Unassigned cell indication	00000000	00000000	00000000	0000xxx0
Meta-signalling (default)[5, 7]	00000000	00000000	00000000	00010a0c
Meta-signalling[6, 7]	0000yyyy	yyyy0000	00000000	00010a0c
General broadcast signalling (default)[5]	00000000	00000000	00000000	00100aac
General broadcast signalling[6]	0000yyyy	yyyy0000	00000000	00100aac
Point-to-point signalling (default)[5]	00000000	00000000	00000000	01010aac
Point-to-point signalling[6]	0000yyyy	yyyy0000	00000000	01010aac
Invalid Pattern	xxxx0000	00000000	00000000	0000xxx1

Tabelle 3.1 - Durch die CCITT vorbesetzte Werte für den Zellheader

Use	Value[1, 2, 3, 4]			
	Octet 1	Octet 2	Octet 3	Octet 4
Segment OAM flow F4 cell[7]	0000aaaa	aaaa0000	00000000	00110a0a
End-to-end OAM flow F4 cell[7]	0000aaaa	aaaa0000	00000000	01000a0a

Tabelle 3.1 - Durch die CCITT vorbesetzte Werte für den Zellheader

1. “a” zeigt an, daß das Bit für die entsprechende Funktion der ATM Schicht frei verfügbar ist.
2. “x” zeigt ‘don’t care’ Bits an.
3. “y” gibt einen beliebigen VPI Wert ungleich 000000 an.
4. “c” zeigt an, daß die rufende Signalling Einheit das CLP Bit auf 0 setzen soll. Dieser CLP-Wert kann durch das Netzwerk verändert werden.
5. Reserviert für das User Signanalling mit lokalem Austausch.
6. Reserviert für Signalling mit anderen Signalling Einheiten (wie andere User oder entfernte Netzwerke)
7. Die sendende ATM Einheit soll das Bit 2 von Octet 4 auf Null setzen. Die empfangende Einheit soll genua dieses Bit ignorieren.

Use	Value[1]			
	Octet 1	Octet 2	Octet 3	Octet 4
Carriage of ILMI message[2]	00000000	00000000	00000001	0000aaa0

Tabelle 3.2 - Durch das ATM-Forum vorbesetzte Werte für den Zellheader

1. “a” zeigt an, daß das Bit für die entsprechende Funktion der ATM Schicht frei verfügbar ist.
2. Die sendende ATM Einheit soll das CLP Bit auf 0 setzen. Die empfangende Einheit soll ILMI Zellen mit CLP=1 wie ILMI Zellen und wie alle anderen CLP=1 Zellen bearbeiten.

3.2.3 ATM Adaptation Layer (AAL)

Diese Schicht erweitert die Funktionalität der ATM-Schicht so, wie sie von der nächsthöheren Schicht, der Benutzerschicht (*user layer*), erwartet wird. Die Funktionen, die von der AAL zur Verfügung gestellt werden, sind stark von den Anforderungen der nächsten Schicht abhängig. Der A*TM Adaptation Layer* ist in zwei Teilschichten unterteilt:

(1) Segmentation And Reassembly Sublayer (SAR)

Die Hauptaufgabe des *Segmentation And Reassembly Sublayer* (SAR) ist die Segmentierung der von der höheren Schicht ankommenden Information in eine Größe, die für den Payloadanteil der entsprechenden ATM-Verbindung passend ist.

(2) Convergence Sublayer (CS)

Der *Convergence Sublayer* (CS) übernimmt die Aufgabe der Identifikation der Nachrichten und außerdem die der Zeitwiederherstellung. Für manche AAL-Typen ist der CS in den *Common Part Convergence Sublayer* (CPCS) und den *Service Specific Convergence Sublayer* (SSCS) unterteilt.

3.2.3.1 Klassen und Typen der AAL

Der AAL ist in fünf Klassen (AAL1 bis AAL5) für die unterschiedlichsten Anwendungsfälle unterteilt. Dabei werden außerdem fünf Dienste (Class A, B, C, D, X) unterschieden. Die Eigenschaften der Klassen und Dienste ist aus den folgenden beiden Tabellen (aus [Pry[93]) ersichtlich.

	Class A	**Class B**	**Class X**	**Class C**	**Class D**
End-to-End Timing	Required		User Defined	Not Required	
Bit Rate	Constant	Variable	User Defined	Variable	
Connection Mode	Connection-oriented				Connectionless

Tabelle 3.3 - Diensteklassifikationen des ATM Adaptation Layer

	AAL1	**AAL2**	**AAL3**	**AAL4**	**AAL5**
Time relation between source and destination	Required	Required	Not Required	Not Required	Not Required
Bit rate	Constant	Variable	Variable	Variable	Variable
Connection Mode	Connection Oriented	Connection Oriented	Connection Oriented	Connec-tionless	Connection Oriented

Tabelle 3.4 - Diensteklassen des ATM AAL

3.2.3.2 AAL1

Constant Bit Rate (CBR) Dienste benötigen eine konstante Bitrate beim Transport von Informationen. Dazu stellt AAL1 folgende Funktionen bereit:

- Transport von *Service Data Units* (SDU) mit einer konstanten Senderate und deren Auslieferung mit derselben Rate

- Transport von Timinginformationen zwischen den Sendepartnern
- Transport von Informationen über Datenstrukturen
- Anzeige von fehlerhafter oder verloren gegangener Information, wenn sie nicht schon korrigiert wurde

Typische Anwendungen von AAL1 sind:

- Audio- und Videoübertragungen mit CBR mit sehr hoher Qualität
- Sprachübertragungen

3.2.3.3 AAL2

AAL2 bietet für die Datenübertragung eine variable Bitrate. Zusätzlich wird Timinginformation zwischen den Kommunikationspartnern übertragen.

3.2.3.4 AAL3/4

Die CCITT empfiehlt AAL3/4 für Datenübertragungen, die eine hohe Qualität (keine Datenverluste), aber nicht unbedingt ein kurzes Delay voraussetzen. AAL3/4 kann sowohl für verbindungsorientierte, als auch für verbindungslose Dienste eingesetzt werden.

3.2.3.5 AAL5

Für Anwendungen, die hohe Geschwindigkeiten voraussetzen, und Anwendungen verbindungsorientierter Dienste ist AAL3/4, wie es von der CCITT empfohlen wurde, nicht sonderlich gut geeignet. So ist der Overhead bei AAL3/4 von 4 Byte bei einer PDU-Länge von 48 Byte einfach zu hoch. Außerdem liefert der 10 Bit lange CRC[6] und die 4 Bit lange Sequenznummer nicht die benötigte Sicherheit, vor allem bei langen Datenblöcken. Aus diesen Gründen hat das ATM-Forum einen neuen AAL-Typ entworfen: AAL5. Z.Z. wird AAL5 von der CCITT für Class C Dienste empfohlen.

Die Abbildung 3.6 zeigt den Aufbau einer AAL5 CPCS[7] PDU[8], die für den Datentransport über AAL5 eingesetzt wird. Diese PDU besteht aus maximal 2^{16} Informationsbytes, einem Feld, das nur dazu dient, die gesamte Länge der PDU auf ein vielfaches von 48 Byte (die Länge einer ATM Zelle) zu vergrößern (PAD), einem Längenfeld, einem Kontrollfeld und einer Prüfsumme.

---

6. *Cyclic Redundancy Check*
7. *Common Part Convergence Sublayer*
8. *Protocol Data Unit*

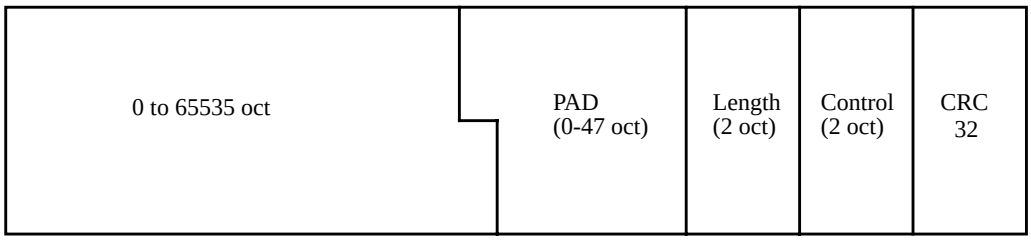

Abbildung 3.6 - AAL5 CPCS PDU [Pry93]

3.2.3.6 Adaption an Signalling

Für das Signalling durch das UNI und das NNI wird ein Service, wie ihn AAL3/4 oder AAL5 bietet, benötigt. Das ATM-Forum spezifizierte AAL5 für diesen Dienst. Bei der CCITT ist man tendenziell auch für die Verwendung von AAL5, aber es gibt noch keinen entsprechenden Standard.

4 Die Programmierschnittstelle der SunATM™-155 Karte

Da z.Z. leider noch kein einheitliches *Application Programmer's Interface* (API) für ATM-Karten standardisiert wurde, hat Sun ein eigenes Interface entworfen, das bis zur Verabschiedung eines Standards durch das ATM-Forum benutzt wird. Leider hat das den Effekt, daß man für die Portierung der Software auf die ATM-Hardware eines anderen Herstellers sehr viele Änderungen am Code eines für die SunATM-Karte entwickelten Programms vornehmen muß. Durch die Implementierung einer möglichst hardwareunabhängigen Schnittstelle soll dies in der vorliegenden Arbeit vermieden werden.

