Pufferdimensionierung für schwankungsbeschränkte Ströme

Technische Universität Dresden

Großer Beleg

Fakultät Informatik

Frank Unger

Matrikelnummer: 2295185

Dresden, den 25.09.1998

Vorwort

Rechnernetze, mit dem Internet als wohl bekanntestes unter ihnen, werden für jedermann immer wichtiger. In Zukunft werden sie das Telefon an Bedeutsamkeit überholen, nicht zuletzt dadurch, weil Telefonie mit ihnen leicht zu realisieren ist. Daran schließt sich die Frage nach der Architektur und Arbeitsweise solche Netze an. Heute setzt sich immer mehr das TCP/IP- Protokoll durch. Die Hardware spielt bei dieser Art von Betrachtungen eine große Rolle, auch die Adaptation an verschiedene Betriebsysteme stellt einen Problemfaktor dar. Fragen der Sicherheit in Netzen und lokalem Mehrbenutzerbetrieb müssen fast ausschließlich durch Software gelöst werden.

In heutiger Zeit, wo das Wort „Multimedia“ auch schon kleinen Kindern bekannt ist, muß man auch beantworten können, was es bedeutet. Ich möchte hier keine vollständige Definition angeben, aber eines ist wohl das wichtigste Merkmal aus dem Blickwinkel eines Softwareentwicklers: die riesigen Datenmengen. Da es sich vornehmlich um Audio- und Videodaten handelt, haben Übertragungen natürlich in einem ununterbrochenen Strom zu erfolgen und müssen eine Mindesttransferrate erfüllen. Aussetzer hierbei können die übermittelten Informationen schnell unbrauchbar machen. Es existieren schon Radiosender im Internet, die ihre Sendungen nur in digitaler Form verschicken, d.h. direkt codiert über das Netz.

Um noch einmal zum Thema der Vernetzung zurückzukehren möchte ich auf die Aspekte einer verteilten Anwendung hinweisen. Ein Rechnerverbund, geschaffen für solch ein Aufgabengebiet, wird mehr und mehr auch diese Multimediafähigkeiten aufweisen müssen. Als Kern dessen gelten die Anforderung um die QoS - Kriterien (quality of service) zu erfüllen. Das ist das Hauptanliegen dieses Belegs in Verbindung mit der Übertragung großer Datenmengen in Portionen. Den Haupttenor machen mathematische Grundlagen und Beispielberechnungen im Sachgebiet aus, z.B. die Berechnung des minimal notwendigen Puffer für eine gültige Übertragung.

Inhalt

1 Beispielarbeiten zum Themenkreis

1.1 Grundprinzipien von J. Gemmel und S. Christodoulakis

1.1.1 Einleitung

Im Paper [2] werden Grundprinzipien für die Darstellung und Speicherung von Multimediadaten (digitale Audiodaten, Videodaten oder Animationen) erklärt. Die Ausführungen erfolgen auf der Basis von Audiodaten, sind aber leicht auf alle Daten dieser Kategorie übertragbar.

Das Paper soll all denen helfen, die Multimediasysteme entwerfen, Hardwareanforderungen abschätzen müssen oder mögliche Entwurfsentscheidungen bewerten sollen.

1.1.2 Motivation

Wie sollte man sich die Wiedergabe von digitalen Audiodaten vorstellen? Die Daten befinden sich auf der Festplatte, einem Vertreter für externe Speichermedien. Diese werden durch den DA-Wandler verbraucht, benötigt. Dazwischen geschaltet befindet sich der zu dimensionierende Puffer. Seine Notwendigkeit entsteht durch den benötigten Datenstrom, der besonderen Anforderungen genügen muß, z.B. darf er nicht abreißen bzw. eine Mindestrate nicht unterschreiten. Mit "read" (lesen) wird hier der Datentransport von der Festplatte in den Puffer bezeichnet; "consume" (verbrauchen) wird für die Bezeichnung des Datentransports vom Puffer zum DA-Wandler verwendet.

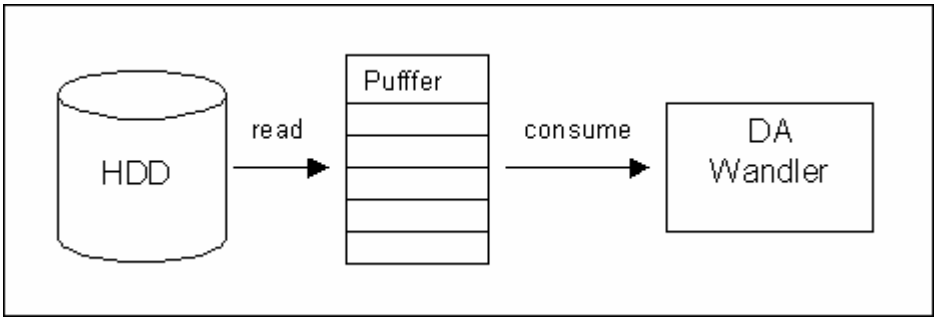

Abb. 1 Der schematische Aufbau

Beim Digitalisieren der Audiodaten werden diese mit einer speziellen Samplefrequenz abgetastet. Genau an diesen Abtastpunkten entstehen Werte, die "Samples". Die Granularität der Samples gibt deren Wertebereich an und wird meist in Bit angegeben. Sie selbst bilden die Daten für den digitalen Audiostrom, die exakt rückgewandelt werden müssen, will man die Audioinformationen hörbar machen. Damit ergibt sich für jedes einzelne Sample eine Deadline, die sich aus der Samplefrequenz ableiten läßt.
Da nun die Daten von der Festplatte nicht regulär geliefert werden können, ist eine Pufferung notwendig, mit der man den geforderten Datenstrom für die korrekte Wiedergabe bereitstellen kann. Der Puffer nimmt die Datendiffernz auf, die sich aus Lese- und Verbrauchsfunktion ergibt.

1.1.3 Grundgrößen

$R(t)$ - reading function – Lesefunktion
ergibt die Anzahl aller bis und einschließlich Zeit t gelesenen Sampels, füt $t < 0$ gilt: $R(t) = 0$
zur Zeit t_r ist das Lesen beendet; daraus folgt für $t > t_r$: $R(t) = R(t_r)$

Es handelt sich um eine monoton steigende Funktion, da die Anzahl der insgesamt eingelesenen Daten nicht rückläufig sein kann. Der Anstieg ist also nichtnegativ. Sie kann jedoch waagerecht verlaufen, insbesondere während solchen Zeiten, in denen gerade nichts gelesen werden kann. Damit sind z.B. Positionierzeiten des Schreib-Lese- Kopfes oder ein voller Puffer gemeint.

$C(t)$ - Consumption function – Verbrauchsfunktion
hat die Form f(x) = mx, also linear

Für den Fall, daß r_c Sampels pro Sekunde verbraucht werden und der Startzeitpunkt mit t_0 festgesetzt ist, ergibt sich auch folgende Definition:

$$C(t,t_0) = \begin{cases} 0 & \text{für} \quad t < t_0 \\ r_c(t-t_0) & \text{"} \quad t \geq t_0 \end{cases} \qquad (1)$$

Hier wird also ein wichtiger Aspekt für unsere weiteren Betrachtungen deutlich. Der Bedarf an Daten als Funktion der Zeit ändert sich nicht sprunghaft. Die Funktion hat keine Treppenform. Die Datenanforderung bzw. der Verbrauch wird als kontinuierlicher Strom modelliert. Es kommen also nur Multimediadatenströme in Frage, die diesem Modell genügen, z.B. Audiodaten im ungepackten Format (alle Ausführungen basieren in ihrer Anschauung auf diesen Daten). In diese Klasse fallen z.B. MPEG - Datenströme nicht.

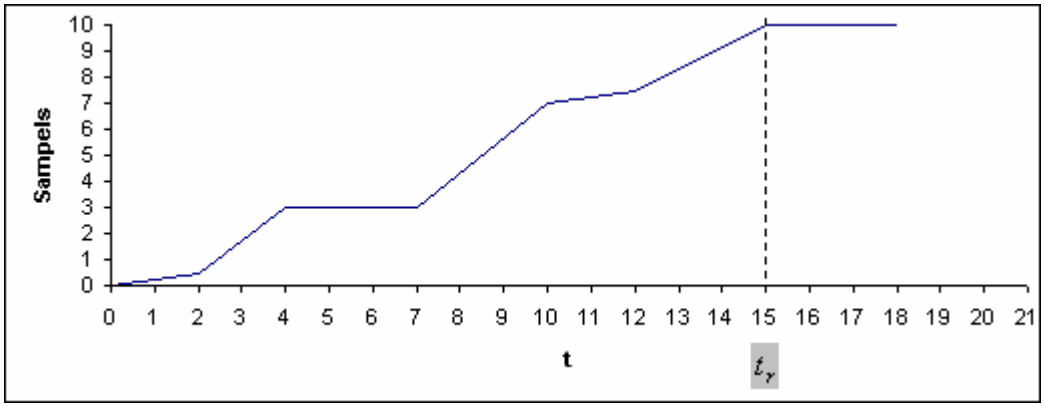

Abb. 2 R(t), Die Lesefunktion

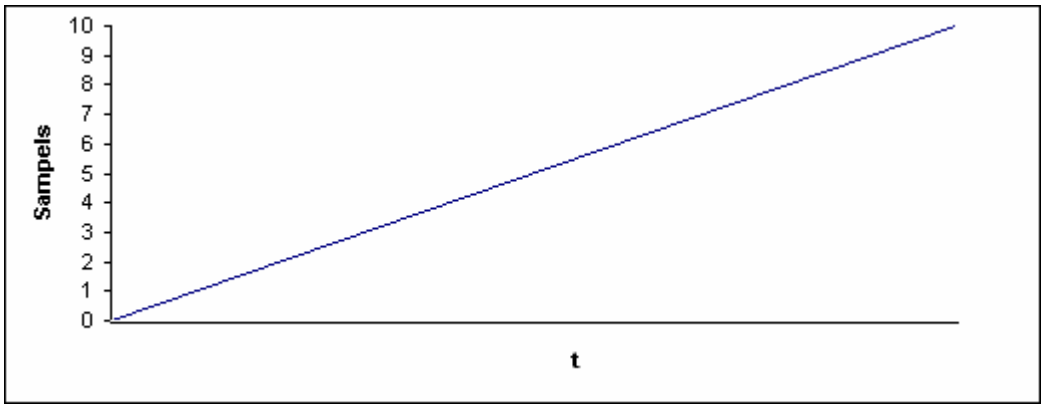

Abb. 3 C(t), Die Verbrauchsfunktion

1.1.4 Wert für die minimale Startzeit und Dimensionierung des Puffers

Im vorherigen Abschnitt haben wir gesehen, wie Lese- und Verbrauchsfunktion verlaufen. Wird nun mehr gelesen als verbraucht, ergibt sich eine zu puffernde Datenmenge. Diese ist ebenfalls in Abhängigkeit von der Zeit definiert, und zwar:

$$\boxed{B(t,t_0) = R(t) - C(t,t_0)} \qquad (2)$$

Der Wert *B* wird als Pufferausnutzung („buffer utilization“) bezeichnet. Das Maximum aller Werte für B stellt die benötigte Puffergröße dar.

Es wird sofort klar, das für die Fehlerbedingung gilt:

$$B(t,t_0) < 0 \qquad \text{,im Intervall}[t_{0,}t_r]$$

In diesem Fall wurde zu wenig gelesen. t_r ist der Endzeitpunkt, zu dem alle Samples eingelesen worden sind.

Ziel ist nun, $B(t,t_0)$ so zu finden, daß eine mögliche Wiedergabe repräsentiert wird (playback solution). Um das zu gewährleisten, muß folgendes erfüllt sein:

$$B(t,t_0) \geq 0 \qquad \text{,im Intervall}\,[t_{0,}t_r]$$

Gesucht wird einzig und allein t_0, für das die Bedingung erfüllt ist. Die Existenz eines solchen t_0 ist gesichert, denn für $t_0 = t_r$ werden alle Samples gepuffert, und es ergibt sich eine korrekt ablaufende Wiedergabe, also eine Lösung im obigen Sinne. Dabei erreicht aber die benötigte Puffergröße einen inakzeptabel großen Wert (Maximum). Die Zeit t_0 ist auch maximal groß. Natürlich wird eine Lösung angestrebt, bei der der Puffer möglichst klein ist und der Startzeitpunkt des Erzeugerprozesses möglichst spät liegt.

Im Paper ([2], S.56 ff) ist das Theorem 1 einschließlich Beweis zu finden. Es besagt, daß die Suche nach der minimalen Startzeit t_0 auch zum Ergebnis für P_{min} führt. Im Theorem 2 ist angegeben, wie man den minimalen Wert für t_0 ermittelt:

a) Ist $B(t, t_0)$ nichtnegativ in $[0, t_r]$, dann ist $B(t, t_0)$ eine Lösung und $t_0 = 0$

b) Das Minimum von $B(t, t_0)$ in $[0, t_r]$ befindet sich beim Zeitpunkt t_{min}.

Es ist:

$$B(t_{\min},0) = -m$$

Der Schnittpunkt der Funktionsgraphen von $R(t)$ und $B(t, t_0) + m$ liegt dann bei $t = t_0$, der minimalen Startzeit für eine Lösung gemäß den Bedingungen s.o. .

Der Beweis hierfür ist ebenfalls aufgeführt. Ein schematisches Beispiel ist in Abb. 4 angegeben.

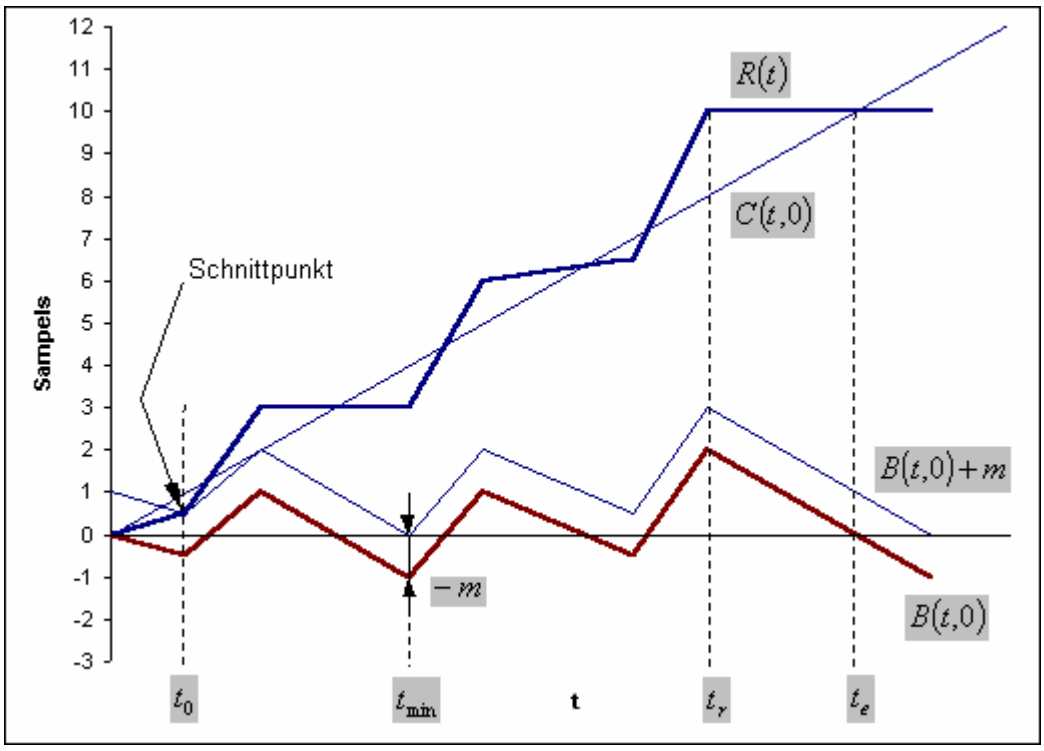

Abb. 4 Ermittlung der minimalen Startzeit

1.1.5 Blockweises Lesen – Implikationen

Daten werden normalerweise nicht kontinuierlich gelesen, sondern in Abschnitten bestimmter Größe (Blöcke). Gewöhnlich beträgt die minimale Datengröße für einen Lesevorgang von einer Festplatte 512 Byte. Dieser Abschnitt wird hier auch mit Sektor bezeichnet. Es ist zu beachten, daß es sich dabei meist nicht nur um reine Nutzdaten handelt, sondern z.B. auch Informationen zur Fehlerkorrektur enthalten sein können.