Das API besteht aus zwei Teilen, die jeweils in ein User API und ein Kernel API aufgeteilt werden:

- Q.93B-Treiber für den Einsatz von Signalling
- Treiber für direkten Zugriff auf das ATM Interface

Abbildung 4.1 zeigt den prinzipiellen Aufbau des API der SunATM-Karte:

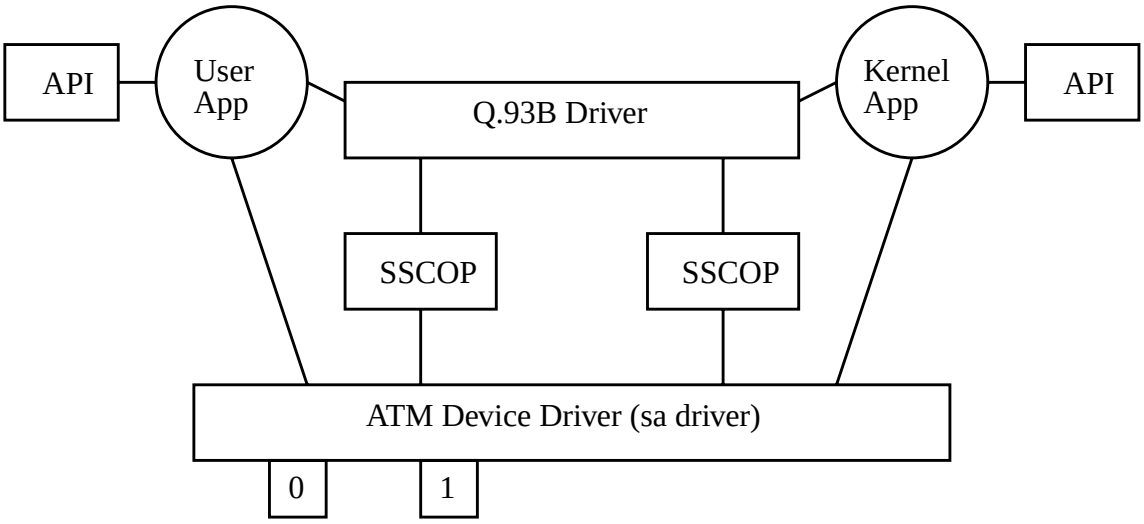

Abbildung 4.1 - Treiberkonfiguration des API[9]

9. Abbildung aus [SunATM]

4.1 Q.93B-Programmierschnittstelle

Alle Kontroll- und Datenblöcke werden über ein System V typisches Streams Modul per putmsg(2)[10] und getmsg(2) zum q93b(7) Device zur Weiterverarbeitung durch den Treiber gesendet bzw. von ihm geholt. Zur Generierung neuer bzw. zum entschlüsseln eingetroffener Kontrollblöcke existieren die folgenden Funktionen (die genaue Syntax ist aus den Manualseiten ersichtlich):

- qcc_bld(3) - Funktionen zum Erzeugen von Signalling-Nachrichten
  - qcc_bld_setup()
  - qcc_bld_call_proceeding()
  - qcc_bld_conect()
  - qcc_bld_release()
  - qcc_bld_release_complete()
  - qcc_bld_status_enquiry()
  - qcc_bld_status()
  - qcc_bld_restart()
  - qcc_bld_restart_ack()
- qcc_len(3) - Funktionen zum Ermitteln der minimalen Puffergröße für die qcc_bld(3) Aufrufe
  - qcc_bld_setup_datalen()
  - qcc_bld_call_proceeding_datalen()
  - qcc_bld_connect_datalen()
  - qcc_bld_connect_ack_datalen()
  - qcc_bld_release_datalen()
  - qcc_bld_release_complete_datalen()
  - qcc_bld_status_enquiry_datalen()
  - qcc_bld_status_datalen()
  - qcc_bld_restart_datalen()
  - qcc_bld_restart_ack_datalen()

10. Die Syntax name(sektion) gibt den Namen eines Befehls, einer Funktion, der Konfigurationsdatei oder des Treibers im UNIX Betriebssystem an, wobei die Sektion die Art des Befehls, der Funktion, der Konfigurationsdatei oder des Treibers angibt. Dabei steht Sektion 1 für Kommandos, Sektion 2 für Systemcalls, Sektion 3 für C-Library-Aufrufe, Sektion 7 für Treiber. Für weitere Informationen über dieses System sollte man ein UNIX Buch zu Rate ziehen.

    - qcc_max_bld_datalen()
    - qcc_ctl_len()
- qcc_parse(3) - Funktionen zum Entschlüsseln und Zerlegen von eingetroffenen Signalling-Nachrichten
    - qcc_parse_setup()
    - qcc_parse_call_proceeding()
    - qcc_parse_connect()
    - qcc_parse_release()
    - qcc_parse_release_complete()
    - qcc_parse_status_enquiry()
    - qcc_parse_status()
    - qcc_parse_restart()
    - qcc_parse_restart_ack()
    - qcc_get_hdr()
- qcc_util(3) - Hilfsfunktionen
    - q_ioc_bind() - Funktion zum Binden eines SAP an einen geöffneten Stream

Leider ist das Q.93B API noch nicht richtig einsetzbar, da zwar SVCs via Signalling aufgebaut, aber leider keine Daten übertragen werden konnten. Dieses Problem gibt es sowohl mit der Version 1.0, als auch mit Version 2.0b der Treibersoftware von Sun. Eine Lösung ist nach Angaben von Sun nach Eintreffen der endgültigen Version 2.0 zu erwarten.

4.2 Treiber-Programmierschnittstelle

Das Treiber-API stellt dem Benutzer zwei Möglichkeiten zur Verfügung, PVCs über die ATM Karte mit dem sa(7) Interface, das die Schnittstelle zwischen der Anwendung und der SunATM-Karte darstellt, zu erzeugen:

4.2.1 Direkter Zugriff auf das Interface

Man kann direkt auf das sa(7) Interface durch eine Reihe von ioctl(2) Aufrufen zugreifen. Dies sind im einzelnen:

- A_ALLOCBW - Alloziert Bandbreite für diesen Stream
- A_RELSEBW - Gibt vorher allozierte Bandbreite wieder frei
- A_ADDVC - Öffnet einen VPCI[11] auf diesem Stream
- A_DELVC - Entfernt einen VPCI vom Stream
- A_ALLOCBW_VC - Alloziert Bandbreite für einen speziellen VPCI
- A_RELSEBW_VC - Gibt für diesen VPCI allozierte Bandbreite wieder frei

4.2.2 Indirekter Zugriff auf das Interface

Die zweite Möglichkeit auf das sa(7) Interface zuzugreifen, stellen die sa_util(3) Funktionen dar, welche in einer C-Library zusammengefaßt wurden. Es existieren folgende Funktionen:

- sa_open() - Öffnet einen Stream zu dem physikalischen Interface (sa0)
- sa_close() - Schließt den Stream
- sa_attach() - Assoziiert einen *Physical Point of Attachment* (PPA) mit einem geöffnetem Stream
- sa_detach() - Entfernt die Assoziation zwischen PPA und Stream
- sa_bind() - Bindet *Service Access Point* (SAP) an offenen Stream
- sa_unbind() - Entfernt die Assoziation zwischen SAP und Stream
- sa_setraw() - Setzt Stream in einen sogenannten Raw-Modus, bei dem VPCI sowohl von der Anwendung an den Stream, als auch umgekehrt immer durchgereicht werden muß
- sa_add_vpci() - Öffnet einen VPCI auf diesem Stream
- sa_delete_vpci() - Entfernt einen VPCI vom Stream
- sa_allocate_bw() - Alloziert Bandbreite für diesen Stream
- sa_release_bw() - Gibt vorher allozierte Bandbreite wieder frei

11. der VPCI (*Virtual Path Connection Identifier*) ist eine Zusammenfassung von einem VPI und einem VCI in einem einzigen Feld.

4.2.3 Vergleich der beiden Möglichkeiten

Der direkte Zugriff auf das Interface ist dem indirekten aus zwei Gründen vorzuziehen:

(1) Nur so ist es möglich, direkt auf ATM Ebene Zellen zu versenden (bei den sa_util(3) Funktionen wird immer AAL5 verwendet).

(2) Die sa_util(3) Funktionen bilden auch nur ein Interface zu den direkten ioctl(2) Aufrufen. Aus Performancegründen ist es besser, dieses Interface zu umgehen.

5 Zeitmessungen unter UNIX

Für das im Rahmen dieser Arbeit zu implementierende Programm ist es wichtig, möglichst genaue Zeitmessungen im Mikrosekundenbereich vorzunehmen. Dabei ist es nur nötig, Zeitdifferenzen zu messen. Die Absolutwerte der Zeit werden zwar benötigt (für Zeitstempel in der Logginginformation), müssen aber nicht den gleichen hohen Anforderungen wie bei den Delaymessungen genügen, bei welchen man eine Genauigkeit im Mikrosekundenbereich erwartet. Dazu wurden die Möglichkeiten von UNIX, insbesondere von Solaris 2.4 untersucht.

5.1 Funktionen

Um das Ziel, also möglichst hochauflösende Zeitmessungen unter UNIX auf Userlevel zu erreichen, gibt es zwei, auf den ersten Blick geeignete Funktionen: gettimeofday(3) und gethrtime(3). Beide wurden auf der Referenzmaschine, einer SPARCstation 20 (SS 20) unter Solaris 2.4, untersucht.

5.1.1 gettimeofday(3)

Der Aufruf von gettimeofday(3) liefert eine C-Struktur mit der aktuellen Tageszeit. Diese Struktur beinhaltet im ersten Teil die Sekunden seit dem 1.1.1970 und in einem zweiten Feld die Mikrosekunden seit der letzten vollen Sekunde.

Um die genaue Auflösung auf der SS20 zu bestimmen, wurde ein kleines Testprogramm geschrieben. Der prinzipielle Aufbau dieses Programms ist aus Abbildung 5.1 ersichtlich. Dabei werden die konkreten Zeitstempel mit eben der Funktion gettimeofday(3) geholt. Der konkrete C-Code ist in Anhang B - Testprogramme für die Zeitmeßroutinen einzusehen.

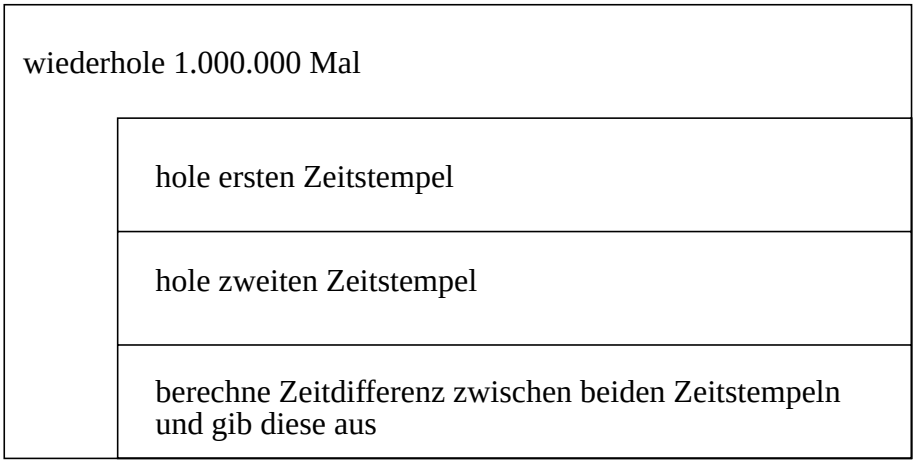

Abbildung 5.1 - Struktogramm des Testprogramms für gettimeofday(3)

Mit diesem Testprogramm wurden die in Tabelle 5.1 gezeigten (gerundeten) Werte ermittelt (es ist angegeben wie oft welche Zeiten gemessen wurden). Dabei kann man sehen, daß zwei aufeinanderfolgende Aufrufe der gettimeofday(3) Funktion im Mittel 2 µs brauchen. Außerdem benötigen ca. 0,05% der Messungen mehr als 5 µs.

Zeit in µsec	**prozentualer Anteil**
1	0,006%
2	98,99%
3	0,074%
4	0,017%
5	0,004%
sonstige	0,046%

Tabelle 5.1 - Ergebnis des gettimeofday(3) Tests

5.1.2 gethrtime(3)

gethrtime(3) liefert unabhängig von der Systemuhr eine Zeit in Nanosekunden, aus der zwar keine Rückschlüsse auf die aktuelle Zeit möglich sind, die aber sehr wohl benutzt werden kann, um Zeitdifferenzen zu messen, da vom System sichergestellt ist (der Rückgabewert ist ein 64 Bit Integerwert, mit dem man einen Zeitrahmen von 584.942 Jahren abdecken kann), daß es sich um eine fortlaufende Zeit handelt, d.h. bei zwei aufeinanderfolgenden Messungen erhält man immer positive Differenzen (kein Überlauf, nie gleiche Zeiten).

Auch diese Funktion wurde mit einem Testprogramm auf ihre Brauchbarkeit für Delaymessungen getestet. Das Programm hat die gleiche Struktur wie das Testprogramm aus Kapitel 5.1.1, nur das diesmal die Zeitstempel über den gethrtime(3)-Aufruf geholt werden. Der C-Code wird in Anhang B - Testprogramme für die Zeitmeßroutinen gezeigt.

Dabei kam es zu den in Tabelle 5.2 gezeigten Ergebnissen (gerundete Werte). Man sieht, daß die meisten Versuche 1,5 µs dauern und nur ca. 0,035% mehr als 3,5 µs.

Zeit in µsec	**prozentualer Anteil**
1,0	0,008%
1,5	74,031%
2,0	25,63%
2,5	0,236%
30	0,055%
3,5	0,003%
sonstige	0,037%

Tabelle 5.2 - Ergebnis des gethrtime(3) Tests

5.1.3 Gegenüberstellung der Funktionen

Wie aus den Testergebnissen ersichtlich ist, spielt es keine große Rolle, ob man nun gethrtime(3) oder gettimeofday(3) für seine Messungen benutzt. Für das zu implementierende Programm wurde gethrtime(3) gewählt, da die Zeitauflösung hier geringfügig besser ist. Leider ist die hier ermittelte Genauigkeit der Zeitfunktionen rein akademischer Natur, da es zwischen den Messungen keinerlei weiterführende Aktivität des Programms gab und auch nur unbedeutend viel Rechenleistung durch das System genutzt wurde, da eine unbelastete SPARCstation 20 zum Einsatz kam. In der Praxis, und das belegen die Referenzmessungen des in der Arbeit entwikkelten Programms, ist eine Zeitmessung im Mikrosekundenbereich auf Benutzerebene unter Solaris nicht machbar.

5.2 Problematiken

Unter UNIX ergeben sich durch die Architektur der Systeme einige Probeme, die, zumindest auf Benutzerebene, nicht lösbar sind. Es gibt zwar auch UNIX-Echtzeitsysteme (Solaris ist ein Ansatz dazu), aber auch bei diesen werden keine Schaltzeiten im Mikrosekundenbereich auf Userebene garantiert. Die wichtigsten Probleme sind das nicht deterministische Zeitverhalten des Kerns bei Systemcalls und natürlich die Beeinflussung des Systemverhaltens bei Prozeßwechseln, beim Paging und Swapping. Lösungsmöglichkeiten werden in Kapitel 8 (Zusammenfassung und Ausblick) besprochen.

5.2.1 Systemcalls

Problematisch sind sowohl Systemcalls anderer Prozesse als auch die eigenen. Ein Systemcall wird immer vollständig im Betriebssystemkern abgearbeitet, wobei die zur Verarbeitung benötigte Zeit keineswegs deterministisch ist. Je nach Implementierung der im Kernel aufgerufenen Routine wird dabei mehr oder weniger Zeit “vergeudet”, d.h. vom Kern selbst in Anspruch genommen. So bremsen manche Operationen, wie z.B. das Lesen von einer Festplatte, das System stark aus. Das ströt im normalen Betrieb nicht weiter, aber bei zeitkritischen Anwendungen, bei denen Zeitdifferenzen im Mikrosekundenbereich benötigt werde.

In den Meßroutinen selbst sind auch einige Systemcalls erforderlich. Das sind im einzelnen die Zeitmessungen gettimeofday(3) und gethrtime(3) und dann noch die putmsg(2) und getmsg(2) Aufrufe zum Versenden bzw. Empfangen von Paketen. Dabei wird der Programmablauf so lange unterbrochen, bis der Kern die verlangten Daten bereitgestellt und in den Benutzerspeicher kopiert hat. Die dafür benötigte Zeit ist nicht exakt zu bestimmen und stark von der Auslastung der Maschine abhängig.

5.2.2 Prozeßwechsel, Paging und Swapping

Ein weiterer Punkt, der die Meßgenauigkeit des Programms stark einschränkt, sind die Prozeßwechsel. Bei jedem Wechsel ist ein vollständiges Kopieren des Kontextes des Prozesses nötig, was allein schon recht zeitintensiv sein kann. Wenn auch noch Teile des Prozesses auf die bzw. von der Festplatte aus- bzw. eingelagert werden müssen (Swapping, Paging), erfordert das sehr viel und vor allem unbestimmbar viel Zeit. So wird der virtuelle Speicher, ein großer Vorteil der UNIX Systeme, zum einschränkenden Faktor für eine korrekte Zeitmessung im Mikrosekundebereich. Das Paging kann man allerdings, viel Hauptspeicher vorausgesetzt, durch den mlockall(3)-Aufruf abstellen, wodurch dem System verboten wird, Teile des Prozesses auf den Swap-Bereich auszulagern.

6 TMT4ATM - Transmit delay Measurement Tool for ATM

Im Rahmen dieser Studienarbeit wurde ein Programm, namens TMT4ATM - Transmit delay Measurement Tool for ATM entwickelt, das der Überwachung von ATM Netzen oder Teilen davon dient. Es ermöglicht die Delaymessung und der Berechnung des sogenannten Jitters (durch Skripten) aus den Meßdaten.

Der Begriff Jitter (*Cell Delay Variation* - CDV) steht für die Schwankungsbreite der Delaywerte (*Cell Transfer Delay* - CTD). Die Berechnung folgt im Falle des B-WiN der Definition aus [Forum95], d.h. der Jitter ergibt sich aus:

$$CDV^{\alpha} = CTD^{\alpha}_{max} - CTD_{min} \qquad \text{Gleichung 6.1}$$

Dabei steht CDV^{α} für den CDV-Wert, der von höchstens α% aller Meßwerte überschritten wird. Für das B-WiN ist α auf 1 festgelegt. Diese Form der Berechnung wird gewählt, um “Ausreißer” in den Messungen zu eliminieren.

Dieses Programm soll sowohl im Batchbetrieb für Langzeitmessungen einsetzbar sein, aber auch, am besten mit grafischer Benutzeroberfläche ausgestattet, für kurze Tests von ATM Verbindungen dienen. TMT4ATM speichert alle Meßergebnisse in einer frei wählbaren Datei (oder gibt sie auf der Standardausgabe aus), so daß es sehr einfach ist, die anfallenden Daten durch Perl- oder Shellskripten auszuwerten.

Das Programm ist ein vier Teilbereiche untergliedert:

- Hauptprogramm mit kompletter Meßroutine
- Initialisierungsroutine
- Schnittstelle zur API der ATM-Karte
- Grafisches X11-Frontend mit voller Statistikfunktionalität

6.1 Hauptprogramm mit kompletter Meßroutine

Alle hardwareunabhängigen Operationen, wie das kontrollierte Versenden und Empfangen von ATM-Zellen und die Steuerung der Zeitmessung, wurden in einem Modul zum Hauptprogramm (main.c) zusammengefaßt. Zuerst werden alle internen Variablen initialisiert (siehe Kapitel 6.2 - Initialisierungsroutine), dann wird eine hardwareabhängige Routine zur Vorinitialisierung der ATM-Karte gestartet und endlich die Messung gestartet. Ein grober Ablaufplan ist in Abbildung 6.1 zu sehen.