Wie modelliert man nun die Lesefunktion, wenn die Daten sektorweise gelesen werden?
Die Daten werden periodisch als „verbrauchbar“ gemeldet. Es handelt sich um eine Treppenfunktion mit Sprüngen an den Stellen der Nutzbarwerdung der Daten.

Im weiteren sind die Betrachtungen nicht darauf gerichtet, ob für das Einlesen Speicher allokiert wurde, sondern, es stellt sich vielmehr die Frage nach dem Wievielfachen eines Sektors, das als Puffer verfügbar sein muß, um eine Wiedergabe der Daten im Sinne einer Lösung zu garantieren.

Dafür werden die folgenden 3 Fälle unterschieden und behandelt:

(1) Die Leserate vom Gerät ist gleich der Verbrauchsrate
(2) Die Leserate vom Gerät ist geringer als die Verbrauchsrate
(3) Die Leserate vom Gerät ist größer als die Verbrauchsrate

Zu jedem Punkt ist der mathematische Apparat einschließlich Beweis angegeben. Im Rahmen dieser Arbeit ist nur (1) von Bedeutung, aufgrund der gegeben Voraussetzungen. Ich möchte jedoch zunächst auch den praxisrelevanten Punkt (3) verständlich machen.

zu (3):
Voraussetzungen, Begriffsklärung:

- Transferrate > Verbrauchsrate, $r_t > r_c$
- Sektorgröße, s_s
- kontinuierlich lesbare Datenmenge (audio record, das gesamte zu verbrauchende Datenvolumen) der Größe $l_a > 1\ s_s$

Ergebnisse:

- max. Pufferausnutzung u_{max}, mindestens:

$$u_{\max} \geq (l_a - 1)\frac{s_s}{r_t}(r_t - r_c) + s_s \qquad (3)$$

- mindestens benötigte, sektorgroße Puffersegmente n_b

$$n_b \geq \frac{l_a - 1}{r_t}(r_t - r_c) + 1 \qquad (4)$$

Dabei wird die Linksstetigkeit analog zu Kap. 2.2.2 vereinbart mit der Auswirkung, daß zum Zeitpunkt $t = 0$ ein Sektor bereits gelesen und verfügbar ist.

$$R(0) = s_s$$

$B(t, 0)$ ist zu keinem Zeitpunkt negativ. Daraus folgt $t_0 = 0$, basierend auf den Ergebnissen aus 1.1.4.

Es wird aufgezeigt, daß man für den von uns betrachteten Fall, $r_t = r_c$, einen Puffer von genau 2 Sektorgrößen benötigt (Punkt (1)), doch dazu später.
In allen anderen Fällen ist der nötige Puffer abhängig vom Gesamtumfang der wiederzugebenden Daten. Das ist natürlich nicht wünschenswert. Arbeitsspeicher kann man z.B. nur begrenzt ausfassen.

Um die Puffergröße zu reduzieren, werden für $r_t > r_c$ künstliche Verzögerungen (artificial delays) eingefügt. Wie diese Pausen letztendlich implementiert werden, ist freigestellt. Man muß aber physikalische Vorgaben beachten. So ist z.B. eine Lesepause bei Festplatten ein Vielfaches der Zeit für eine Umdrehung des Plattenstapels. Es besteht die Forderung, daß die Größe der Pausen immer ein Vielfaches eine minimalen Dauer d_{min} beträgt.

Als Vorbedingungen für die Benutzung solcher Verzögerungen gelten zwei Ansprüche. Zum Ersten müssen immer gepufferte Daten zum Verbrauch bereitstehen, und zweitens muß zu jedem Zeitpunkt $r_t \geq r_c$ erfüllt sein.

Dann ist die benötigte Anzahl der Puffersegmente mit der Größe eines Sektors für eine korrekt ablaufende Wiedergabe:

$$n_b = \left\lceil \left(\frac{s_s}{r_t} + d_{\min} \right) \cdot \frac{r_c}{s_s} \right\rceil + 1 \qquad (5)$$

zu (1):
Um auf unseren Fall, zurückzukommen, sei noch einmal das Ergebnis für den Puffer angegeben. Zuerst jedoch die Voraussetzungen:

- $r_t = r_c$
- $R(0) = s_s$

Dann sind:

- max. Pufferausnutzung u_{max}, mindestens:

$$\boxed{u_{\max} \geq s_s} \tag{6}$$

- mindestens benötigte (zu allokierende), sektorgroße Puffersegmente n_b

$$\boxed{n_b \geq 2} \tag{7}$$

Beweis:

(a) Kann nicht parallel zum Verbrauch eingelesen werden, sind zwar alle Wiedergabedaten vorher einzulesen und werden damit gepuffert, mindestens aber das Doppelte einer Sektorgröße, was aus der Voraussetzung $l_a > 1\ s_s$ resultiert.

(b) Bei möglichem parallelen Ablauf von Einlese- und Verbrauchsvorgang kann natürlich nicht in ein Puffersegment eingelesen werden, aus dem gerade entnommen wird. Hier muß das Prinzip des wechselseitigen Ausschlusses angewendet werden. Somit werden mindestens zwei sektorgroße Bereiche benötigt, die in ihrer Nutzung alternieren.

1.2 Metascheduling für Datenströme von David P. Anderson [3]

1.2.1 Einleitung

Die Abkürzung CM steht für „continuous media“, und mit diesem Begriff werden hier alle Video- und Audiodatenströme zusammengefaßt. CM – Unterstützung ist integraler Bestandteil eines Betriebsystems, wenn die folgenden Punkte gelten:

- Daten werden in digitaler Form gespeichert und übertragen.
- CM – Daten erfahren die gleiche Handhabung durch die Hardware wie andere Daten auch.
- CM – Daten werden auch im gleichen Softwarerahmen (Betriebssystem, Filesystem, ...) verarbeitet

Typische Applikationen in einem integrierten CM – System sind entfernte Wiedergabe (remote playback, s. Abb. 5) und Telefonie, die in unserem Falle nicht von weiterer Bedeutung ist.

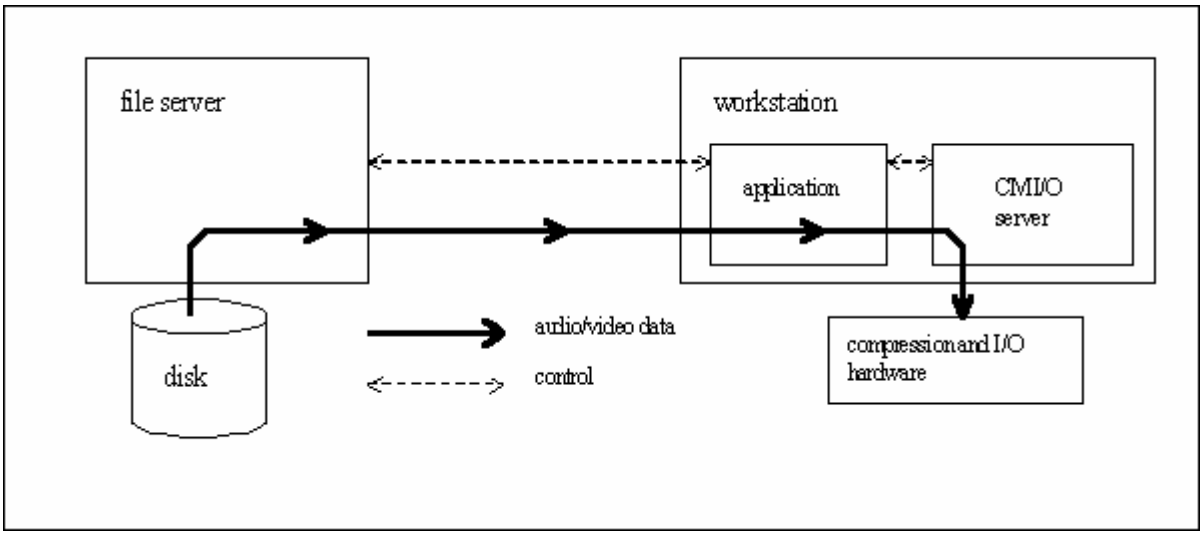

Abb. 5 Entfernte Wiedergabe - remote playback

Diese Anwendungen haben Anforderungen, die es zu erfüllen gilt. Ein Aspekt für die entfernte Wiedergabe ist z.B. der „Punkt zu Punkt“ – Durchsatz. Ein typischer Fall aus der Praxis ist die Übertragungsrate von Festplatte zum Darstellungsgerät. Der Betrag des geforderten Datendurchsatzes ist dabei insbesondere abhängig vom spezifischem Datenformat.

1.2.2 Grundproblematik und Lösungsansatz

Um die gesetzten Ziele zu erreichen, bedarf es bestimmter Anforderungen bezüglich und Beziehungen zwischen den Hardwarekomponenten, Betriebssystemmechanismen und Schedulingstrategien.

Schwerpunkt der Arbeit von Anderson [3] ist der letztgenannte Punkt. Hauptbestandteil sind die Betrachtungen zu einem Reservierungsmechanismus der für alle Komponenten der sich ergebenden Übertragungskette anwendbar sein soll.

Ein Agent (selbständig agierendes und reagierendes Programm mit eigenen Wahrnehmungen, Eindrücken), welcher solche Reservierungen tätigt, wird hier mit „Metascheduler“ bezeichnet. Ausschlaggebend für dessen Aktionen ist das Verhalten der

Clients. Besonders wichtig ist: Er beeinflußt nicht die momentane Auslastung der Komponenten.

Dazu muß ein einheitliches Modell, genutzt von eben diesem Metascheduler, existieren, das Operationen und Interaktionen dieser Komponenten beschreibt. Dabei sind folgende Designziele maßgeblich:

- Start – Ende – Semantik, Beschreibung aller Einzelkomponenten vom Start zum Ziel des Übertragungspfades
- flexible Abstraktion
- Koexistenz, zwischen Echtzeit- und Nichtechtzeitdaten, auch Minimierung des Overheads (Mehroperationen)

Bezeichnet mit „CM – Ressourcen – Modell“ ergibt das die Basis für Metascheduling. Die Clientprozesse benutzen dieses Modell um die entsprechenden Ressourcen für ihre Minimalanforderungen zu reservieren. Weitere Details findet man in den Quellen, die in diesem Paper angeben werden und direkt auf S.229 oben in ihrer Bedeutung aufgegliedert werden. Einige davon habe ich auch im Anhang vermerkt.

1.2.3 Begriffe und Voraussetzungen

1.2.3.1 Ressourcen und Vorausarbeit

Ressource
- einzelne Komponenten (CPU)
- aber auch komplexere Subsysteme (Netzwerk)
- besitzt Standardschnittstelle für Reservierung

Arbeitsmenge (workload)
- parametrisiert durch CM – Ressourcen – Modell
- besteht aus Anzahl diskreter Nachrichten (Einheiten, typischerweise Blöcke der CM - Daten)

$N_I(t_0,t)$
- Anzahl der Nachrichten, die im Interface I (Schnittstelle) während des Zeitintervalls $[t_0,t)$ empfangen werden

LBAP (linear bounded arrival process)
- empfängt Nachrichten
- 3 Parameter:
  - M – maximale Nachrichtengröße [Byte]
  - R – maximale Nachrichtenrate [Nachrichten/Sekunde]
  - W – Vorausarbeitslimit [Nachrichten]
- für alle $t_0 < t$ gilt:

$$N_I(t_0,t) \le R(t-t_0)+W \tag{8}$$

- durchschnittliche Datenrate = MR [Bytes/Sekunde]
- W ermöglicht schnellere logische Übertragung, nicht physikalisch (Im Grunde werden erst später benötigte Daten schon früher angefordert, übertragen und gepuffert)

Vorausarbeit $w(t)$ eines LBAP zur Zeit T ist:

$$w(t) = \max_{t_0<t}\{0, N(t_0,t) - R(t-t_0)\} \tag{9}$$

Bemerkung:

w(t) erhöht sich um 1 bei jedem Nachrichtenempfang und fällt mit dem Anstieg $-R$ zu anderen Zeitpunkten, wird aber nie negativ (s. Abb. 6).

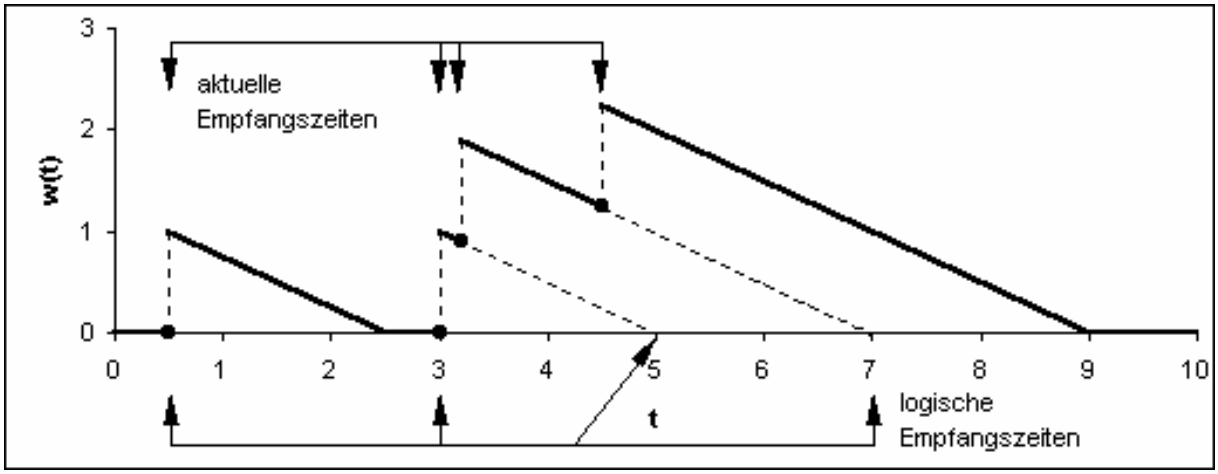

Abb. 6 Vorausarbeit und Empfangszeiten

1.2.3.2 Verzögerung und logische Ankunftszeit

Sei $m_0 ... m_n$ die Folge der Nachrichten und $a_0 ... a_n$ die Ankunfts- bzw. Empfangszeiten der einzelnen Elemente. Die logische Ankunftszeit $l(m_i)$ einer Nachricht m_i ist:

$$\boxed{\begin{aligned} &l(m_i) = a_i + \frac{w(a_i)}{R} \\ &oder\ rekursiv \\ &l(m_0) = a_0 \\ &l(m_{i+1}) = \max\left(a_{i+1}, l(m_i) + \frac{1}{R}\right) \end{aligned}} \qquad (10)$$

Intuitiv könnte man sagen, daß $l(m\ )$ die Ankunftszeit von m wäre, würde man Vorausarbeit nicht zulassen.

$w(t)$ quantifiziert den „Vorsprung“ bezüglich der Vorausarbeit. Der Wert wird benutzt, um logische Ankunftszeiten zu kalkulieren.

Die logische Verzögerung $d(m)$ einer Nachricht m zwischen zwei Interfaces (Schnittstellen) I_1 und I_2 ist:

$$\boxed{d(m) = l_2(m) - l_1(m)} \qquad (11)$$

$l_i(m)$ ist die logische Ankunftszeit der Nachricht m am Interface I.