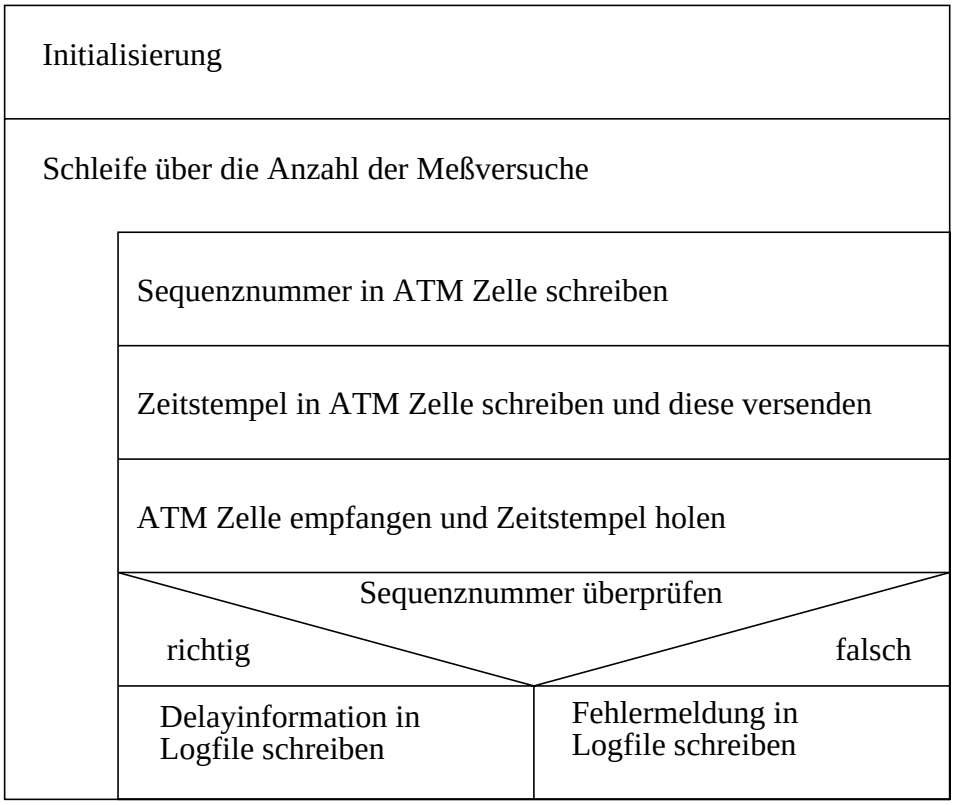

Abbildung 6.1 - Prinzipieller Ablaufplan von TMT4ATM

6.1.1 Versenden und Empfangen von ATM-Zellen

Es werden zwei verschiedene Betriebsmodi unterschieden:

- Zweirechnersystem mit EchoTask
- Einrechnersystem mit Loop über das ATM-Netzwerk

Beim Zweirechnersystem wird zuerst eine ATM-Verbindung zwischen den beiden Computern aufgebaut und auf der Empfängerseite ein einfacher Prozeß gestartet, der einfach alle ankommenden ATM-Pakete[12] reflektiert (EchoTask). Dann erst setzt der Meßprozeß ein.

Beim Einrechnersystem mit Loop über das ATM Netzwerk muß ein VC im ATM-Netz so konfiguriert werden, daß er als Loop zurück zum Rechner zeigt. Dann wird über diesen VC die Messung gestartet.

12. Hier wird wieder der Begriff Paket benutzt, da es sich um eine definierbare Anzahl (im Normalfall aber nur eine) von ATM Zellen handelt.

Der Meßprozeß, der bei beiden Versionen identisch ist, besteht aus zwei unabhängigen UNIX-Tasks, wobei die eine als Sender fungiert, der mit einer konstanten Rate Pakete (ein Paket besteht aus einer definierbaren Anzahl von Zellen) über das ATM-Interface sendet. Die andere Task ist der Empfangsprozeß, der die Pakete empfängt, auswertet und die eigentliche Zeitmessung durchführt. Dazu wurde in jedes Paket vom Sender ein Zeitstempel eingebaut, der nun ausgewertet werden kann.

6.1.2 Steuerung der Zeitmessungen

Die Zeitmessung wurde als Zwei-Prozeß-System mit einem Sendeprozeß und einem Empfangsprozeß implementiert, da es sonst sehr schwer wäre, sowohl mit einer konstanten Zellrate zu senden, als auch sofort bei der Ankunft eines Pakets dieses auszuwerten und entsprechende Informationen in das Outputfile zu schreiben.

6.1.2.1 Sendetask

Zur Messung der Laufzeit einer (vieler) ATM-Zelle(n) (es wird immer die Laufzeit eines Paketes bestimmt, das aus konfigurierbar vielen Zellen besteht), wird kurz vor dem Absenden eines Paketes mit gethrtime(3) ein Zeitstempel geholt und im Paket gespeichert. Weiterhin wird noch eine Sequenznummer eingefügt, um Paketverluste feststellen zu können. Dann wird das Paket über das API dem ATM-Interface zum Verschicken übergeben. Der Begriff Paket wurde gewählt, da es sich um konfigurierbar viele ATM Zellen handelt. Sequenznummer und Zeitstempel werden immer an den Anfang der ersten ATM Zelle des Paketes geschrieben.

Um im Mittel eine bestimmte Senderate einzuhalten, ist ein variables Delay einzubauen. Dies geschieht mit einem einfachen adaptiven Algorithmus, der bei einer hohen Senderate dafür sorgt, daß nicht zuviel Zeit für das Ermitteln der Datenrate benötigt wird, oder genauer, er übergeht sich selbst bei Zeitmangel. Bei niedrigen Raten wird mittels der aktuellen Zeit (ermittelt von gettimeofday(3)), einer initialen Startzeit und der gewünschten Senderate das nötige Delay errechnet und dann mit der Funktion my_usleep(), die eine beliebige Zeit in Mikrosekunden wartet, eine Verzögerung eingebaut. Der benutzte Algorithmus ist in Abbildung 6.2 zu sehen. my_usleep() benutzt den select(3) Aufruf, der u.a. (eigentlich ist er nicht für diesen Zweck gedacht) nach einer in Mikrosekunden definierbaren Zeit abbricht.

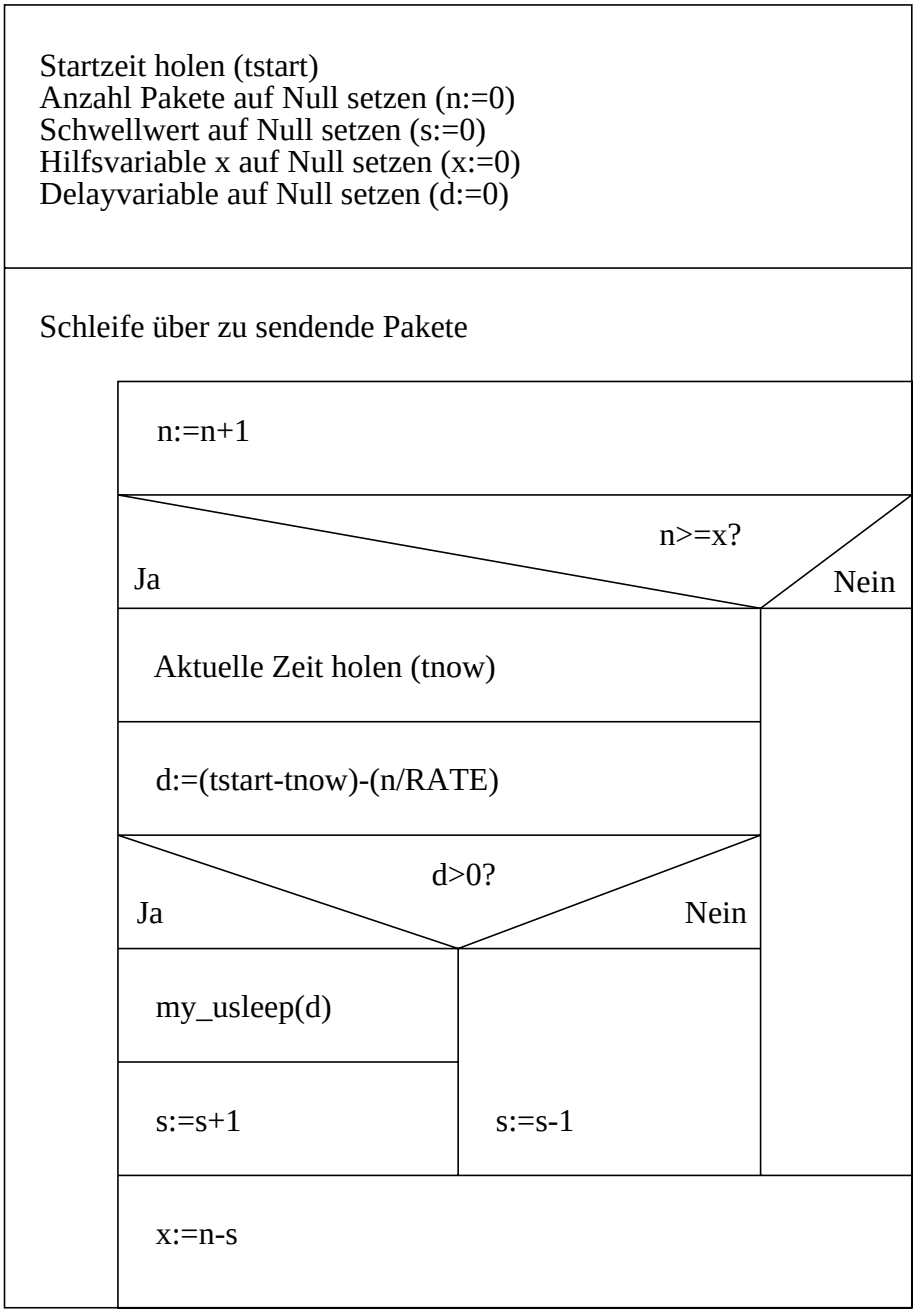

Abbildung 6.2 - Routine zur Einhaltung einer definierten Senderate

6.1.2.2 Empfängertask

Der Empfängerprozeß empfängt ankommende Pakete vom Interface, wobei er mit blockierender IO arbeitet, d.h. er kommt erst zum Zug, wenn wirklich ein Paket angekommen ist. Die erste Operation nach dem Paketempfang ist der Aufruf von gethrtime(3) zur Ermittlung des aktuellen Zeitstempels (Empfangszeit). Danach wird die Sequenznummer überprüft und im Fehlerfalle eine entsprechende Meldung ausgegeben. Ist das richtige Paket angekommen, wird die Differenz von Sende- und Empfangszeit ausgegeben.

6.2 Initialisierungsroutine

Zur Konfiguration des Programms sollten sowohl Parameter über die Kommandozeile, als auch Parameter über diverse Konfigurationsfiles übergeben werden. Gelöst wurde das Problem, indem zuerst ein Standard-Konfigurationsfile eingelesen wird, danach alle über die Kommandozeile angegebenen Konfigurationsfiles und erst zuletzt die Kommandozeilenparameter ausgewertet werden. Es werden jeweils die Einstellungen überschrieben. Dadurch ist es sehr einfach möglich, ein globales Konfigurationsfile zu schreiben und für jedes Teilproblem ein weiteres, in dem nur die zu ändernden Parameter angegeben werden.

Zum Einlesen der Konfigurationsfiles wurde der flex(1)[13] benutzt, mit dem sich sehr leicht einfache Auswerteprogramme für reguläre Grammatiken (sogenannte Parser) bauen lassen. Ein Konfigurationsfile hat folgenden Aufbau:

```
# Kommentar
<TAG> Information
```

Die definierten Tags und ihre Bedeutung sind aus Tabelle 6.1 ersichtlich.

Tag	Information	Bedeutung
DEBUG	0 bis 5	Debuglevel
METHOD	SA IO SIG	Benutzung der sa_util(3) Funktionen der API Direkte Kommunikation mit dem Device sa(7) Benutzung von Signalling mit dem Device q93b(7)
AAL	0 oder 5	AAL Level
VPI	0 bis 256	VPI
VCI	0 bis 65.535	VCI
SRC_ADDR	ATM address	ATM Zieladresse für Signalling
DST_ADDR	ATM address	ATM Quelladresse für Signalling
BANDWIDTH	0 bis 134	reservierte Bandbreite
INTERFACE	sa0	ATM Interface
RATE	1 bis 3000	Pingrate

Tabelle 6.1 - Definierte Tags in den Konfigurationsdateien

13. Der flex(1) ist ein Tool zur einfacheren Erstellung von Parsern. Eine genauere Beschreibung ist in den Manualseiten zum flex(1) zu finden.

Tag	Information	Bedeutung
COUNT	0 bis 1.000.000	Anzahl der zu verschickenden Pakete (0 steht für unendlich viele)
CCP	1 bis 100	Anzahl Zellen pro Paket
OUTPUT	filename	Name der Logdatei (stdout für STDOUT)

Tabelle 6.1 - Definierte Tags in den Konfigurationsdateien

Die Kommandozeile wird mit getopt(3) analysiert. Alle Funktionen des Konfigurationsfiles sind auch durch Optionen steuerbar. Der Aufruf ’tmt4atm -h’ zeigt alle verfügbaren Optionen. Mit der Option ’-v’ kann man sich die aktuelle Programmversion anzeigen lassen.

```
# tmt4atm -h
Usage: tmt4atm [-E] [-f <file>] [-D <level>] [-m <method>] [-a <level>] [-
svpi <vpi>] [-dvpi <vpi>] [-svci <vci>] [-dvci <vci>] [-sa <addr>] [-da
<addr>] [-b <bandwidth>] [-i <interface>] [-r <rate>] [-c <count>] [-o <file>]
[-u] [-q]
     -E                - starte EchoTask
     -f <file>         - Konfigfile [main.cfg]
     -D <level>        - Debuglevel (0-5) [5]
     -m <method>       - Methode (SA/IO/SIG) [SA]
     -a <level>        - AAL Level (0-5) [5]
     -P <vpi>          - Source VPI [0]
     -C <vci>          - Source VCI [99]
     -s <addr>         - Source ATM Address
     -d <addr>         - Destination ATM Address
     -b <bandwidth>    - Bandwidth (MBit/s) [10]
     -i <interface>    - Interface [sa0]
     -r <rate>         - Pingrate (pkt/s) [10]
     -c <count>        - Anzahl der zu uebertragenden Pakete [100]
     -p <ccp>          - Anzahl der Zellen pro Paket [1]
     -o <file>         - Outputfilename [stdout]
     -u                - Ungepufferte Ausgabe
     -q                - weniger ‘geschwaetzige’ Ausgabe

     tmt4atm -v
                       - Ausgabe der Versionsnummer
#
```

```
# tmt4atm -v
This is TMT4ATM (Transmit delay Measurement Tool for ATM) Version 1.0
     Copyright (c) 1996 by Falko Dressler (fd@bsd.rrze.uni-erlangen.de)
#
```

6.3 Schnittstelle zum API der SunATM™-155 Karte

Die ganze Funktionalität des Verbindungsaufbaus, der Übertragung von ATM-Zellen und der korrekten Terminierung der Verbindung wurde in ein separates Modul (sunatm.c) gesteckt, so daß sich eine Portierung auf eine andere Hardwareumgebung vereinfachen würde.

Folgende C-Funktionen mit beschriebener Funktionalität wurden erstellt (das “sun” steht für die spezielle Hardware und API der SunATM-Karte und ist durch einen anderen Term bei der Portierung auf andere Hardwareumgebungen zu ersetzen), wobei allen die Übergabe eines Zeigers auf eine, die wichtigsten Verbindungsparameter für die SunATM Karte beinhaltende Struktur (sunatm_connection) gemeinsam ist:

• sunatm_init()

  Hier werden alle zur Initialisierung der Karte und der Software notwendigen Operationen durchgeführt. Diese Funktionalität wird z.Z. nur beim Signalling benötigt.

  Prototyp:

  ```
  int sunatm_init(struct sunatm_connection *ac, int sap);
  ```

  Aufrufparameter:

  ```
  ac      - Zeiger auf eine sunatm_connection Struktur
  sap     - Service Access Point
  ```

  Rückgabewerte:

  ```
  0       - es ist kein Fehler aufgetreten
  -1      - q_ioc_bind() ist fehlgeschlagen
  -2      - open() ist fehlgeschlagen
  ```

• sunatm_create_vc()

Mit dieser Funktion wird ein VC (PVC) geöffnet, über den dann die Pakete übertragen werden sollen. Konkret wird abhängig von der benutzten Methode, mit der die Karte angesprochen wird, entweder indirekt mit den sa_util(3) Funktionen oder direkt mit ioctl(2) Aufrufen, der VC aufgebaut. Wird als Methode Signalling gewählt, werden alternative Funktionen aufgerufen.

Prototyp:

```
int sunatm_create_vc(struct sunatm_connection *ac);
```

Aufrufparameter:

```
ac      - Zeiger auf sunatm_connection Struktur
```

Rückgabewerte:

• sa_util(3):

```
0       - es ist kein Fehler aufgetreten
-1      - sa_setraw() ist fehlgeschlagen
-2      - sa_add_vpci() ist fehlgeschlagen
-3      - sa_allocate_bw() ist fehlgeschlagen
-4      - sa_attach() ist fehlgeschlagen
-5      - sa_open() ist fehlgeschlagen
```

• sa(7) via ioctl(2):

```
0       - es ist kein Fehler aufgetreten
-3      - ioctl() A_ALLOCBW_VC ist fehlgeschlagen
-4      - ioctl() A_ADDVC ist fehlgeschlagen
-5      - open() ist fehlgeschlagen
```

• sunatm_delete_vc()

Hiermit wird ein bestehender VC (PVC) wieder geschlossen, d.h. vom System abgemeldet. Wie beim Öffnen des VCs wird abhängig von der gewählten Methode ein bestimmter Programmteil ausgeführt.

Prototyp:

```
int sunatm_delete_vc(struct sunatm_connection *ac);
```

Aufrufparameter:

```
ac      - Zeiger auf sunatm_connection Struktur
```

Rückgabewerte:

• sa_util(3):

```
0       - es ist kein Fehler aufgetreten
-3      - sa_close() ist fehlgeschlagen
-4      - sa_release_bw() ist fehlgeschlagen
-5      - sa_delete_vpci() ist fehlgeschlagen
```

• sa(7) via ioctl(2):

```
0       - es ist kein Fehler aufgetreten
-4      - close() ist fehlgeschlagen
-5      - ioctl() A_DELVC ist fehlgeschlagen
-6      - ioctl() A_RELSEBW_VC ist fehlgeschlagen
```

• sunatm_connect_via_signalling() und sunatm_waitcon_via_signalling()

Wird als Methode Signalling gewählt, wird mit den qcc_bld(3) Funktionen und dem Device q93b(7) eine ATM-Verbindung über das Netz via Signalling (siehe Kapitel 3.1.4 - Signalling) aufgebaut. Leider wird diese Methode in der aktuellen Version des Programms nicht unterstützt, da zwar ein Verbindungsaufbau möglich ist, aber keine Datenübertragung über den erzeugten VC[14]. Die benötigten Routinen sind zwar schon im Programm vorhanden, nur eben nicht getestet. Auf Senderseite wird die Funktion sunatm_connect_via_signalling() aufgerufen und auf Empfängerseite sunatm_waitcon_via_signalling().

• sunatm_connect_via_signalling()

Prototyp:

```
int sunatm_connect_via_signalling(
        struct sunatm_connection *ac);
```

Aufrufparameter:

```
ac      - Zeiger auf sunatm_connection Struktur
```

Rückgabewerte:

```
0       - es ist kein Fehler aufgetreten
<0      - es ist ein Fehler aufgetreten
```

14. Ein Bugreport an Sun ist unterwegs, aber z.Z. noch unbeantwortet.

• sunatm_waitcon_via_signalling()

Prototyp:

```
int sunatm_waitcon_via_signalling(
        struct sunatm_connection *ac);
```

Aufrufparameter:

```
ac      - Zeiger auf sunatm_connection Struktur
```

Rückgabewerte:

```
0       - es ist kein Fehler aufgetreten
<0      - es ist ein Fehler aufgetreten
```

• sunatm_release_via_signalling() und sunatm_waitrel_via_signalling()

Diese Funktionen werden benutzt, um eine via Signalling aufgebaute Verbindung wieder zu beenden. Dabei wird vom Sender nach erfolgreicher (Meß-)Datenübertragung sunatm_release_via_signalling() aufgerufen. Der Kommunikationspartner nimmt das RELEASE mit der Funktion sunatm_waitrel_via_signalling() entgegen und sendet ein RELEASE_COMPLETE als Antwort zurück. Erst nachdem dieses den Sender der RELEASE Botschaft erreicht hat, ist die Verbindung korrekt beendet.