1.2.3.3 Sitzungen

Ressourcen handhaben CM-Nachrichtenströme und andere natürlich auch. Der Weg der Daten führt dabei über eine Eingabeschnittstelle (input interface), durch die Ressource selbst und dann durch die Ausgabeschnittstelle. Der Client muß eine Ressource für die Nutzung reservieren. Diese Reservierungen werden Sitzungen (sessions) genannt. Eine Sitzung besitzt folgende Parameter:

M	Maximale Nachrichtengröße [Bytes]
R	Maximale Nachrichtenrate [Nachrichten/Sekunde]
W_{in}	Vorausarbeitslimit für die Eingabe [Nachrichten]
W_{out}	Vorausarbeitslimit für die Ausgabe [Nachrichten]
D	Maximale logische Verzögerung [Sekunden]
A	Minimale aktuelle Verzögerung [Sekunden]
U	Minimale ungepufferte aktuelle Verzögerung [Sekunden]

Jede Ressource exportiert eine prozedurale Schnittstelle der folgenden Form für das Erzeugen von Sitzungen:

- reserve()
- relax()
- free()

Die Details einer Schnittstelle hängen vom Ressourcentypus ab. Jede Funktion reserve() bietet eine Kostenfunktion, mit der man die zugehörige Bewertung vornimmt. Man kann eine Objektorientiertheit erkennen.

1.2.4 Zusammengesetzt Sitzungen (compound sessions)

In einer entfernten Wiedergabeanwendung, s. Kap. 1.2.1, kommen mehrere Ressourcen zum Zuge, z.B. CPU1, Netzwerk, CPU2. Im CM Ressourcen Modell wird eine solche Situation als „zusammengesetzte Sitzung“ bezeichnet:

Definition zusammengesetzte Sitzung:
Eine zusammengesetzte Sitzung ist eine Folge von Sitzungen $S_1 ... S_n$, bei denen die Ausgabeschnittstelle von S_i die Eingabeschnittstelle von S_{i+1} ist.

Alle beteiligten Sitzungen, als bildliche Vorstellung ist eine Kette gut geeignet, haben natürlich das gleiche Durchsatzlimit für die Daten; das Vorausarbeitslimit der Ausgabe von S_i kann nicht das Vorausarbeitslimit der Eingabe von S_{i+1} überschreiten. Ansonsten weist eine zusammengesetzte Sitzung äquivalente Merkmale einer Sitzung auf (black box):

- Eingabeschnittstelle ist die von S_1
- Ausgabeschnittstelle ist die von S_n
- Die logische Verzögerung d einer Nachricht m in S ist die Differenz der logischen Ankunftszeiten von m an diesen beiden Schnittstellen.
- Logische Verzögerungen und damit auch Verzögerungsgrenzen addieren sich in zusammengesetzten Sitzungen auf

1.2.5 Verhinderung eines Datenstromabrisses – Verhungern

Sind die Daten bei der Ausgabe nicht rechtzeitig oder gar nicht vorhanden, wenn sie benötigt werden, spricht man von „verhungern“. Dies tritt vor allem dann auf, wenn die Daten mit einer Mindestrate gefordert werden. Die Durchschnittsrate der Datenanlieferung kann zwar über dieser Mindestrate liegen, es kann aber zwischenzeitlich zu Engpässen kommen, die man eben unbedingt vermeiden will. Im folgenden wird gezeigt, wie der Empfänger diese Forderung erfüllt, indem er den Start der Ausgabe zeitlich verzögert.

Man sagt der Client ist konservativ, wenn er Daten zwischenpuffert und bis zur Zeit $t_1 = t_0 + D$ mit dem Beginn der Ausgabe wartet. t_0 ist die Ankunftszeit der ersten Nachricht an der Eingabeschnittstelle von S, und D ist der Grenzwert der Verzögerung von S. Ein Ankunftsprozeß wird Vorausarbeit-positiv genannt, wenn für alle $t \in (a_0, a_n)$ gilt:

$$\boxed{N(a_0, t) \leq R \cdot (t - a_0)} \tag{12}$$

(Bedeutung a_0 s. Formel (10)). In der Verbindung :
Ist der Ankunftsprozeß einer zusammengesetzten Sitzung Vorausarbeit-positiv an dessen Eingabeschnittstelle und der Empfänger ist konservativ, dann kommt es nicht zum Verhungern.

Um dieses konservative Verhalten zu ermöglichen ist es unter anderem notwendig, das Problem der Zeit in verteilten Systemen zu lösen, d.h. die Uhren der an der zusammengesetzten Sitzung beteiligten Rechner müssen synchronisiert werden oder es sind gewisse Parameter bekannt. Dabei werden im Paper drei Möglichkeiten unterschieden :

(1) Voraussetzung ist, daß alle Uhren innerhalb eines Toleranzintervalls ε synchron sind. Der Sender verpaßt der ersten Nachricht einen Zeitstempel (t_0). Der Empfänger verzögert die Ausgabe bis $t_0 + D + \varepsilon$.
(2) Voraussetzung ist, daß alle Verzögerungen vom Sender zum Empfänger bekannt sind. Die erste, eine besondere Nachricht bekommt ein Feld namens „total delay“, initiiert mit Null. Jeder Rechner addiert dann seine lokale Verzögerung und die Verzögerung seiner nachfolgenden Verbindung. Der Empfänger wartet dann nach Erhalt dieser ersten Nachricht D - D_{total}.
(3) Der Empfänger wartet D - $A_{\min}$. $A_{\min}$ ist die Summe aller aktuellen Verzögerungen der Ressourcen in S.

Dabei wird zusätzlich zum benötigten Puffer, der im folgenden betrachtet wird, zusätzlich Pufferplatz gebraucht, und zwar εR für den Fall 1, Null im Fall 2 und R(D - $A_{\min}$) für den letzten Punkt.

1.2.6 Erforderliche Puffergröße

Kommen wir nun zur Berechnung der benötigten Puffergröße für einen Rechner H in einer zusammengesetzten Sitzung S. Zuerst die Symbolik:

$X_1 ... X_n$	Ressourcen in S, die im Hauptspeicher von H Daten puffern
U	Minimale ungepufferte Zeit von X_n
W_{out}	Vorausarbeitslimit für die Ausgabe [Nachrichten]
D_i	Logische Verzögerungsgrenze von X_i , $D = \Sigma_i X_i$
W	das ankommende Vorausarbeitslimit von X_1
R	Maximale Nachrichtenrate von S

Es gilt:
Die maximale Anzahl zu speichernder Nachrichten in einem Rechner H (host) für eine Sitzung S ist:

$$\boxed{W + R(D - U) \; \ldots Puffergröße\, in\, Nachrichten} \qquad (13)$$

Der Beweis ist angegeben. Für weitere Details möchte ich auf das Paper verweisen. Hier sind dann Angaben zu folgenden Themen zu finden:

- Regulierung der Vorausarbeit
- Erstellen zusammengesetzter Sitzungen
- Zusammengesetzte Kostenfunktion
- Implementierungshinweise mit Beispiel CPU und FDDI

2 Berechnungen zu schwankungsbeschränkten Strömen am Lehrstuhl für Betriebssysteme

2.1 Problemstellung und mathematisches Modell

- Erzeugerprozeß EP füllt periodisch einen möglichst kleinen Puffer der Gesamtgröße $P_{\min}$. Die Bereitstellung der Daten erfolgt mit Sicherheit jeweils zu Beginn einer Periode, die Füllzeit wird vernachlässigt („Pufferbereich wird der Anwendung übergeben“).

  T_A - Periodenlänge einer Fülloperation
  $t_A = 0$ - Startzeitpunkt für das Füllen
  Q - Quantum, konstante Datenmenge pro Fülloperation

- Verbraucherprozeß VP entnimmt periodisch Datenmengen. Das Entnehmen (Leeren) dauere die Zeit $L << T_B$, am Ende jeder Periode ist der zugehörige Pufferbereich garantiert frei, die angeforderten Daten sind mit Sicherheit „verbraucht“ („Pufferbereich wird der Anwendung entzogen“).

  T_B - Periodenlänge einer Entnahmeoperation
  $t_B = 0$ - Startzeitpunkt für das Leeren
  $D_j, j=1,...,n$ - spezifisch große Datenmengen, die in der Reihenfolge j entnommen werden
  $n \geq 1$ - Anzahl dieser Datenmengen

- Die mittleren Raten r_A und r_B von EP und VP sollen gleich sein.
- VP darf nicht in den Zustand „wartend“ gelangen. Die pro Periode geforderte Datenmenge muß immer zur Verfügung stehen, sich im Puffer befinden. Dazu wird folgendes benötigt:

  V - Vorlauf von EP
  P_0 - Anfangsfüllstand des Puffers
  T_0 - Vorperiode (Phasenverschiebung) von EP

- Weiter sei

  $D = \sum_{j=1}^{n} D_j > 0$ Gesamtzahl, -menge der angeforderten Daten.

  Die kleinsten Einheiten der Daten werden mit natürlichen Zahlen $1,...,D$ identifiziert (Bytezähler, kontinuierliche äquidistante Speicheradressen, ...).

- Aufgabe: gegeben T_B, Q, (D_j), n
  gesucht T_A, $P_{\min}$, P_0, T_0 , V

2.2 Modell 1 – Beschreibung von VP mittels kumulativer Bedarfsfunktion

2.2.1 Berechnung von T_A

Sei

$$m = \frac{D}{Q}$$

und damit $\lceil m \rceil$ die Anzahl der Einheiten der Größe Q, in denen D eingelesen wird (ggf. sind zuletzt nur $D - \lfloor m \rfloor Q$ Daten zu lesen).

Damit gilt für die mittleren Raten von EP und VP:

$$r_A = \frac{D}{mT_A}, \qquad r_B = \frac{D}{nT_B}$$

Aus der Forderung $r_A = r_B$ folgt sofort

$$T_A = \frac{nT_B}{m}$$

oder unter Berücksichtigung der gegebenen Größen

$$\boxed{T_A = \frac{nQ}{D} \cdot T_B = \frac{Q}{r_B}} \tag{14}$$

Bsp. (s.u.). $Q = 3$, $T_B = 40$ ms, $n = 10$, $D = 20$, $(D_j) = 1, 2, 1, 4, 4, 3, 1, 3, 0, 1$

Dann ist $T_A = \frac{10 \cdot 3}{20} \cdot 40\text{ms} = 60\text{ms}$; $m = \frac{20}{3}$, zuletzt sind $20 - \left\lfloor \frac{20}{3} \right\rfloor \cdot 3 = 2$ Daten einzulesen.

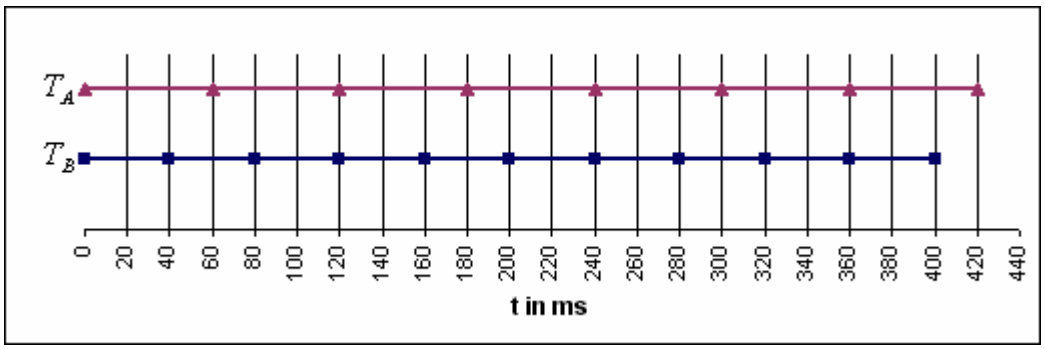

Abb. 7 Die verschiedenen Periodenlängen im Beispiel

Anzahl der Fülloperationen: $f = \lceil m \rceil$.

2.2.2 Berechnung von *V*

Sei

$$S_j := \sum_{k=1}^{j} D_k \quad , j = 1,...,n$$

Nun sei , s. Abb. 8

$$B(t) = \begin{cases} 0 & \text{für} \quad t < 0 \\ S_j & " \quad t \in [(j-1)T_B, jT_B), \; j = 1,...,n \\ D & " \quad t \geq nT_B \end{cases}$$

„kumulative Bedarfsfunktion“, kumulative Entnahme; Treppenfunktion

Sprungstellen – Periode: T_B

Sprunghöhe: D_j

und

$$A(t) = \begin{cases} 0 & \text{für} \quad t < 0 \\ iQ & " \quad t \in [(i-1)T_A, iT_A), \; i = 1,...,\lceil m-1 \rceil \\ D & " \quad t \geq \lceil m-1 \rceil \cdot T_A \end{cases}$$

„vorläufige (kumulative) Angebotsfunktion“, Füllfunktion; Treppenfunktion

Sprungstellen Periode: T_A

Sprunghöhe: D_j (zuletzt $D - \lfloor m \rfloor Q$)

Die Funktionen sind linksstetig. Interpretation s. Abb. 8. Die für eine (eineindeutige) Funktion *F* übliche Bezeichnung F^{-1} der Inversen versteht sich im folgenden als Bezeichnung der Pseudo-Inversen einer Treppenfunktion, d.h. konkret für die folgendermaßen auf [0, *D*] definierten Funktionen (s. Abb. 9).