• sunatm_release_via_signalling()

Prototyp:

```
int sunatm_release_via_signalling(
        struct sunatm_connection *ac);
```

Aufrufparameter:

```
ac      - Zeiger auf sunatm_connection Struktur
```

Rückgabewerte:

```
0       - es ist kein Fehler aufgetreten
<0      - es ist ein Fehler aufgetreten
```

• sunatm_waitrel_via_signalling()

Prototyp:

```
int sunatm_waitrel_via_signalling(
        struct sunatm_connection *ac);
```

Aufrufparameter:

```
ac      - Zeiger auf sunatm_connection Struktur
```

Rückgabewerte:

```
0       - es ist kein Fehler aufgetreten
<0      - es ist ein Fehler aufgetreten
```

• sunatm_transmit()

Unabhängig von der gewählten Methode wird diese Funktion zur Übertragung eines Paketes beliebiger Länge (das sind u.U. einige ATM Zellen) über das ATM Netz benutzt. System V typisch (wir benutzten Solaris 2.4), wird das über Streams mit putmsg(2) getan.

Prototyp:

```
int sunatm_transmit(int fd, char *ctlbufp, int ctllen,
        char *databufp, int datalen);
```

Aufrufparameter:

```
fd      - Filedescriptor des Streams
ctlbufp - Zeiger auf den Kontrollpuffer
ctllen  - Länge des Kontrollpuffers
databufp- Zeiger auf den Datenpuffer
datalen - Länge des Datenpuffers
```

Rückgabewerte:

```
0       - es ist kein Fehler aufgetreten
-1      - putmsg() ist fehlgeschlagen
```

• sunatm_receive()

Diese Funktion holt ein Paket vom ATM Interface ab und gibt dieses an das Hauptprogramm weiter. Analog zum Verschicken von Paketen, wird die Streams Funktionalität durch die Funktion getmsg(2) genutzt.

Prototyp:

```
int sunatm_receive(int fd, char *ctlbufp, int *ctllenp,
        char *databufp, int *datalenp, int maxlen);
```

Aufrufparameter:

```
fd      - Filedescriptor des Streams
ctlbufp - Zeiger auf den Kontrollpuffer
ctllenp - Zeiger auf das Kontrollpufferlängenfeld
databufp- Zeiger auf den Datenpuffer
datalenp- Zeiger auf das Datenpufferlängenfeld
maxlen  - maximale Länge des Kontroll- und Datenpuffers
```

Rückgabewerte:

```
0       - es ist kein Fehler aufgetreten
-1      - getmsg() ist fehlgeschlagen
ctlbufp - gelesener Kontrollblock
ctllenp - gelesene Länge des Kontrollblocks
databufp- gelesener Datenblock
datalenp- gelesene Länge des Datenblocks
```

6.4 Grafisches X11 Frontend mit Statistikfunktionalität

Zur einfachen und schnellen Erstellung eines grafischen Frontends zum Programm wurde das tk-Toolkit[15] benutzt, mit dem sich sehr einfach eine X11-Oberfläche entwickeln läßt. Als Programmiersprache wurde Perl (in diesem Fall tkperl(1)) gewählt.

Da das Programm TMT4ATM für einfache, über eine X11-Oberfläche steuerbare Kurzzeitmessungen von Delays, als auch für Langzeitmessungen gedacht ist, wurde alle Statistikfunktionalität aus TMT4ATM herausgenommen und in die Oberfläche bzw. in diverse Steuerskripten integriert.

Die Kommunikation von tmt4atm und dem X11-Frontend, xtmt4atm, geschieht über Pipes, was sich als technisch am einfachsten realisierbar herausstellte. Dazu wird STDERR von TMT4ATM auf eine Pipe und STDOUT auf eine zweite umgeleitet. Nach dem Aufbau des X11-Fensters bleibt xtmt4atm in der tkmainloop() Schleife, die alle Aktivitäten des Benutzers, wie z.B. das Drücken eines Buttons, registriert und entsprechende Callbackfunktionen aufruft. Die Auswertung der Daten von tmt4atm wird mittels asynchroner IO aus den Pipes gelesen und sofort ausgewertet (angezeigt, Mittelwert, Minimum und Maximum berechnet, etc.). Die Eingaben des Benutzers (z.B. VCI) werden immer sofort auf Fehler überprüft und gegebenenfalls werden diese sofort berichtigt.

Eine noch anzufertigende Erweiterung wäre die Möglichkeit, Konfigurationsfiles einzulesen bzw. abzuspeichern.

15. tk ist eine Entwicklung zur Gestaltung für X11 Anwendungen für die Programmiersprache tcl. Mittlerweile existiert auch eine Implementierung für Perl. Genauere Informationen sind den Manualseiten zu entnehmen.

7 Messungen

Es wurden sowohl Referenzmessungen des Programms angefertigt, die dann mit den Ergebnissen eines richtigen ATM Monitors verglichen wurden, als auch konkrete Messungen im B-WiN.

7.1 Referenzmessungen

Als Referenz wurde eine Messung mit TMT4ATM von einer SPARCstation 20 über einen ForeRunner™ ASX-200WG ATM Switch mit einer Messung des Hewlet Packard® ATM Monitors BSTSpx (Broadband Series Test System) verglichen. Wie erwartet (siehe Kapitel 5.2 - Problematiken) unterschieden sich die Messungen von TMT4ATM sehr von denen des ATM-Monitors. Dabei ist anzumerken, daß es sich bei den Referenzmessungen um rein lokale handelt. Die Delays im B-WiN sind schon aufgrund der Leitungslänge und durch die Anzahl der Switches um einiges größer. Leider ist eine Jittermessung trotzdem nicht möglich, da man nur das Systemverhalten der benutzten Workstation testen würde. Eine sinnvolle Nutzung des Programms ist weitergehend möglich, um z.B. die Verfügbarkeit von ATM-Verbindungen zu testen (durch Auswertung der Zellverlustmeldungen) oder um Veränderungen der Leitungsqualität festzustellen.

In den Abbildungen 7.1 und 7.2 wird eine Delaymessung mit dem HP ATM Monitor gezeigt. Daraus ist ersichtlich, daß das Delay über den Fore Switch im Mittel 16 µs beträgt mit Schwankungen von ±0,5µs.

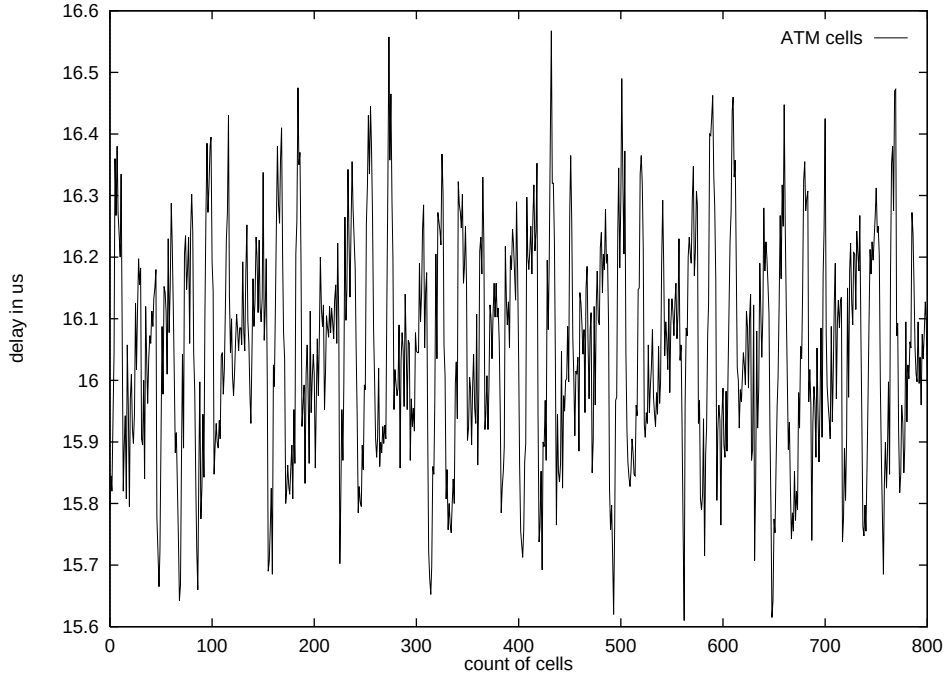

Abbildung 7.1 - Delaymessung auf ATM Ebene mit dem HP Monitor

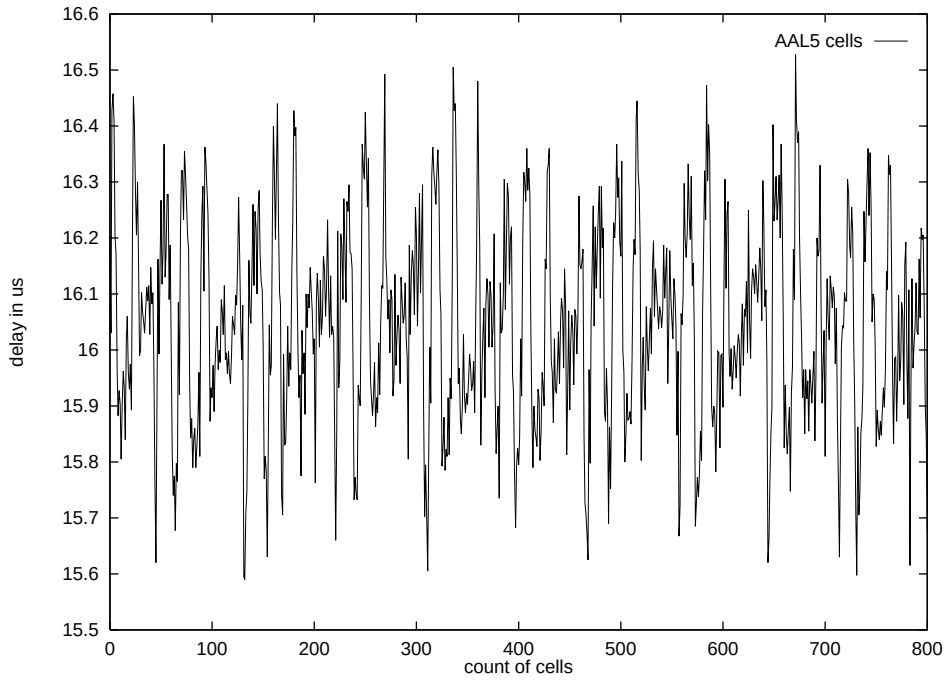

Abbildung 7.2 - Delaymessung über AAL5 mit dem HP Monitor

Führt man die gleichen Messungen mit TMT4ATM durch, ergibt sich ein Delay von ca. 500 µs bei Schwankungen von bis zu 25 ms (Abbildung 7.3). Die in Abbildung 7.4 und 7.5 gezeigte Ausschnittsvergrößerung der Meßwerte zeigt deutlich die kurzzeitigen Abweichungen des gemessenen Delays, was auf Swapping/Paging Aktivitäten der genutzten Workstation zurückzu-