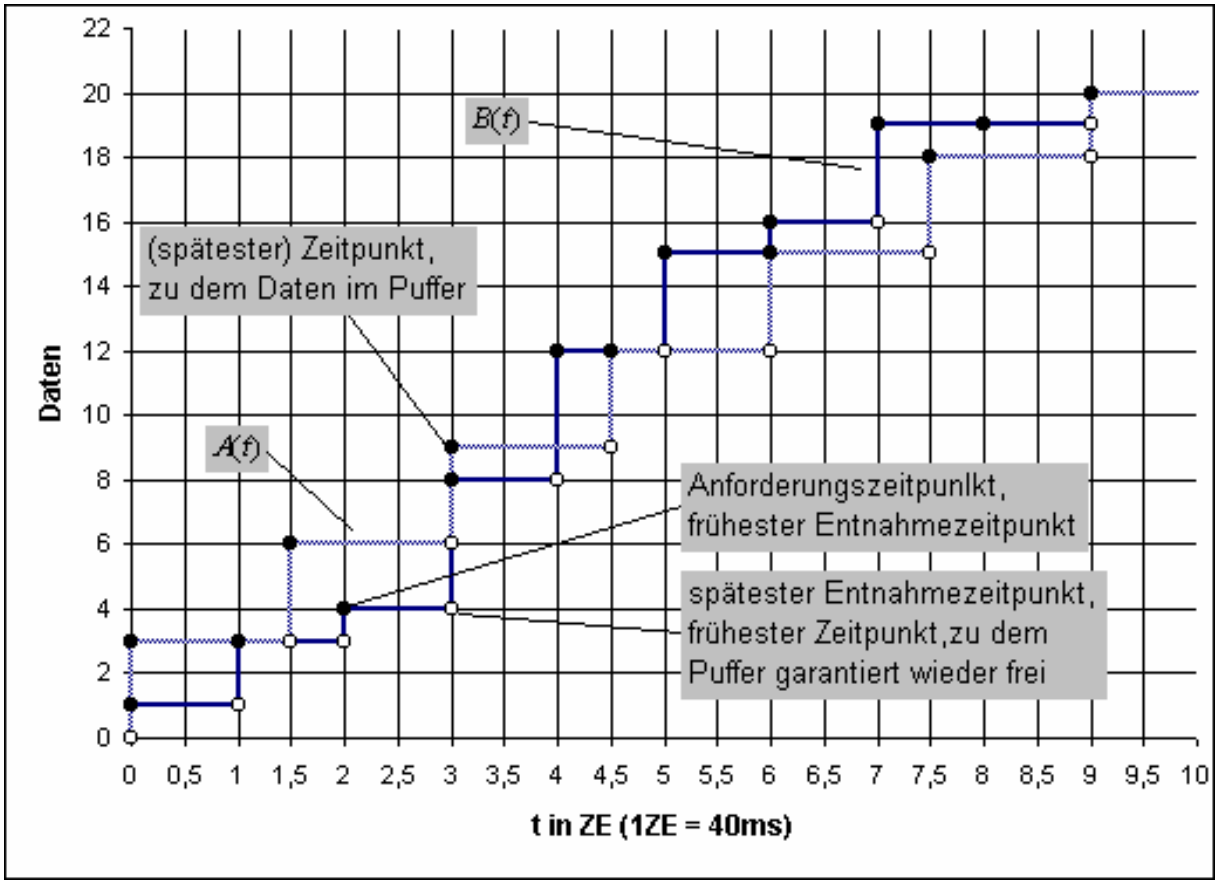

Abb. 8 Angebots- und Bedarfsfunktion

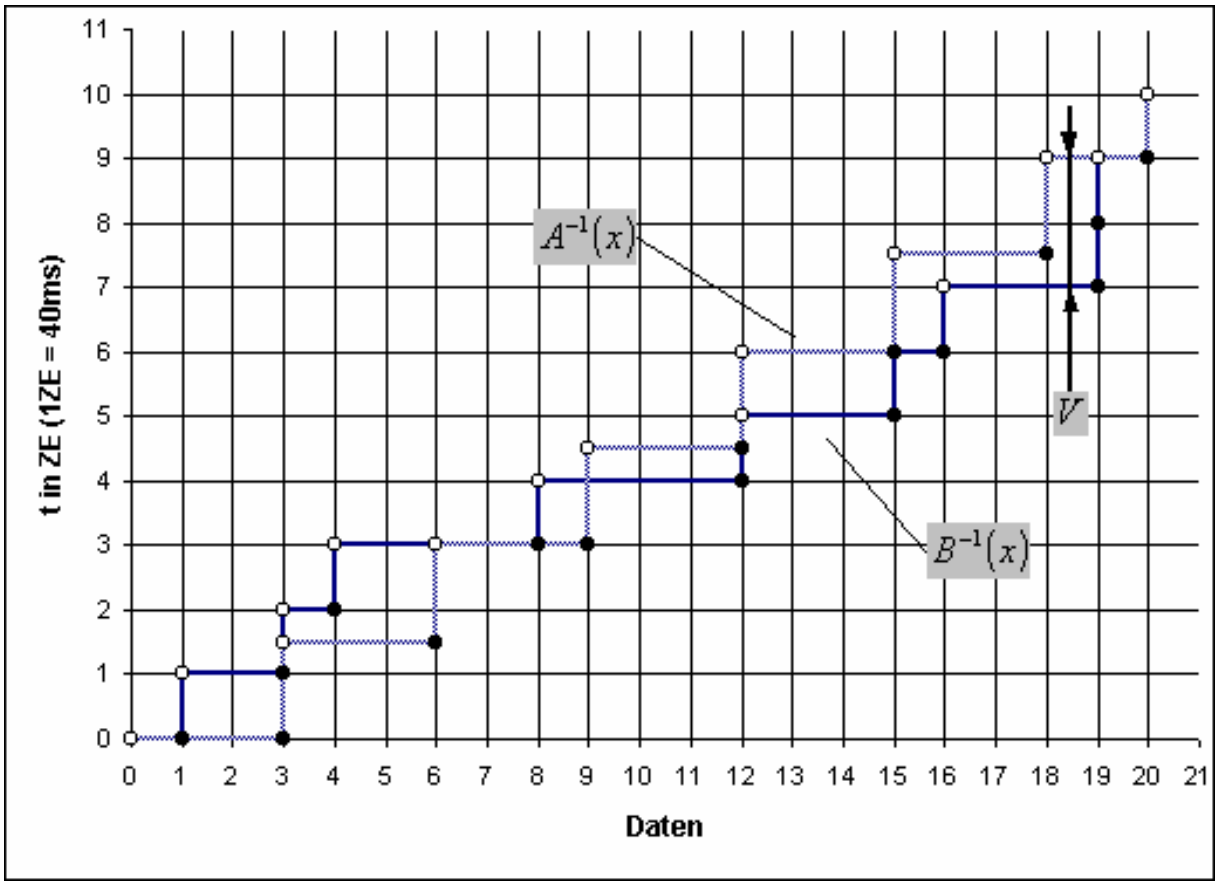

Abb. 9 Die entsprechenden Umkehrfunktionen

$$A^{-1}(x)=\begin{cases} iT_A & \text{für} \quad x\in(iQ,(i+1)Q] \\ \lceil m-1\rceil T_A & \text{"} \quad x\in(\lceil m-1\rceil Q, D] \end{cases}$$

$$B^{-1}(x)=jT_B \quad \text{für } x\in(S_j,S_{j+1}],\ j=0,\ldots,n-1$$
$$\text{wobei } S_0:=0,\ (x,x]:=\emptyset$$

A^{-1} und B^{-1} sind rechtsstetige Treppenfunktionen mit den Vielfachen von Q bzw. den Werten D_j als Sprungstellen und den Perioden T_A bzw. T_B als Sprunghöhen. $A^{-1}(x)$ drückt aus, wann das Datum x eingelesen ist (Pufferplatz belegt), entsprechend $B^{-1}(x)$ wann es bereitstehen muß.

Schließlich sei

$$\Delta := A^{-1}-B^{-1}.$$

Ist die Differenz $\Delta(x)$ positiv, so bedeutet dies, daß der Zeitpunkt des Einlesens (Füllens) von Datum x größer ist, das Einlesen also später erfolgt als der des Entnehmens von x. Die Differenz dieser Ordinaten für den Wert x ist gleich dem Abstand der Strecken der Funktionen A und B in Höhe von x, die deren Sprungstellen senkrecht,

also in *x*-Richtung, miteinander verbinden, die aber im strengen Sinne nicht zu den Treppenfunktionen A und B gehören.

In den Bereichen (Intervallen der *t*-Achse), in denen A rechts von B liegt, treffen die Daten zu spät ein, und die Verspätung ist dort am größten, wo dieser Abstand am größten ist (s. Abb. 11). Wird daher die Funktion A um diesen Abstand V nach links, in Richtung negativer *t*-Werte verschoben, so treffen nunmehr *alle* Daten um die Zeit V früher und damit rechtzeitig ein; ist die Verschiebung geringer als V, so wird zumindest ein zu V gehöriges Datum x mit $\Delta(x) = V$ zu spät eingelesen. Damit ergibt sich zusammengefaßt als Mindestvorlauf für EP (s. Abb. 10):

$$\boxed{V = \max_{x\in[0,D]} \Delta(x)} \tag{15}$$

und es ist

$$\hat{A}(t) := A(t+V), \qquad t \in \mathbf{R}$$

die endgültige Angebotsfunktion (s. Abb. 11). Es gilt offenbar:

$\hat{A}(t) \geq B(t)$	$\forall t \in \mathbf{R}$	Puffer stets ausreichend gefüllt
$\hat{A}^{-1}(x) \leq B^{-1}(x)$	$\forall x \in [0,D]$	alle Daten rechtzeitig eingelesen

- ***Bemerkung 1.*** In der Regel wird $\Delta(t)$ mindestens gleich 0 sein (zumindest die zuerst eingelesenen Daten werden auch sofort benötigt). Sollte sich $V < 0$ ergeben, so bedeutet dies eine „Reserve“. EP könnte dann auch um $|V|$ später gestartet werden.
- ***Bemerkung 2.*** Ein Übergang von VP in den Zustand „wartend“ kann auch ausgeschlossen werden durch Verschieben von $A(t)$ nach oben oder um die Zeit V' nach links, wobei V' die Länge des längsten Intervalls I mit $A(t) - B(t) \leq 0 \;\; \forall t \in I$ ist; beides führt i.a. zu größerem Vorlauf bzw. zu größerem P_0 und $P_{\min}$.

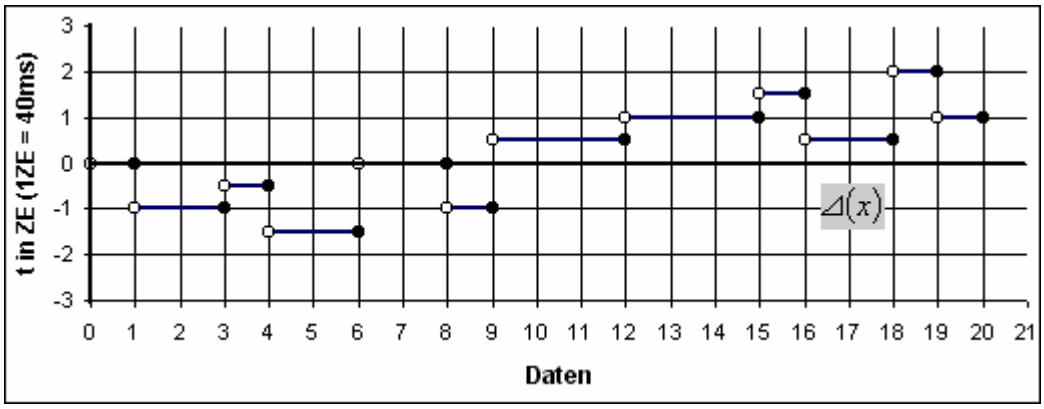

Abb. 10 Bestimmung von V, $\max \Delta(x) = 2$

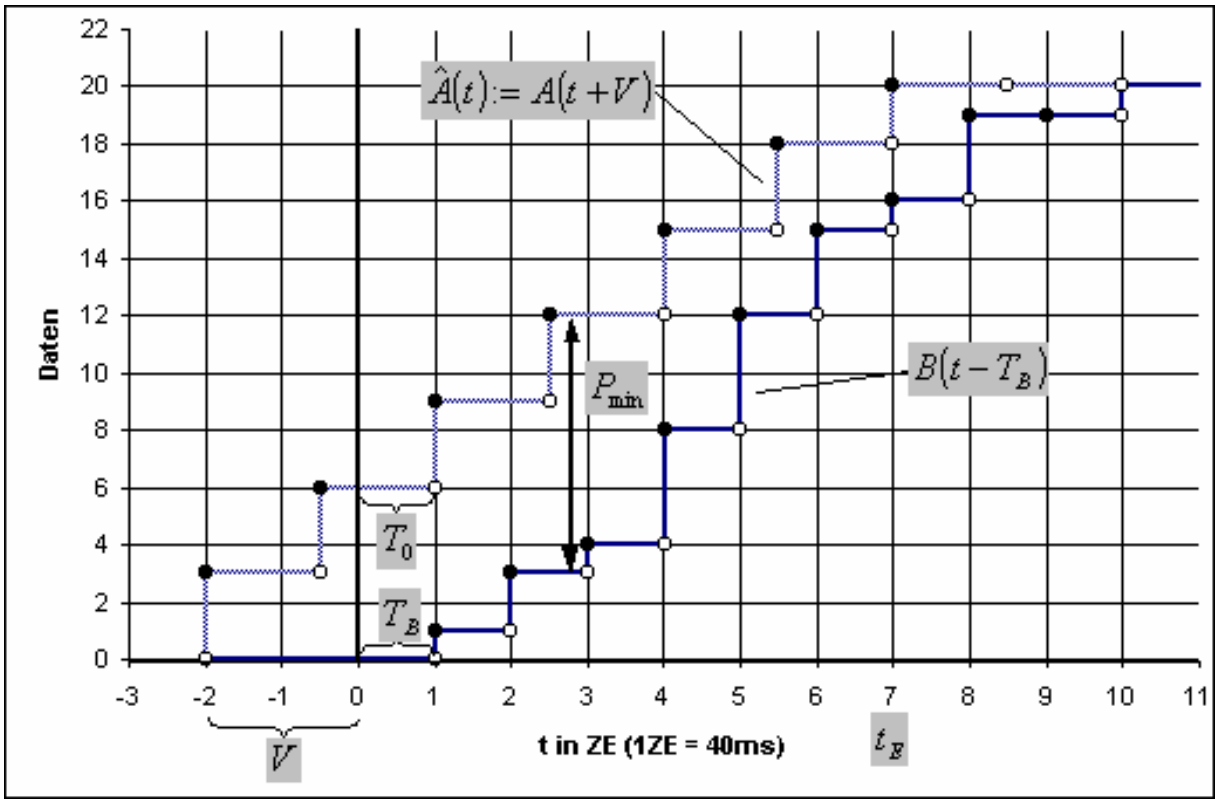

Abb. 11 Die Auswirkung von Vorlauf V

2.2.3 Berechnung der Anfangsfüllung P_0 und Vorperiode T_0

Es gilt (s. Abb. 11; man beachte den Sonderfall $\frac{V}{T_A} \in \mathbf{N}$):

$$P_0 = \left(\left\lfloor \frac{V}{T_A} \right\rfloor + 1 \right) \cdot Q \tag{16}$$

$$T_0 = \left\lceil \frac{V}{T_A} \right\rceil \cdot T_A - V$$

$$t_E = \lceil m - 1 \rceil \cdot T_A - V \tag{17}$$

t_E: Zeitpunkt des letzten Füllens, Ende von EP.

2.2.4 Berechnung des kleinsten mindestens erforderlichen Puffers P_{min}

Es ist ersichtlich, daß sich dieser Puffer aus der größten Differenz von Füll- und Entnahmefunktion ergibt:

$$P = \max_t \left(\hat{A}(t) - B(t) \right).$$

Allerdings setzt dies voraus, daß die Entnahme „schnell genug“ geschieht. Im ungünstigsten Fall ist die Entnahme aber erst zum Ende eines Intervalls von VP beendet. Dies bedeutet eine Verschiebung von $B(t)$ um die Periodenlänge T_B von VP nach rechts (s. Abb. 11). Damit folgt:

$$P_{\min} = \max_{teR} \left(A(t + V) - B(t - T_B) \right) \tag{18}$$

Dabei ist unterstellt, daß die Zeitpunkte des Füllens exakt eingehalten werden, ansonsten ist eine weitere Verschiebung von A um T_A erforderlich.

Die für die Implementation dann erforderlichen Werte für P_0', T_0', t_E' ergeben sich unmittelbar aus (16) und (17), indem dort V durch $V + T_A$ ersetzt wird.

2.2.5 Berechnung von Anzahl und Gültigkeitsdauer der Pufferbereiche

Wie eingangs erwähnt, füllt EP „irgendwann“ Pufferbereiche der Größe Q mit Daten und erklärt diese Bereiche nacheinander im konstanten Abstand T_A als gültig für VP. Nach einer bestimmten Zeit (Gültigkeitsdauer T_Q) werden diese Bereiche VP wieder entzogen, wenn die Daten nicht mehr benötigt werden. Außerdem benutzt EP in jedem Fall vollständige Pufferbereiche der Größe Q, auch wenn $P_{\min}$ dies nicht erfordern würde ($P_{\min} = 7$ bei $Q = 3$); es sei n_Q deren kleinste Anzahl, die gewährleistet, daß $P_{\min}$ darin gespeichert werden kann. Dann gilt:

$$n_Q = \left\lfloor \frac{P_{\min} - 2}{Q} \right\rfloor + 2$$
$$T_Q = n_Q \cdot T_B \qquad (19)$$

Dies liegt darin begründet, daß die Bereiche linear gefüllt und ebenso – lediglich zeitlich versetzt – geleert werden. Damit befinden sich die $P_{\min}$ gültigen Daten zusammenhängend im ungünstigsten Fall in der angegebenen Zahl von Bereichen.

Abb. 12 Der Puffer minimaler Größe

Bsp. $P_{\min} = 9$, $Q = 3$

2.3 Modell 2 – Beschreibung von VP durch Rate und maximale Schwankung

Für VP mögen jetzt nicht mehr die detaillierten Informationen (D_j), $j=1,...,n$ zur Verfügung stehen, sondern, VP sei beschrieben durch die mittlere Datenrate R sowie die maximale obere und untere Abweichung B^+, B^- der aktuell angeforderten Datenmenge vom Durchschnitt.

2.3.1 Voraussetzungen - Begriffe – Bezeichnungen

Strom	-	Folge von Ereignissen (Erzeugen/ Verbrauchen eines Datums, einer Dateneinheit)
$B(t)$	-	Anzahl der bis zum Zeitpunkt t insgesamt eingetretenen Ereignisse von VP (Gesamtanzahl der angeforderten Daten)
$A(t)$	-	analog
$R(t)=\frac{B(t)}{t} \quad (t>0)$	-	aktuelle Rate
$D>0$	-	Gesamtanzahl der Daten
T_{ges}	-	Gesamtzeit, hier speziell: $T_{\text{ges}} = nT_B, \quad n, T_B$ wie in Modell 1
$R=\frac{D}{T_{ges}}>0$	-	mittlere Rate von VP

Weiter sei (s. Abb. 13):

$$\overline{B}(t) := Rt$$
$$B^+ := \max_t\left(B(t)-\overline{B}(t)\right)$$
$$B^- := \min_t\left(B(t)-\overline{B}(t)\right)$$

Wir betrachten ein anderes Beispiel mit folgenden Werten:

$Q = 3$, $T_B = 40$ ms, $n = 10$, $D = 20$, $(D_j) = 1, 1, 1, 1, 5, 5, 1, 1, 3, 1$

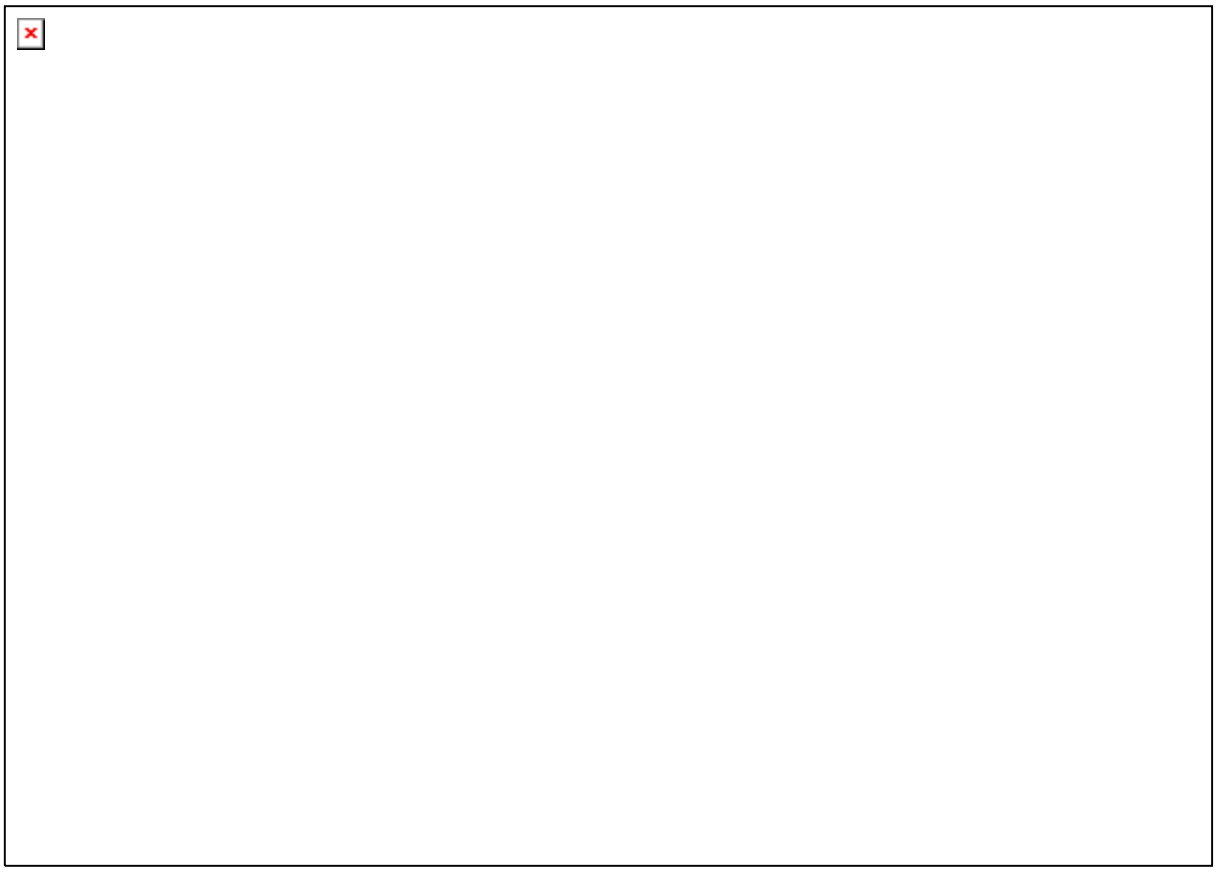

Abb. 13 Die Bedarfsbeschreibung

2.3.2 Berechnung von *V*

EP muß soviel vorher starten, daß unabhängig vom konkreten Verlauf von $B(t)$ innerhalb des Streifens $\left[\overline{B}(t)-B^{-},\ \overline{B}(t)-B^{+}\right]$ alle Daten rechtzeitig vorhanden sind, bzw. alle Ereignisse rechtzeitig eingetreten sind. Es sei

$$B_o(t) := \overline{B}(t) + B^{+} = Rt + B^{+}.$$

Aus $B_o(t) = 0$ folgt

$$t_0 = -\frac{B^{+}}{R}$$

und damit

$$\boxed{V = |t_0| = \frac{B^{+}}{R}} \qquad (20)$$

Anfangsfüllstand und Phasenverschiebung(Vorperiode) ergeben sich wie in Modell I.

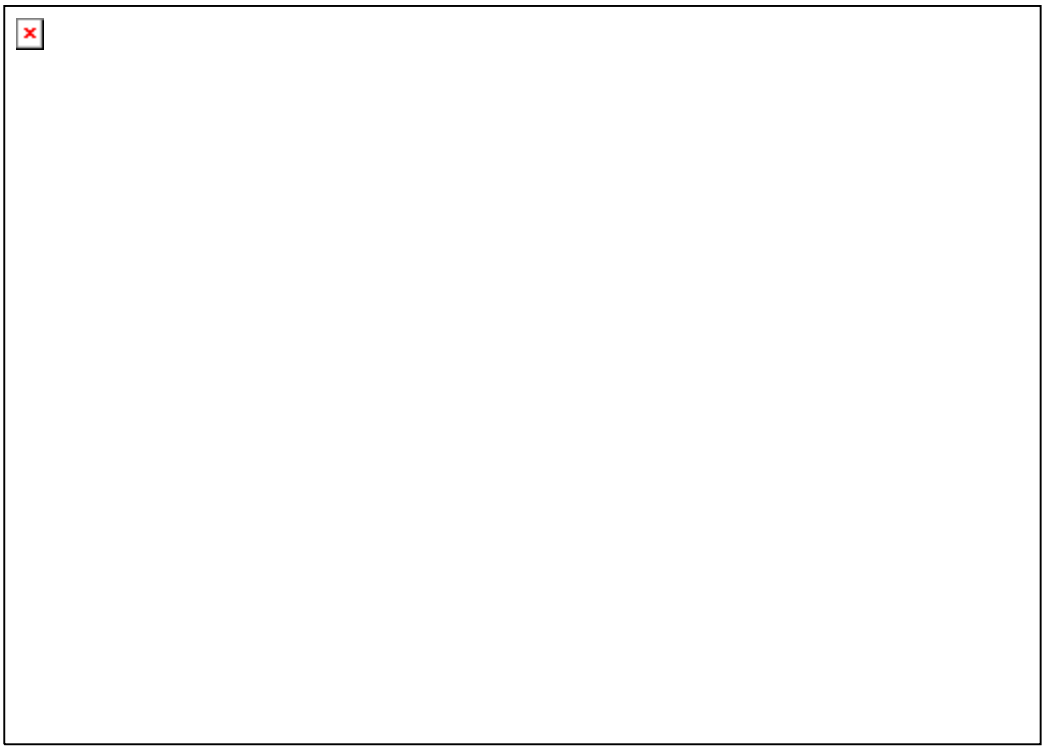

Abb. 14 Angebotsfunktion und $B_o(t)$

2.3.3 Berechnung des erforderlichen Puffers P_1

genauer: eines mit Sicherheit ausreichenden und möglichst kleinen, aber nicht notwendig des kleinsten Puffers

P_1 ergibt sich (s. Abb. 15) aus dem Abstand a zwischen der Geraden $A_o(t)$, sie begrenzt die Angebotsfunktion $A(t)$ nach oben, und der Geraden $B_u(t)$, die die um T_B nach rechts verschobene Bedarfsfunktion $B(t)$ nach unten begrenzt (vgl. Berechnung von $P_{\min}$ in Modell 1). Genauer, P_1 ist die größte ganze Zahl, die kleiner als *a* ist. Denn ist $a \notin \mathbf{N}$, so ist $P_1 = \lfloor a \rfloor$, da die Differenz zwischen $A(t)$ und $B(t)$ als Treppenfunktionen mit ganzzahligen Werten selbst ganzzahlig sein muß. Ist $a \in \mathbf{N}$, so ist $P_1 = a - 1$ wegen der Linksstetigkeit von $A(t)$ und $B(t)$.

Nun ist offenbar

$$A_o(t) = Rt + B^+ + Q$$
$$B_u(t) = Rt + B^- - 2RT_B$$

Damit folgt:

$$P_1 = \lceil A_o(t) - B_u(t) - 1 \rceil,$$

also

$$P_1 = \left| B^+ - B^- + Q + 2RT_B - 1 \right| \qquad (21)$$

(man beachte, daß i.a. $B^- < 0$ ist) und $P_{min} \leq P_1$.

Formel (19) gilt analog.

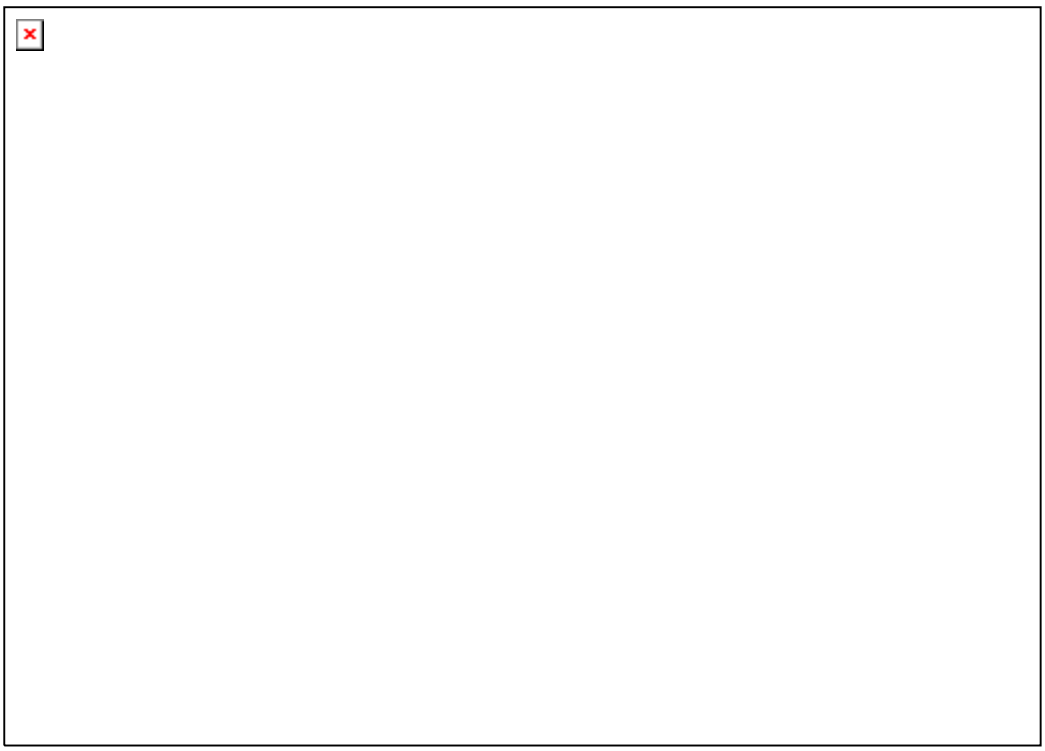

Abb. 15 Daten zur Pufferberechnung

3 Implementierung des Berechnungsprogramms

Zur Implementierung wurde sich für Microsofts Visual C++ 5.0 entschieden, weil es weite Verbreitung findet und leicht zu bedienen ist. Der integrierte Debugger leistet gute Arbeit und läßt sich komfortabel einrichten sowie handhaben. Das grafische Frontend des zu erstellenden Programms ist nicht Schwerpunkt der Arbeit. Es geht vielmehr darum, daß die darunterliegenden Funktionen modular und leicht nachvollziehbar programmiert werden. Um dieser Forderung gerecht zu werden, hielt ich mich an die Termini der mathematischen Modelle. Man kann also leicht intuitiv die Bedeutung der verschiedenen Variablen- und Funktionennamen interpretieren.

3.1 Globale Variablen, Konstanten, Funktionen

Um sich schnell im Quellcode zurechtzufinden, ist hier eine Zusammenstellung aller wichtigen Bezeichner angegeben:

Konstanten	
MAX_ITEM_STR	Maximale Zeichenkettelänge für eine Zahleneingabe, Element von (D), in dem Eingabefeld
MAX_ITEM	Maximale Anzahl der Datenpackete, Datenpackete
MAX_BUFFER	Maximale Pufferlänge für Formatumwandlungen und String – Einlesen
Variablen	
DSFolge[MAX_ITEM]	Summenfolge der Folge der Datenpackete, im Modell mit S bezeichnet, das erste Element mit Index 0 hat auch immer den Wert 0, nur bei manueller Methode benutzt
Dgesamt	Summe aller Datenpackete, Gesamtdatenvolumen
anz_item	Anzahl der Datenpackete
inputQ	eingegebener oder eingelesener Wert für Q (Quantum)
inputTB	eingegebener oder eingelesener Wert für T_B (Periode des Verbraucherprozesses)

inputDatei[_MAX_PATH]	vollständiger Name der Datei mit den einzulesenden Werten, (Pfad + Dateiname)
inputDateiname[_MAX_PATH]	nur der Name dieser Datei
resultm	errechneter Wert für m
resultTA	Ergebnis für die Periode des Erzeugerprozesses
resultV	Ergebnis für den Vorlauf der Angebotsfunktion
resultT0	errechneter Wert für die Vorperiode T_0
resultP0	Ergebnis für den Anfangsfüllstand des Puffer zum Zeitpunkt des Startens des Verbraucherprozesses
resultPmin	Ergebnis für die minimale Größe des Puffers
HauptFunktionen	
CalculateTA_MM1	MM1 bedeutet „Manuelle Methode 1“. Manuell, weil die Werte der Datenpacketgrößen von Hand eingegeben werden. Belegung der globalen Variablen für die Periode des Erzeugerprozesses, resultm, resultTA. Voraussetzung: inputQ, inputTB
CalculateV_MM1	Wert für resultV wird berechnet und zugewiesen. Voraussetzungen: inputQ, inputTB, resultTA
CalculatePmin_MM1	Ermittlung der minimalen Größe des Puffers, Belegung von resultPmin. Voraussetzungen: inputQ, inputTB, resultTA, resultV
CalculateT0	Belegung von T_0 Voraussetzung: resultTA, resultV
CalculateP0	Belegung von P_0 Voraussetzung: inputQ, resultTA, resultV Berechnung von T_0 und P_0 sind bei beiden ersten Modellen, manuell oder über Datei, gleich.
CalculateBplusBminusM	Errechnung und Zuweisung der Werte für resultBplus sowie resultBminus („M“ - manuelle

	Eingabe der Ausgangswerte)
CalculateV_MM2	Belegung von resultV
CalculatePmin_MM2	Berechnung von resultPmin nach der 2.Methode, manuelle Eingabe

Die dazu analog existierenden Funktionen für das Arbeiten mit einer Eingabedatei seien hier noch kurz genannt. Sie unterscheiden sich nur sehr wenig in ihrer Implementation von den oben erklärten. „DM1“ zum Beispiel, steht für „Dateimethode 1“, und so erkennt man auch, welche Routine für welche Aufgabe zuständig ist.