führen ist. Bei Benutzung einer schnelleren Workstation, die ansonsten gänzlich unbelastet ist, war eine Abnahme dieser extremen Meßfehler festzustellen. Eine Möglichkeit, die systemabhängigen Probleme zu lösen, wird in Kapitel 8 (Zusammenfassung und Ausblick) gezeigt.

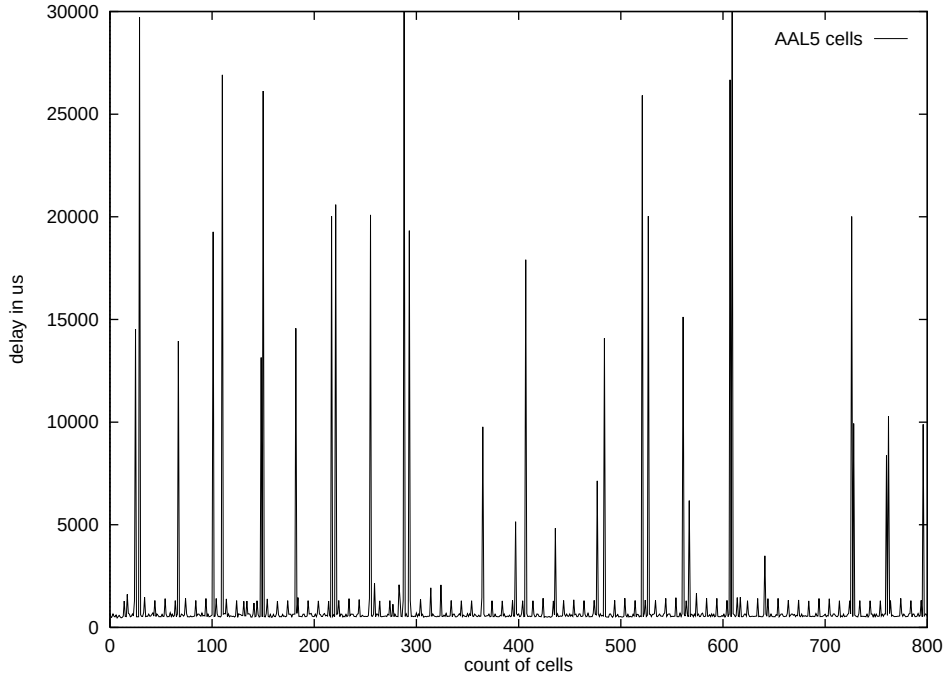

Abbildung 7.3 - Delaymessung über AAL5 mit TMT4ATM mit Methode SA

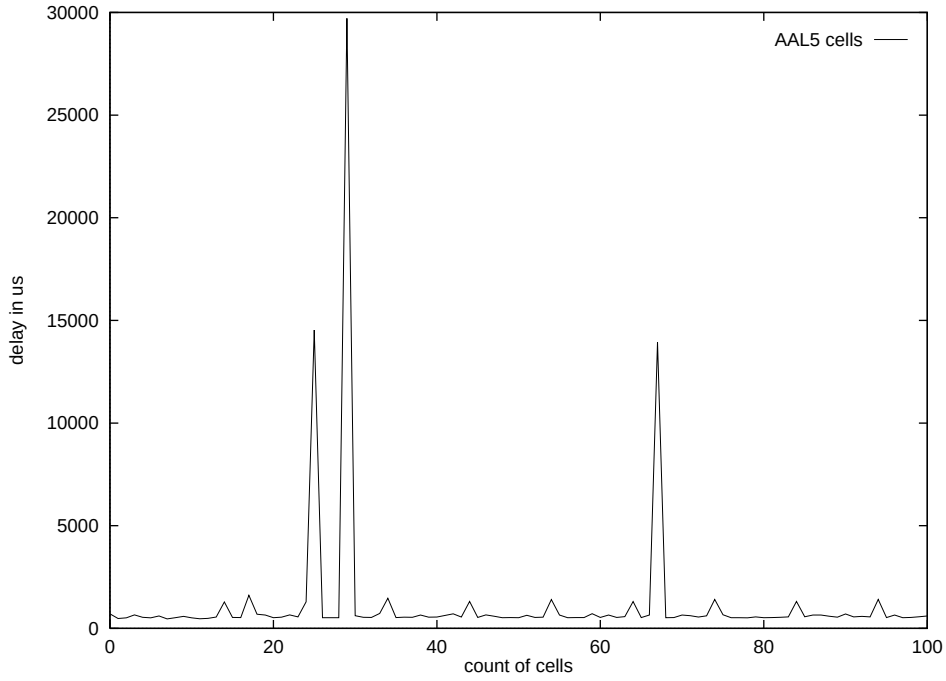

Abbildung 7.4 - Vergrößerung von Abbildung 7.3

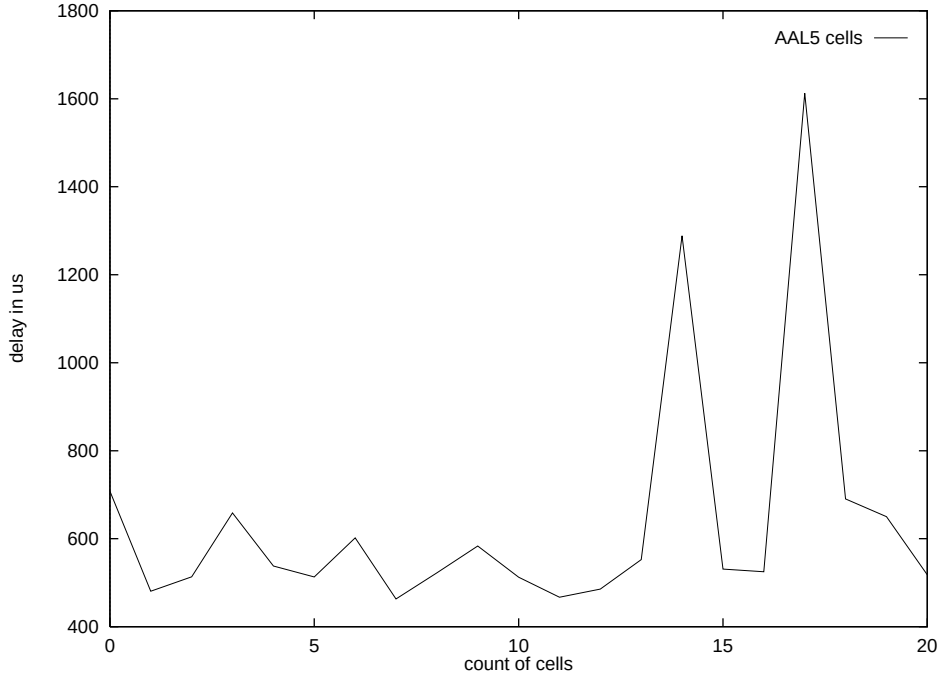

Abbildung 7.5 - Vergrößerung von Abbildung 7.4

7.2 Messungen im B-WiN

Im Breitband-Wissenschaftsnetz (B-WiN) wird vom Betreiber bestimmte Dienstgüteparameter wie z.B. ein bestimmtes End-to-End Delay garantiert. Um die Einhaltung dieder Dienstgüteparamter zu überwachen, soll das Programm TMT4ATM - Transmit delay Measurement Tool for ATM, welches im Rahmen dieser Arbeit entstand eingesetzt werden. Mit dem Programm ist es sowohl möglich Delays zu messen, als auch durch die Auswertung der Logfiles den Jitter zu bestimmen.

Um ein so komplexes Netz wie das deutsche B-WiN (siehe Abbildung 1.1) korrekt durchmessen zu können, ist es nötig, jede Teilstrecke des Netzes einzeln zu überwachen und das Delay auf diesem Stück zu messen. Nur so kann man Ausagen über das Gesamtverhalten des Netzes gewinnen.

ATM stellt durch seine verbindungsorientierte Arbeitsweise und das VPI/VCI Konzept Mittel bereit, die es erlauben Schleifen durch ein ATM Netzwerk zu legen. Um nun das Delay im B-WiN zu bestimmen, hat man im Auftrag des DFN-Vereins Loops für jeden Teilabschnitt des Netzes konfiguriert, über die man mit dem entwickelten Tool das Delay messen kann. Für jede einzelne Schleife wird je ein Prozeß auf einer für die Delaymessung reservierten Workstation gestartet. Mit den erhaltenen Daten, die man über Skripten weiterverarbeiten kann, können nun

Aussagen über das Gesamtverhalten des Netzes gemacht werden. So kann z.B. festgestellt werden, ob eine Verbindung komplett ausgefallen ist, oder ob eine Verbindung plötzlich ein übermäßig hohes Delay aufweist.