- CalculateTA_DM1
- CalculateV_DM1
- CalculatePmin_DM1
- CalculateBplusBminusD
- CalculatePmin_DM2
- CalculateV_DM2

3.2 Funktionen zur Berechnung von Ausgangsgrößen

(1) long AngebotM(long t)
(2) long BedarfM(long t)
(3) long AObenMinusEinsM(long x)
(4) long BObenMinusEinsM(long x)
(5) long DeltaM(long x)
(6) float BquerM(long t)

(7) long AngebotD(long t)
(8) long BedarfD(long t)
(9) long AObenDinusEinsM(long x)
(10) long BObenMinusEinsD(long x)
(11) long DeltaD(long x)
(12) float BquerD(long t)

(13) double GetDSF(long i)

Die ersten beiden Funktionen erwarten ihre ganzzahligen Argumente vom Betrag in Millisekunden (als gedankliches Modell). Da Millisekunde eine gute Größenordnung für diese Problematik darstellt, habe ich auch dieses Einheitenformat auf der Dialogfläche vermerkt. Tatsächlich ist hier aber kein wirklicher Einheitenzwang vorhanden, so daß man auch eine andere Größenordnung verwenden kann, an die man sich dann aber auch strikt halten muß und die sich natürlich auch auf die Ergebnisse auswirkt. Gebrochene Zeitwerte dürfen nicht eingegeben werden, obwohl das keine direkten Fehler erzeugt. Im Hintergrund berechnete Zwischenergebnisse könne einen Nachkommaanteil besitzen; an entsprechender Stelle wird aber stets so auf- oder abgerundet, daß man bei den Endergebnissen immer auf der sicheren Seite ist. Alle aufgeführten Funktionen entsprechen, intuitiv nach ihrem Namen zu urteilen, den Funktionen aus den mathematischen Grundlagen.

(3) und (4) haben mit den Argumenten x jeweils eine Datenmenge als Eingangsgröße. (5) bedient sich dieser beiden Funktionen; der Eingabeparameter wird durchgereicht. Der Wert x gibt eine Datenmenge in Bytes an. Die Funktion (6) schließlich wird für die Berechnung der Geraden Bquer im zweiten Modell benötigt. Ihr Argument ist ein ganzzahliger Zeitwert mit der oben erklärten Einheit .

(7) bis (12) sind die Pendants für die Arbeit mit einer Datei als Eingabemöglichkeit für die Eingangsgrößen. Manche Funktionen sind ihren Partnern aus der manuellen Eingabeverarbeitung identisch. Eine klare Trennung bringt aber erheblich mehr Überblick.

Kernstück des zweiten Weges ist die Funktion (13). Sie ist für den wahlfreien Zugriff auf die Summenfolge der Datenpacketfolge verantwortlich. Als Argument übergibt man einfach den Index des gewünschten Elements, muß aber auf dessen Wertebereich achten, der mit [0,anz_item] festgelegt ist. Das Element der Summenfolge mit dem Index Null besitzt per Definition den Wert Null. Diese Funktion nutzt eine Datei mit dem Namen „daten.sum“ (Erzeugung beim Dateneinlesen). Diese Summenfolgendatei kann unbedenklich nach getaner Arbeit gelöscht werden.

Damit sind alle mathematischen Grundfunktionen abgehandelt. Für weitere Details sei auf den Quelltext im Anhang verwiesen (s. Anhang).

3.3 Ereignisbehandlung für den Dialog

3.3.1 Überblick und Semantik

Die hauptsächliche Programmiertätigkeit besteht in einem ereignisbasierten System, in unserem Falle Microsoft Windows, aus der Belegung und Implementation der den Ereignissen zugeordneten Behandlungsroutinen.

Ein Beispiel: Betätigt man die linke Maustaste über einer Schaltfläche, dann löst das System intern eine Nachricht aus. Möchte man nun, das nach diesem Klick etwas Spezifisches abläuft, programmiert man eine Behandlungsroutine für genau diese Nachricht aus.

Im Visual C++ hilft einem dabei der Klassenassistent. Mit ihm lassen sich innerhalb einer Anwendung komfortabel solche Ereignishandler erstellen und insgesamt verwalten. Im folgenden sind nun die Handler aufgeführt und erklärt, die von mir implementiert wurden. Wie man sieht, sind alle Funktionen Memberfunktionen des Dialogobjekts CCalcmmDlg.

(1) void CCalcmmDlg::OnKillfocusDatenpackete()
(2) void CCalcmmDlg::OnKillfocusQuantum2()
(3) void CCalcmmDlg::OnKillfocusTB1()
(4) void CCalcmmDlg::OnManuM1()
(5) void CCalcmmDlg::OnManuM2()
(6) void CCalcmmDlg::OnDateiM1()
(7) void CCalcmmDlg::OnDateiM2()
(8) void CCalcmmDlg::OnDateiOut1()
(9) void CCalcmmDlg::OnDateiOut2()
(10) void CCalcmmDlg::OnDateiLaden()
(11) void CCalcmmDlg::OnClearAll()

Das Ereignis „Killfocus“ wird für ein Steuerelement ausgelöst, sobald es den Status „aktuelles Steuerelement“ verliert. Man „hangelt“ sich durch die einzelnen, auf dem Dialog liegenden Elemente, indem man die Tabulatortaste betätigt. Betroffen sind aber nur solche Elemente, die auch vom Programmierer dafür freigeschaltet wurden.

Auf so ein Ereignis wird mit den ersten drei Funktionen reagiert. In den anderen Routinen wird ein Klick auf die jeweilige Schaltfläche behandelt.

3.3.2 Erläuterung der Funktionalität

Prozedur (1) errechnet aus den eingegeben Datenpacketen deren Anzahl und Summe. Die Ergebnisse werden den globalen Variablen anz_item und Dgesamt zugewiesen. Weiterhin wird die Summenfolge der Folge der Datenpackete berechnet und in dem eindimensionalen Feld DSFolge abgelegt. Damit kann man später leicht auf deren Elemente zugreifen.
Die Prozedur bringt die errechneten Ergebnisse auch in den dafür sich auf dem Dialog befindlichen Ausgabefeldern zur Ansicht. Alle diese Ausgabefelder sind graublau unterlegt als Kennzeichnung, daß sie einen Schreibschutz besitzen. Nebenan kommen hauptsächlich die Bibliotheksfunktionen zur Umwandlung von Datentypen zur Anwendung.

Die Prozeduren (2) und (3) sind für die Belegung der globalen Variablen inputQ und inputTB verantwortlich. Sie wandeln die eingegebenen Werte vom Typ String in Zahlen um und nehmen dann die Zuweisung vor. Damit stehen diese beiden Eingangsgrößen allen anderen Funktionen zur Verfügung.

Die restliche Prozeduren (4) - (11) werden als direkte Folge einer Schaltflächenbetätigung gestartet. (4) und (5) berechnen die Endergebnisse nach dem ersten und zweiten mathematischen Modell mit manueller Eingabe und stellen diese auf dem Dialog dar. (6) und (7) arbeiten analog für die Eingabe über eine Datei. Welche Schaltflächen dafür betätigt werden müssen, ist intuitiv ersichtlich.

(8) wird aufgerufen, wenn man auf die Schaltfläche mit der Aufschrift „manuell.out schreiben“ linksklickt. Wie der Titel schon sagt, macht diese Funktion nur Sinn in Verbindung mit dem manuellen Verfahrensweg, der auf der linken Hälfte des Dialoges zu finden ist. Sind hier alle Eingaben vollständig und die Endergebnisse jeweils nach Modell1 oder Modell2 berechnet (angezeigt), wird somit eine Datei mit dem obengenannten Namen im aktuellen Verzeichnis erzeugt bzw. neu beschrieben. Das Aussehen des Dateiinhaltes ist in Kap.3.5 erläutert.

Durch Prozedur (9) wird ebenfalls eine Ausgabedatei erstellt. Sie trägt den Namen der Eingabedatei mit geänderter Dateierweiterung, nämlich „.out“. Das ist nur sinnvoll am Ende des zweiten Weges (Eingangsdaten aus Eingabedatei). Man kann sie leicht mit einem einfachen Editor betrachten; es handelt sich um eine ASCII – Datei. Das Inhaltsformat finden Sie in Kap. 3.5.

(10) steht am Anfang des zeiten Weges, aufgezeigt auf der rechten Seite des Dialoges. Diese Prozedur ist mit der Schaltfläche „Datei öffnen und auswerten“ verknüpft. Wie dieser Name schon verrät, werden hier neben dem Laden der Daten auch noch Berechnungen ausgeführt. Eingelesen werden alle notwendigen Eingangsgrößen für die anstehenden mathematischen Berechnungen. Aus der sich in der Datei befindlichen Datenpacketfolge wird deren Summenfolge gebildet und in der Summenfolgendatei „daten.sum“ vermerkt.

Prozedur (11) löscht die Inhalte aller sich auf dem Dialogfenster befindlichen Felder und setzt alle für die Berechnung wichtigen globalen Variablen des Programms auf den Wert Null zurück. Man benutzt die dafür verantwortliche Schaltfläche „Feldinhalte löschen und globale Variablen auf Null“ vor dem Beginn einer weiteren Berechnung, und zwar zwingend notwendig, denn, wenn man den linearen Berechnungs- und Eingabeablauf nicht einhält, kommt es auf jeden Fall zu falschen Ergebnissen. Im schlimmsten Fall kann das Programm in eine Endlosschleife gelangen. Man schafft sich also somit einen bereinigten Ausgangsstatus des Programms.

3.4 Vorhandenes Dateiformat für das Dateneinlesen

Ein Beispeil:

```
###########
20
10
3
40
###########
1
1
1
1
5
5
1
1
3
1
```

Abb. 16 name.txt als Eingabedatei

Erklärung:
Die Beiden Doppelkreuzlinien schließen die Eingangswerte ein. Dabei gilt folgende Reihenfolge:

Dgesamt	Gesamtdatenvolumen in Byte oder einer anderen kleinsten Einheit
anz_item	Anzahl der Datenpackete, die dann im folgeneden auch aufgelistet sind
InputQ	Größe des Quantums in Byte oder einer anderen kleinsten Einheit in Übereinstimmung mit Dgesamt
InputTB	Periode des Verbraucherprozesses in Millisekunden oder einer anderen Zeiteinheit, die dann als Dimension für weitere Ergebnisse maßgeblich ist

Dann folgen die Datenpackete in ihrer chronologischen Reihenfolge. Es ist zu beachten, das nur so viele Werte hier eingelesen werden, wie in anz_item vermerkt sind. Sollten es zuwenig sein, zieht das Programmfehler nach sich. Die Datei ist eine normale Textdatei mit der Erweiterung „txt“. Sie läßt sich also leicht mit einem Editor erstellen. Ebenso einfach gestaltet sich die Erzeugung durch ein anderes Programm.

3.5 Ausgabedateiformate

Als Vorbemerkung möchte ich nur kurz darauf hinweisen, daß das Format der Summenfolgendatei „daten.sum“, die ja auch eine Ausgabedatei darstellt, keine Darstellungen und Betrachtungen außer einer Hexadezimalanalyse zuläßt. In dieser Datei sind Werte vom Typ double aneinandergereiht abgelegt. Da sie für interne Zwecke verwendet wird, ist sie für den Benutzer nicht weiter von Bedeutung.

```
####Eingabe##########
D =20.
n =10.
Q =3.
TB=40.
#####Packetfolge#####
1
2
1
4
4
3
1
3
0
1
####################
#####Ergebnisse######
TA=
Pmin=13.
V=100.
P0=
T0=
```

Abb. 17 manuell.out

Diese Datei kann man wahlweise nach einem Berechnungsweg mit manueller Eingabe erzeugen lassen. Sie hat das strukturelle Aussehen, wie es in Abb. 17 dargestellt ist. Die einzelnen Abschnitte sind auch benannt. Die ersten beiden Bereiche geben noch einmal die Eingabedaten wieder und am Ende finden wir die zugehörigen Ergebnisse. Bei Methode 1 sind alle Ergebnisse mit einem Wert versehen, bei Methode 2 nur Pmin und V, wie wir es in Abb. 17 sehen können. Möchte man diese Ausgaben behalten und archivieren, so kann man die Datei entweder umbenennen oder an eine anderen Ort kopieren.

```
TA=60.
Pmin=9.
V=80.
P0=6.
T0=40.
```

Abb. 18 name.out

Die in Abb. 18 gezeigte Beispieldatei kann man am Ende des zweiten Berechnungsweges erzeugen lassen. Der Inhalt entspricht strukturell exakt dem Ergbnisbereich der Datei „manuell.out“. Die Bedeutung ist selbsterklärend.

4 Zusammenfassung

Wie wir gesehen haben, sind die Mechanismen für eine fehlerlose Übertragung von Multimediadaten stark von deren Format abhängig. Es leuchtet jedem ein, das ein kontinuierlicher Strom, wie er bei Audiodaten charakteristisch ist, ganz andere Kriterien in die Berechnung einbringt als ein Videostrom, bei dem nur der Differenzdatenbetrag zum vorherigen Bild übertragen werden muß.

Grob wird also zwischen zwei Formen von Verbrauchsfunktionen unterschieden. Auf der einen Seite haben wir eine Geradenfunktion mit dem Anstieg m>0 und auf der anderen Seite hat die Verbrauchsfunktion eine Treppenform. Im letzteren Falle sind es meist komprimierte Daten, die im allgemeinen Fall packetweise dekodiert werden. Genau an den Grenzen dieser Packetgrößen hat die Verbrauchsfunktion dann ihre Sprungstellen.

Auch was den Graph der Erzeuger- bzw. Angebotsfunktion anbelangt, sind große Unterschiede festzustellen. Hier wird ebenfalls hauptsächlich zwischen kontinuierlichen und Treppenfunktionen unterschieden. Dem aufmerksamen Leser wird nicht entgangen sein, daß man sich in nahezu allen Fällen der Berechnung auf eine mittlere Rate der Datenanlieferung stützt und damit eine Abstraktion erreicht.

Als Abschluß möchte ich noch einen Ausblick geben. Stellen Sie sich einen tragbaren CD-Spieler vor, der, um die Aussetzer der Wiedergabe bei einer Erschütterung auszugleichen, einen internen Puffer fester Größe besitzt. Nun könnte man meinen, je größer dieser Puffer ist, desto besser. Aber überlegen wir einmal etwas tiefgründiger. Ist dieser Puffer sehr groß, dauert es auch lang bis er gefüllt ist. Es kommt zu Verzögerungen beim Start der Wiedergabe. Nun könnte man meinen, es wäre besser die Lesegeschwindigkeit zu erhöhen und den Puffer zu verkleinern. Erhöhte Lesegeschwindigkeit bedeutet aber in genau diesem Fall erhöhte mechanische Anfälligkeit, als Vertreter der allzu beliebten Störgröße Z; ein unendliches Spiel also.

Was ich damit ausdrücken will ist, daß man in Zukunft Wahrscheinlichkeiten mit in die Berechnungen einschließen muß. Die Wahrscheinlichkeit, daß man das Gerät sehr lange schüttelt, ist eben bedeutend geringer als nur eine kürzere Rüttelzeit. Um

also bei der Ressourcenvergabe den „golden“ Mittelweg zu finden, bedarf es genauer Kalkulation.

Beleuchtet man andere Problemgebiete, kann man sagen. schnelle Speichermedien verlieren ihre Vorzüge, sobald viele Nutzer gleichzeitig darauf zugreifen. Bei Netzwerken verhält es sich analog. Es hat also keinen Sinn, sich darauf zu verlassen, die Hardware würde eines Tages so ein hohes Niveau aufweisen, daß man solche Überlegungen nicht mehr anstellen braucht. Wahrscheinlichkeiten und Durchschnittsrechnung werden mehr und mehr eine Rolle spielen, denn eine Datenbereitstellung mit garantierten Eckdaten ist nur in den seltensten Fällen möglich. Eine Ausnahme stellen nur spezielle Systeme dar.

5 Anhang

5.1 Quellenverzeichnis

Das Verzeichnis enthält neben den Angaben über die diesem Beleg zugrunde liegenden Quellen auch weiterführende und aktuelle Werke zu den behandelten Themenfeldern

[1] Tanenbaum: *Moderne Betriebssysteme*. 1. Aufl. München: Prentice Ha Internat., 1995.

[2] Jim Gemmell, Stavros Christodoulakis: *Principles of Delay-Sensitiv Multimedia Data – Storage and Retrieval.* ACM, Vol. 10, Nr. 1, Jan 1992, S. 50-90

[3] David P. Anderson: *Metascheduling for Continuous Media.* ACM, Vol. 11, Nr. 3 Aug 1993, S. 226-252

[4] David P. Anderson, Yoshitomo Osawa, Ramesh Govindan: *A File System for Continuous Media.* ACM, Vol. 10, Nr. 4, Nov 1992, S. 311-337

[5] D. P. Anderson, R. G. Herrtwich, C. Schäfer: *SRP: A ressource reservation protocol for guaranteed-performance communication in the Internet.* Tech. Rep. 90-006, National Computer Science Institute, Feb 1990*rinciples*

[6] A. Park, P. English: Proceedings of the International Conference on Multimedia Information Systems 1991 (Singapore, 16.-18. Jan 1991): *A variable rate strategy for retrieving audio data from secondary storage.* McGraw-Hill, New York 1991. S. 135-146

[7] A D. P. Anderson, R. Govindan, G. Homsy: Proceedings of the International Conference on Multimedia Information Systems 1991 (Singapore, 16.-18. Jan 1991): *Abstractions for continuous media in a network window system.* McGraw-Hill, New York 1991. S. 273-298

5.2 Internet – Recherche

5.2.1 World Wide Web - Links

- http://bmrc.berkeley.edu/projects/cmt/index.html
  - Welcome to the home of the *Berkeley Continuous Media Toolkit* (CMT) -- a portable, freely distributed software package for developing distributed multimedia applications. It is built on top of Tcl/Tk, a scripting language and graphical user interface toolkit, and Tcl-DP, which provides network tools included a remote procedure call package and a name server. CMT is freely distributed and has been ported to a several platforms.

- http://www.cs.huji.ac.il/~grishac/arch-papers.html#multi-media
  - *Implementation Techniques for Continous Media Systems and Applications*, B. C. Smith, PhD Thesis, UCSB
  - *RIVL: A Resolution Independant Video Language*, B. C. Smith, Submitted for publication.
  - *A Continous Media Player*, L. A. Rowe and B. C. Smith, 3-rd Int. Workshop on Network and OS Support for Digital Audio and Video, Nov., 1992
  - *Efficient Algorithms for Optimal Video Transmission*, Dexter Kozen, Yaron Minsky and Brian Smith, Computer Science Department, Cornell University, TR95-1517, May, 1995
  - *Cyclic-UDP: A Priority-Driven Best-Effort Protocol*, B. C. Smith, Submitted for publication.
  - *MPEG Video in Software: Representation, Transmission, and Playback* , L. A. Rowe, K. D. Patel, B. C. Smith and K. Liu, High Speed Networking and Multimedia Computing, IST/SPIE Symp. on Elec. Imaging Sci. & Tech., Feb., 1994

- http://www-users.cs.umn.edu/~yjwon/research.html – verschiedene Publikationen zu folgenden Themen
  - Application of Massive Scale Continuous Media Server
  - Stochastic Modeling of the Storage Hierarchies

- http://www.eecis.udel.edu/~bao/prjQoS.html
  - Dynamic Quality of Service (QoS) Control for Multimedia Network Applications

- http://hpc.ee.ntu.edu.tw/~murphy/reports/Study/DMQOSSurvey.html
  - ummfassend zu QoS
  - communication protocols
  - OSs
  - MM databases
  - file servers
  - and those directly affecting the human user

- http://www.cis.ohio-state.edu/~jain/index.html
  - mehr aus der Sicht der Netzwerkseite, QoS, kompletter Lehrstuhl

- Hayter93 MD Hayter. A Workstation Architecture to Support Multimedia. Technical Report 319, Cambridge University Computer Laboratory, November 1993. Ph.D. dissertation. thesis

- Hopper90 A Hopper. Pandora - an experimental system for multimedia applications. ACM Operating Systems Review, 1990. Hopper90

- Pasquale92 Joseph Pasquale. I/O System Design for Intensive Mutitmedia I/O. In Third Workshop on Workstation Operating Systems,, Key Biscayne, Florida, April 1992. Pasquale92

- Pfeifer93 T Pfeifer. Micronet Machines. New Architectural Approaches for Multimedia End-Systems. In Proceedings of 4th International workshop on OS support for Digital Audio and Video, November 1993. nossdav:berkom

- Martin Moser, Tohoku University, Declarative Scheduling for Optimally Graceful QoS Degradation, Proceedings of the IEEE Multimedia Systems '96, June 1996, Hiroshima, Japan

- http://amazon.postech.ac.kr/qos/index.html
  - end to end QOS Managment

- http://lrcwww.epfl.ch/WebOverATMDir/about_woa/olivier.html
  - Video Quality of Service (QoS) definition and control for MPEG-2 coded streams

- http://www.cs.columbia.edu/~jakka/pricing.html
  - Economic Paradigms for QoS in Information Networks. (Pricing and QoS Provisioning)

- http://www.arl.wustl.edu/arl/refpapers/gopal/os.html
  - Efficient Quality of Service Support in OS for Multimedia Applications

- http://www-mobile.ecs.soton.ac.uk/books/index.html
  - Published Books of the Communications Group, Department of Electronics and Computer Science, University of Southampton, U.K.: Modern Video Communications - Principles and Applications for Fixed and Wireless Channels.

- http://comet.ctr.columbia.edu/research/transport/qosmapping.html
  - QOS Mapping in Multimedia Networks

5.2.2 Papers/Reports

- Cranor, Chuck; Parulkar, Guru M.; "Universal Continuous Media I/O: Design and Implementation, Technical Report WUCS-94-34, Washington University, Department of Computer Science, August 1994.view paper

- Gopalakrishnan, R.; Parulkar, Guru M.; "Bringing Real-time Scheduling Theory and Practice Closer for Multimedia Computing," Proceedings of the ACM SIGMETRICS Conference on Measurement andModeling of Computer Systems, Philadelphia, May 1996. download postscript

- Gopalakrishnan, R.; Parulkar, Guru M.; "Efficient User space Protocol Implementations with QoS Guarantees using Real-time Upcalls," Technical Report WUCS-96-11, Washington University, Department of Computer Science, 1996. download postscript

- Buddhikot, Milind M.; Parulkar, Guru M.; Gopalakrishnan, R.; " Scalable Multimedia-On-Demand via World-Wide-Web (WWW) with QOS Guarantees," International Conference on Network and Operating Systems Support for Digital Audio and Video, NOSSDAV 96, Japan, 1996. download postscript

- Gopalakrishnan, R.; Parulkar, Guru M.; "Real-time Upcalls: A Mechanism to Provide Real-time Processing Guarantees," Technical Report WUCS-95-06, Washington University, Department of Computer Science, September 1995. download postscript

- Gopalakrishnan, R.; Parulkar, Guru M.; "A Real-time Upcall Facility for Protocol Processing with QoS Guarantees, ACM Symposium on Operating Systems Principles, (Poster Session), Copper Mountain CO, December 1995, pp. 231. download postscript

- Gopalakrishnan, R.; Parulkar, Guru M.; "RMDP-A Real-time CPU Scheduling Algorithm to Provide Guarantees for Protocol Processing," Proceedings of IEEE Real-time Technology and Applications Symposium,(Poster Session), Chicago, May 1995. download postscript

- Gopalakrishnan, R.; Parulkar, Guru M.; Quality of Service Support for Protocol Processing Within Endsystems," High Speed Networking for Multimedia Applications, Wolfgang Effelsberg et. al. (Editors), Kluwer Academic Publishers, 1995. download postscript

- Gopalakrishnan, R.; Parulkar, Guru M.; "A Framework for QoS Guarantees for Multimedia Applications within an Endsystem," Swiss German Computer Society Conference, September, 1995. download postscript

- Gopalakrishnan, R.; Parulkar, Guru M.; "Application Level Protocol Implementa tions to Provide QoS Guarantees at Endsystems," Proceedings of the Ninth IEEE Work shop on Computer Communications, Duck Key, Florida, October 1994. download postscript

- Gopalakrishnan, R.; Parulkar, Guru M.; "Efficient Quality of Service Support in Multimedia Computer Operating Systems," Technical Report WUCS-TM-94-04, Washington University, Department of Computer Science, August 1994. download postscript

- M. Alfano. Design and Implementation of a Cooperative Multimedia Environment with QoS Control. Computer Communications, Elsevier, Fall 1997.

- M. Alfano. User requirements and resource control for cooperative multimedia applications. Lecture Notes in Computer Science: Multimedia Applications, Services and Techniques - ECMAST '97, Fdida S. and Morganti M. (Eds.), Springer, vol. 1242, pp. 537-552, 1997.

- M. Alfano, N. Radouniklis. A Cooperative Multimedia Environment with QoS Control: Architectural and Implementation Issues. International Computer Science Institute Tecnical Report TR-96-040, September 1996.

- M. Alfano, R. Sigle. Controlling QoS in a collaborative multimedia environment. Proc. of the 5th IEEE International Symposium on High-Performance Distributed Computing (HPDC-5), Aug 7-9, 1996, Syracusae (NY), USA.

- M. Alfano, R. Sigle, R. Ulrich. A cooperative multimedia environment with user-driven QoS control. Proc. of the 8th IEEE Workshop on Local and Metropolitan Area Networks, Aug 25-28, 1996, Berlin/Potsdam, Germany. Abstract, Slides

- M. Alfano, R. Sigle, R. Ulrich. Management of cooperative multimedia sessions with QoS requirements. Proc. of IEEE Gigabit Networking Workshop GBN `96 in conjuction with IEEE INFOCOM'96, March 24, 1996, San Francisco (CA - USA).

- M. Alfano. A Quality of Service Management Architecture (QoSMA): A preliminary study. International Computer Science Institute Tecnical Report TR-95-070, December 1995.

- Harrick M. Vin, Pawan Goyal, Alok Goyal and Anshuman Goyal, A Statistical Admission Control Algorithm for Multimedia Servers, In Proceedings of the ACM Multimedia'94, San Francisco, Pages 33-40, October 1994. Source: DMCL multimedia paper

- Prashant J. Shenoy and Harrick M. Vin, Efficient Support for Scan Operations in Video Servers Technical Report TR-96-35, Department of Computer Sciences, University of Texas at Austin Source: Prashant Shenoy's Recent Publication

- Prashant J. Shenoy and Harrick M. Vin, Cello: A Disk Scheduling Framework for Next Generation Operating Systems Technical Report TR-97-27, Department of Computer Sciences, Univ. of Texas at Austin, October 1997 Source: Prashant Shenoy's Recent Publication

- Raj Rajkumar, Kanaka Juvva, Anastasio Molano and Shuichi Oikawa, Resource Kernels: A Resource-Centric Approach to Real-Time and Multimedia Systems , Multimedia Computing & Networking 1998, Source: CMU Research Project on Realtime Mach,

- Prashant J. Shenoy, Pawan Goyal, Sriram S. Rao, and Harrick M. Vin, Symphony: An Integrated Multimedia File System Multimedia Computing & Networking 1998

5.3 Quellcode - Hauptbestandteile

5.3.1 Headerdateien

```
// usermain.h ////////////////////////////////////////////////
//////////////////////////////////////////////////////////////
//Konstanten

#define MAX_ITEM_STR                    100
#define MAX_ITEM                          50
#define MAX_BUFFER                        255

//////////////////////////////////////////////////////////////
//Variablen
extern double DSFolge[MAX_ITEM];

extern double Dgesamt;
extern double anz_item;
extern double Tgesamt;
extern double mRate;

//Eingaben
extern double inputQ;
extern double inputTB;

extern char inputDatei[_MAX_PATH];
extern char inputDateiname[_MAX_PATH];
extern char outputDatei[_MAX_PATH];

//Ergebnisse
extern double resultm;
extern double resultTA;
extern double resultV;
extern double resultT0;
extern double resultP0;
extern double resultPmin;

extern double resultBplus;
extern double resultBminus;

// functions.h ///////////////////////////////////////
//////////////////////////////////////////////////////

//manuell modell 1
void CalculateTA_MM1(void);
void CalculateV_MM1(void);
void CalculatePmin_MM1(void);

// datei modell 1
void CalculateTA_DM1(void);
void CalculateV_DM1(void);
void CalculatePmin_DM1(void);

// extra
void CalculateT0(void);
void CalculateP0(void);

//manuell modell 2
void CalculateBplusBminusM(void);
void CalculateV_MM2(void);
```

```
void CalculatePmin_MM2(void);

//datei modell 2
void CalculateBplusBminusD(void);
void CalculateV_DM2(void);
void CalculatePmin_DM2(void);

// datei
bool DateikopfAuswerten(char datei[]);
void SFDateiSchreiben(char datei[]);

double GetDSF(long i);
```