Leider ist es aufgrund der Beschränkungen des UNIX Betriebssystems nicht möglich, exakte Delays für eine Jitterbestimmung zu messen, sondern man kann nur Aussagen über das Delayverhalten einer Leitung machen. Möglichkeiten, diesen Mangel zu beheben, werden in Kapitel 8 (Zusammenfassung und Ausblick) besprochen.

8 Zusammenfassung und Ausblick

In der vorliegenden Studienarbeit wurden die Grundlagen für die Entwicklung und die Implementation eines Programms zur Bestimmung von Dienstgüteparametern in ATM-Netzen dargelegt. Das Programm nutzt alle Möglichkeiten, die sich auf Benutzerebene unter UNIX bieten, Zeiten im Mikrosekundenbereich zu messen, aus, und ist für konkrete Messungen im entstehenden B-WiN gedacht. Es können sowohl Delaymessungen als auch prinzipiell (über zusätzliche Skripten) Jittermessungen gemacht werden.

Leider stellte sich im Verlauf der Arbeit heraus, daß die Meßdaten nicht die erwartete Genauigkeit ausweisen, d.h. daß Delaymessungen im Mikrosekundenbereich unter UNIX auf Benutzerebene nicht machbar sind. Auch besonders leistungsfähige Workstations wie z.B. eine SPARCstation 20 mit 64 MB RAM erreichen nicht die geforderte Leistung.

Um das Programm doch noch für brauchbare Delaymessungen nutzbar zu machen, gibt es eigentlich nur einen sinnvollen Ansatz: Man muß die Zeitstempel direkt im Kern in die ATM-Zellen schreiben. Dies war im Rahmen dieser Arbeit leider nicht mehr möglich, da Sun Microsystems die Quellen des Treibers für die ATM-Karte nicht rechtzeitig liefern konnte.

Zum Austesten dieses Ansatzes kann man in jede ausgehende und ankommende ATM-Zelle (außer in solchen auf reservierten VPI/VCIs) an definierte Stellen Zeitstempel einbauen. Mit der Auswertung dieser Zeitstempel kann man sich auf Benutzerebene dann Zeit lassen, da hier keine zeitkritischen Operationen mehr ausgeführt werden müssen.

Führt dieser Test zu brauchbaren Meßergebnissen, so muß man den ATM-Treiber so erweitern, daß man im Kern eine Tabelle mit VCs (VPI/VCI) anlegt, und nur noch in Zellen auf diesen VCs Zeitstempel einbaut. Dadurch wird der normale ATM-Verkehr, wie z.B. IP über ATM oder beliebige andere ATM Anwendungen, nicht mehr eingeschränkt und man kann trotzdem genaue Messungen durchführen. Für die Wartung der Tabelle, d.h. Einfügen und Entfernen von Einträgen, schreibt man am besten einen neuen Systemcall, der genau die gewünschte Funktionalität zur Verfügung stellt.

Eine entsprechende Erweiterung von TMT4ATM ist sehr einfach, da man eigentlich nur die Funktionalität einschränken muß. D.h. man muß einfach nur die Programmteile der Sendetask entfernen, die die Zeitstempel in die ATM-Zellen schreiben bzw. den Empfänger so verändern, daß er nur noch die Zeitinformationen aus den Zellen liest und auswertet, statt selbst nach dem Empfang einer Zelle vom System einen Zeitstempel holt. Außerdem muß man natürlich den neuen Systemcall einbauen, so daß der Meß-VC in die Tabelle aufgenommen wird.

A Die SunATM™-155 SBus Karte

A.1 Version

Anfangs arbeiteten wir mit Version 1.0, in der leider noch einige Fehler anzutreffen waren. Nach einem Bugreport an Sun bekamen wir Version 2.0b zum Test. Leider stellte sich heraus, daß auch hier noch nicht alle Funktionen der API nutzbar waren. Ein Test der Version 2.0 war nicht mehr möglich, da Sun diese Version nicht rechtzeitig liefern konnte.

A.2 Installation

Die Installation der Karte ist in zwei Teile gegliedert:

(1) Hardwareinstallation

Die Hardwareinstallation beschränkt sich auf das Einsetzen der Karte in einen freien SBus Slot in der SPARCstation und das Verbinden der Workstation mit einem ATM Switch. Die Installation kann man vom Bootmanager der Sun aus mit dem Kommando show-devs testen:

```
<#0> ok show-devs /iommu/sbus
/iommu@f,e0000000/sbus@f,e0001000/sa@3,0
...
<#0> ok
```

(2) Softwareinstallation

Die Softwareinstallation geschieht Solaris typisch durch das Einspielen der speziellen Softwarepakete:

```
# pkgadd -d /cdrom/sunatm_1_0 SUNWatm SUNWatmu SUNWatma
```

A.3 Konfiguration

Die Konfiguration beschränkt sich auf die des IP. Dazu muß ein VC konfiguriert werden, der speziell für IP reserviert ist. Dabei passt die Treibersoftware die für IP zur Verfügung gestellte Bandbreite automatisch an, d.h. es wird immer die maximal verfügbare Bandbreite benutzt.

Es müssen die Dateien /etc/aarconfig und /etc/hostname.sa0 angelegt bzw. angepaßt werden. In ./etc/hostname.sa0 wird der Rechnername, den das Interface sa0 bekommen soll eingetragen. Ein Beispiel für die Datei /etc/aarconfig ist:

```
Interface   Host    ATM Address                 VCI     Flag
sa0         -       <20-stellige ATM Adresse>   -       L
```

Als Daemonprozesse müssen aarpd (*ATM ARP Daemon*) und der ilmid (ATM Adressregistrierungsprozeß) während des Bootvorganges der Workstation gestartet werden.

Nach einem ”boot -rv” ist das ATM Interface voll einsatzfähig.

B Testprogramme für die Zeitmeßroutinen

B.1 gettimeofday.c

```
#include <sys/time.h>
int main(void) {
     struct timeval t1,t2;
     int i;
     for(i=0;i<1000000;i++) {
          gettimeofday(&t1,0);
          gettimeofday(&t2,0);
          printf(„%ld usec\n“,
               (t2.tv_sec*1000000+t2.tv_usec)-
               (t1.tv_sec*1000000+t1.tv_usec));
     }
     return 0;
}
```

B.2 gethrtime.c

```
#include <sys/time.h>
int main(void) {
     hrtime_t t1,t2;
     int i;
     for(i=0;i<1000000;i++) {
          t1=gethrtime();
          t2=gethrtime();
          printf(„%lld nsec\n“,t2-t1);
     }
     return 0;
}
```

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[Pry93] - Asynchronous Transfer Mode - Solutions for Broadband ISDN, Second Edition, Martin de Prycker, Alcatel Bell, Antwerp, Belgium, Ellis Horwood Limited, GB, 1993

[BSTS95-1] - Implementing ATM Signalling: Avoiding the Interoperability Pitfalls, Hewlett Packard Company, BSTS Solutions Note 5963-7514E, 1995

[BSTS95-2] - Operation LANs in an ATM Environment, Hewlett Packard Company, BSTS Solutions Note 5963-7513E, 1995

[Forum95] - The ATM Forum - Technical Committee: Traffic Management Specification Version 4.0, ATM Forum/95-0013R8, Straw Vote, Oct. 1995

[Forum94] - ATM User-Network Interface Specification, Version 3.1, ATM Forum, Sept. 1994

[SunATM] - SunATM™-155 SBus Cards Manual, Sun Microsystems Computer Company, Revision A, May 1995

[DFN95] - DFN Mitteilungen, Heft 37, März 1995

[Teke94] - ATM Pocket Guide, Tekelec, Publication 908-0119-01, Revision B, July 1994

[B-WiN] - B-WiN-Vertrag, DFN Verein, Sept. 1995

Abkürzungsverzeichnis

AAL..... ATM Adaptation Layer

AFI ..... Authority and Format Identifier

ANSI..... American National Standards Institute

API ..... Application Programmer’s Interface

ATM ..... Asynchronous Transfer Mode

BCH..... Bose-Chadhuri-Hocquenghem Code

B-ISDN..... Broadband (aspects of) Integrated Services Digital Network

B-WiN ..... Breitband-Wissenschaftsnetz

CBR..... Constant Bit Rate

CCITT ..... Comité Consultatif Internationale Télégraphique et Téléphonique, heute ITU

CLP..... Cell Loss Priority

CPCS ..... Common Part Convergence Sublayer

CS ..... Convergence Sublayer

ESI..... End System Identifier

GFC ..... Generic Flow Control

HEC..... Header Error Control

ILMI ..... Interim Local Management Interface

ISDN..... Integrated Services Digital Network

ISO ..... International Standards Organisation

ITU ..... International Telecommunications Union

ITU-T ..... International Telecommunications Union’s Telecommunications Standardization Sector

LAN..... Local Area Network

MAC..... Medium Access Control

MIB ..... Management Information Base

MTBF..... Mean Time Between Failure

NNI..... Network Node Interface

NSAP..... Network Service Access Point

OSI ..... Open Systems Interconnection

PCR ..... Peak Cell Rate

PDU ..... Protocol Data Unit

PPA ..... Physical Point of Attachment

PTI ..... Payload Type Identifier

PVC ..... Permanent Virtual Circuit

QoS ..... Quality of Service

RFC ..... Request for Comments

RTB ..... Regionales Testbed

SAP ..... Service Access Point

SAR ..... Segmentation And Reassembly (sublayer)

SDU ..... Service Data Unit

SNMP ..... Simple Network Management Protocol

SONET ..... Synchronous Optical Network

SSCS ..... Service Specific Convergence Sublayer

SVC ..... Switched Virtual Circuit

UNI ..... User Network Interface

VBR ..... Variable Bit Rate

VC ..... Virtual Channel

VCC ..... Virtual Channel Connection

VCI ..... Virtual Channel Identifier

VP ..... Virtual Path

VPC ..... Virtual Path Connection

VPCI ..... Virtual Path Connection Identifier

VPI ..... Virtual Path Identifier

WAN ..... Wide Area Network

WiN ..... Wissenschaftsnetz