5.3.2 Implementierungsdateien

```
// calcmm.cpp : Legt das Klassenverhalten für die Anwendung fest.
//

#include "stdafx.h"
#include "calcmm.h"
#include "calcmmDlg.h"
#include "usermain.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
//Variablen
double DSFolge[MAX_ITEM];

double Dgesamt = 0;
double anz_item = 0;
double Tgesamt = 0;
double mRate = 0;

//Eingaben
double inputQ = 0;
double inputTB = 0;

char inputDatei[_MAX_PATH] = "";
char inputDateiname[_MAX_PATH] = "";
char outputDatei[_MAX_PATH] = "";

//Ergebnisse
double resultm;
double resultTA = 0;
double resultV = 0;
double resultT0 = 0;
double resultP0 = 0;
double resultPmin = 0;

double resultBplus = 0;
double resultBminus = 0;

/////////////////////////////////////////////////////////////////////////////
//
// CCalcmmApp

BEGIN_MESSAGE_MAP(CCalcmmApp, CWinApp)
```

```
      //{{AFX_MSG_MAP(CCalcmmApp)
            // HINWEIS - Hier werden Mapping-Makros vom Klassen-Assistenten
eingefügt und entfernt.
            //    Innerhalb dieser generierten Quelltextabschnitte NICHTS
VERÄNDERN!
      //}}AFX_MSG
      ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
//
// CCalcmmApp Konstruktion

CCalcmmApp::CCalcmmApp()
{
      // ZU ERLEDIGEN: Hier Code zur Konstruktion einfügen
      // Alle wichtigen Initialisierungen in InitInstance platzieren
}

/////////////////////////////////////////////////////////////////////////////
//
// Das einzige CCalcmmApp-Objekt

CCalcmmApp theApp;

/////////////////////////////////////////////////////////////////////////////
//
// CCalcmmApp Initialisierung

BOOL CCalcmmApp::InitInstance()
{
      AfxEnableControlContainer();

      // Standardinitialisierung
      // Wenn Sie diese Funktionen nicht nutzen und die Größe Ihrer ferti-
gen
      //  ausführbaren Datei reduzieren wollen, sollten Sie die nachfolgen-
den
      //  spezifischen Initialisierungsroutinen, die Sie nicht benötigen,
entfernen.

#ifdef _AFXDLL
      Enable3dControls();                 // Diese Funktion bei Verwendung
von MFC in gemeinsam genutzten DLLs aufrufen
#else
      Enable3dControlsStatic();     // Diese Funktion bei statischen MFC-
Anbindungen aufrufen
#endif
  CCalcmmDlg dlg;
      m_pMainWnd = &dlg;
      int nResponse = dlg.DoModal();
      if (nResponse == IDOK)
      {
            // ZU ERLEDIGEN: Fügen Sie hier Code ein, um ein Schließen des
            //  Dialogfelds über OK zu steuern
      }
      else if (nResponse == IDCANCEL)
      {
            // ZU ERLEDIGEN: Fügen Sie hier Code ein, um ein Schließen des
            //  Dialogfelds über "Abbrechen" zu steuern
      }

      // Da das Dialogfeld geschlossen wurde, FALSE zurückliefern, so dass
wir die
```

```
      //  Anwendung verlassen, anstatt das Nachrichtensystem der Anwendung
zu starten.
      return FALSE;
}


// calcmmDlg.cpp : Implementierungsdatei
//

#include "stdafx.h"
#include "calcmm.h"
#include "calcmmDlg.h"
#include "cderr.h"
#include "usermain.h"
#include "functions.h"


#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif



/////////////////////////////////////////////////////////////////////////////
//
// CAboutDlg-Dialogfeld für Anwendungsbefehl "Info"

class CAboutDlg : public CDialog
{
public:
      CAboutDlg();

// Dialogfelddaten
      //{{AFX_DATA(CAboutDlg)
      enum { IDD = IDD_ABOUTBOX };
      //}}AFX_DATA

      // Vom Klassenassistenten generierte Überladungen virtueller Funktio-
nen
      //{{AFX_VIRTUAL(CAboutDlg)
      protected:
      virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV-
Unterstützung
      //}}AFX_VIRTUAL

// Implementierung
protected:
      //{{AFX_MSG(CAboutDlg)
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
      //{{AFX_DATA_INIT(CAboutDlg)
      //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
      CDialog::DoDataExchange(pDX);
      //{{AFX_DATA_MAP(CAboutDlg)
```

```
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// Keine Nachrichten-Handler
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
//
// CCalcmmDlg Dialogfeld

CCalcmmDlg::CCalcmmDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CCalcmmDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CCalcmmDlg)
	//}}AFX_DATA_INIT
	// Beachten Sie, dass LoadIcon unter Win32 keinen nachfolgenden De-
stroyIcon-Aufruf benötigt
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CCalcmmDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CCalcmmDlg)
	DDX_Control(pDX, IDC_Quantum2, m_Quantum2);
	DDX_Control(pDX, IDC_OkDatei, m_OkDatei);
	DDX_Control(pDX, IDC_TB1, m_TB1);
	DDX_Control(pDX, IDC_Datei, m_Datei);
	DDX_Control(pDX, IDC_Anzahl2, m_Anzahl2);
	DDX_Control(pDX, IDC_Anzahl1, m_Anzahl1);
	DDX_Control(pDX, IDC_TB2, m_TB2);
	DDX_Control(pDX, IDC_Quantum1, m_Quantum1);
	DDX_Control(pDX, IDC_Datenmenge2, m_Datenmenge2);
	DDX_Control(pDX, IDC_Datenmenge1, m_Datenmenge1);
	DDX_Control(pDX, IDC_Pmin, m_Pmin);
	DDX_Control(pDX, IDC_P0, m_P0);
	DDX_Control(pDX, IDC_T0, m_Vorperiode);
	DDX_Control(pDX, IDC_V, m_Vorlauf);
	DDX_Control(pDX, IDC_TA, m_TA);
	DDX_Control(pDX, IDC_Datenpackete, m_Datenpackete);
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CCalcmmDlg, CDialog)
	//{{AFX_MSG_MAP(CCalcmmDlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_EN_KILLFOCUS(IDC_Datenpackete, OnKillfocusDatenpackete)
	ON_BN_CLICKED(IDC_ManuM1, OnManuM1)
	ON_BN_CLICKED(IDC_ManuM2, OnManuM2)
	ON_BN_CLICKED(IDC_DateiLaden, OnDateiLaden)
	ON_EN_KILLFOCUS(IDC_Quantum1, OnKillfocusQuantum1)
	ON_EN_KILLFOCUS(IDC_TB1, OnKillfocusTB1)
	ON_BN_CLICKED(IDC_DateiM1, OnDateiM1)
	ON_BN_CLICKED(IDC_DateiM2, OnDateiM2)
	ON_BN_CLICKED(IDC_ClearAll, OnClearAll)
	ON_BN_CLICKED(IDC_DateiOut1, OnDateiOut1)
	ON_BN_CLICKED(IDC_DateiOut2, OnDateiOut2)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
/////////////////////////////////////////////////////////////////////////////
//
// CCalcmmDlg Nachrichten-Handler

BOOL CCalcmmDlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Hinzufügen des Menübefehls "Info..." zum Systemmenü.

	// IDM_ABOUTBOX muss sich im Bereich der Systembefehle befinden.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMe-
nu);
		}
	}

	// Symbol für dieses Dialogfeld festlegen. Wird automatisch erledigt
	//  wenn das Hauptfenster der Anwendung kein Dialogfeld ist
	SetIcon(m_hIcon, TRUE);			// Großes Symbol verwenden
	SetIcon(m_hIcon, FALSE);		// Kleines Symbol verwenden

	// ZU ERLEDIGEN: Hier zusätzliche Initialisierung einfügen

	return TRUE;  // Geben Sie TRUE zurück, außer ein Steuerelement soll
den Fokus erhalten
}

void CCalcmmDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}

// Wollen Sie Ihrem Dialogfeld eine Schaltfläche "Minimieren" hinzufügen,
benötigen Sie
//  den nachstehenden Code, um das Symbol zu zeichnen. Für MFC-Anwendungen,
die das
//  Dokument/Ansicht-Modell verwenden, wird dies automatisch für Sie erle-
digt.

void CCalcmmDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // Gerätekontext für Zeichnen
```

```
            SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

            // Symbol in Client-Rechteck zentrieren
            int cxIcon = GetSystemMetrics(SM_CXICON);
            int cyIcon = GetSystemMetrics(SM_CYICON);
            CRect rect;
            GetClientRect(&rect);
            int x = (rect.Width() - cxIcon + 1) / 2;
            int y = (rect.Height() - cyIcon + 1) / 2;

            // Symbol zeichnen
            dc.DrawIcon(x, y, m_hIcon);
      }
      else
      {
            CDialog::OnPaint();
      }
}

// Die Systemaufrufe fragen den Cursorform ab, die angezeigt werden soll,
während der Benutzer
//  das zum Symbol verkleinerte Fenster mit der Maus zieht.
HCURSOR CCalcmmDlg::OnQueryDragIcon()
{
      return (HCURSOR) m_hIcon;
}


void CCalcmmDlg::OnKillfocusDatenpackete()
{
      // TODO: Code für die Behandlungsroutine der Steuerelement-
Benachrichtigung hier einfügen
      long i = 0;
      long len = 0;
      long item;

      char itemstr[MAX_ITEM_STR];
      char *stopstr;

      char buffer[MAX_BUFFER];

      anz_item = m_Datenpackete.GetLineCount();
      Dgesamt = 0;
      DSFolge[0] = 0;

      for(i=0;i<anz_item;i++)
      {
            len = m_Datenpackete.GetLine(i,itemstr,MAX_ITEM_STR);
            itemstr[len]=0;
            item = strtol(itemstr,&stopstr,10);
            Dgesamt += item;
            DSFolge[i+1] = Dgesamt;
      }

      m_Anzahl1.SetWindowText(gcvt(anz_item,10, buffer));
      m_Datenmenge1.SetWindowText(gcvt(Dgesamt,10, buffer));

      //ergebnisse löschen
      m_TA.SetWindowText("");
      m_Vorlauf.SetWindowText("");
      m_Vorperiode.SetWindowText("");
      m_P0.SetWindowText("");
      m_Pmin.SetWindowText("");
```

```
}

void CCalcmmDlg::OnManuM1()
{
	// TODO: Code für die Behandlungsroutine der Steuerelement-
Benachrichtigung hier einfügen

	char buffer[MAX_BUFFER];

	AfxGetApp()->DoWaitCursor(1);

	if((Dgesamt != 0)&&(anz_item != 0)&&(inputQ != 0)&&(inputTB != 0))
	{
		CalculateTA_MM1();
		CalculateV_MM1();
		CalculateT0();
		CalculateP0();
		CalculatePmin_MM1();

		m_TA.SetWindowText(gcvt(resultTA,10, buffer));
		m_Vorlauf.SetWindowText(gcvt(resultV,10, buffer));
		m_Vorperiode.SetWindowText(gcvt(resultT0,10, buffer));
		m_P0.SetWindowText(gcvt(resultP0,10, buffer));
		m_Pmin.SetWindowText(gcvt(resultPmin,10, buffer));
	}
	AfxGetApp()->DoWaitCursor(-1);
}


void CCalcmmDlg::OnManuM2()
{
	// TODO: Code für die Behandlungsroutine der Steuerelement-
Benachrichtigung hier einfügen
	char buffer[MAX_BUFFER];

	AfxGetApp()->DoWaitCursor(1);

	if((Dgesamt != 0)&&(anz_item != 0)&&(inputQ != 0)&&(inputTB != 0))
	{
		//Tgesamt berechnen
		Tgesamt = anz_item * inputTB;

		//mRate berechnen
		mRate = Dgesamt / Tgesamt;

		CalculateBplusBminusM();
		CalculateV_MM2();
		CalculatePmin_MM2();

		m_TA.SetWindowText("");
		m_Vorlauf.SetWindowText(gcvt(resultV,10, buffer));
		m_Vorperiode.SetWindowText("");
		m_P0.SetWindowText("");
		m_Pmin.SetWindowText(gcvt(resultPmin,10, buffer));
	}
  AfxGetApp()->DoWaitCursor(-1);

}

void CCalcmmDlg::OnDateiLaden()
{
	// TODO: Code für die Behandlungsroutine der Steuerelement-
Benachrichtigung hier einfügen
```

```
      OPENFILENAME ofn;
      DWORD ret;
      CDC *dc = GetDC();
      char buffer[MAX_BUFFER];

      // Initialize OPENFILENAME
      ZeroMemory(&ofn, sizeof(OPENFILENAME));
      ofn.lStructSize = sizeof(OPENFILENAME);
      ofn.hwndOwner = NULL;
      ofn.lpstrFile = inputDatei;
      ofn.nMaxFile = sizeof(inputDatei);
      ofn.lpstrFilter = "All (*.*)\0*.*\0Text (*.txt)\0*.TXT\0";
      ofn.nFilterIndex = 1;
      ofn.lpstrFileTitle = inputDateiname;
      ofn.nMaxFileTitle = sizeof(inputDateiname);
      ofn.lpstrInitialDir = NULL;
      ofn.Flags = OFN_PATHMUSTEXIST |OFN_EXPLORER |
                        OFN_FILEMUSTEXIST | OFN_ALLOWMULTISELECT;

      GetOpenFileName(&ofn);
      ret = CommDlgExtendedError() ;

      m_Datei.SetWindowText(inputDatei);



  // file checken, summenfolgen datei schreiben
      m_OkDatei.SetWindowText("?");
      if(DateikopfAuswerten(inputDatei))
      {
            m_OkDatei.SetWindowText("Ok");
            m_Datenmenge2.SetWindowText(gcvt(Dgesamt,10, buffer));
    m_Anzahl2.SetWindowText(gcvt(anz_item,10, buffer));
        m_Quantum2.SetWindowText(gcvt(inputQ,10, buffer));
        m_TB2.SetWindowText(gcvt(inputTB,10, buffer));
            SFDateiSchreiben(inputDatei);

            if((Dgesamt != 0)&&(anz_item != 0)&&(inputQ != 0)&&(inputTB !=
0))
                  m_OkDatei.SetWindowText("Ok");
      }
      else
      {
            m_Datenmenge2.SetWindowText("");
    m_Anzahl2.SetWindowText("");
        m_Quantum2.SetWindowText("");
        m_TB2.SetWindowText("");
      }

}

void CCalcmmDlg::OnKillfocusQuantum1()
{
      // TODO: Code für die Behandlungsroutine der Steuerelement-
Benachrichtigung hier einfügen
      char *stopstr;
      CString line;

      m_Quantum1.GetWindowText(line);
      inputQ = strtod(line,&stopstr);

}

void CCalcmmDlg::OnKillfocusTB1()
```

```
{
	// TODO: Code für die Behandlungsroutine der Steuerelement-
Benachrichtigung hier einfügen
	char *stopstr;
	CString line;

	m_TB1.GetWindowText(line);
	inputTB = strtod(line,&stopstr);

}

void CCalcmmDlg::OnDateiM1()
{
	CString ok_str;
	char buffer[MAX_BUFFER];

	AfxGetApp()->DoWaitCursor(1);

	m_OkDatei.GetWindowText(ok_str);
	if(ok_str==CString("Ok"))
	{
		CalculateTA_DM1();
		CalculateV_DM1();
		CalculateT0();
		CalculateP0();
		CalculatePmin_DM1();

		m_TA.SetWindowText(gcvt(resultTA,10, buffer));
		m_Vorlauf.SetWindowText(gcvt(resultV,10, buffer));
		m_Vorperiode.SetWindowText(gcvt(resultT0,10, buffer));
		m_P0.SetWindowText(gcvt(resultP0,10, buffer));
		m_Pmin.SetWindowText(gcvt(resultPmin,10, buffer));
	}



	AfxGetApp()->DoWaitCursor(-1);

}

void CCalcmmDlg::OnDateiM2()
{
	CString ok_str;
	char buffer[MAX_BUFFER];

	AfxGetApp()->DoWaitCursor(1);

	m_OkDatei.GetWindowText(ok_str);
	if(ok_str==CString("Ok"))
	{
		//Tgesamt berechnen
		Tgesamt = anz_item * inputTB;

		//mRate berechnen
		mRate = Dgesamt / Tgesamt;

		CalculateBplusBminusD();
		CalculateV_DM2();
		CalculatePmin_DM2();

		m_TA.SetWindowText("");
		m_Vorlauf.SetWindowText(gcvt(resultV,10, buffer));
		m_Vorperiode.SetWindowText("");
		m_P0.SetWindowText("");
```

```
            m_Pmin.SetWindowText(gcvt(resultPmin,10, buffer));
      }

      AfxGetApp()->DoWaitCursor(-1);

}

void CCalcmmDlg::OnClearAll()
{
  m_Datenpackete.SetWindowText("");
      m_Datenmenge1.SetWindowText("");
  m_Anzahl1.SetWindowText("");
      m_Quantum1.SetWindowText("");
      m_TB1.SetWindowText("");

      m_TA.SetWindowText("");
      m_Vorlauf.SetWindowText("");
      m_Vorperiode.SetWindowText("");
      m_P0.SetWindowText("");
      m_Pmin.SetWindowText("");

  m_Datei.SetWindowText("");
  m_OkDatei.SetWindowText("");
      m_Datenmenge2.SetWindowText("");
  m_Anzahl2.SetWindowText("");
  m_Quantum2.SetWindowText("");
  m_TB2.SetWindowText("");

  Dgesamt   =0;
      anz_item=0;
      Tgesamt     =0;
      mRate       =0;

      inputQ      =0;
      inputTB     =0;

      strcpy(inputDatei,"");
      strcpy(inputDateiname,"");
      strcpy(outputDatei,"");

  resultm               =0;
  resultTA        =0;
  resultV               =0;
  resultT0        =0;
  resultP0        =0;
  resultPmin      =0;

  resultBplus     =0;
  resultBminus=0;

}

void CCalcmmDlg::OnDateiOut1()
{
  // ergebnisse in manuell.out
      CFileException ferror;
      CStdioFile datei;
      CString hstr(inputDatei);
      char itemstr[MAX_ITEM_STR];
      char buffer[MAX_BUFFER];

      long len = 0;
      long i = 0;
```

```
      if(datei.Open(".\\manuell.out",CFile::modeWrite | CFile::modeCreate
,&ferror))
      {
            datei.WriteString("####Eingabe########## \n");

            hstr = "D =" + CString(gcvt(Dgesamt,10,buffer)) + "\n";
            datei.WriteString(LPCTSTR(hstr));

            hstr = "n =" + CString(gcvt(anz_item,10,buffer)) + "\n";
            datei.WriteString(LPCTSTR(hstr));

            hstr = "Q =" + CString(gcvt(inputQ,10,buffer)) + "\n";
            datei.WriteString(LPCTSTR(hstr));

            hstr = "TB=" + CString(gcvt(inputTB,10,buffer)) + "\n";
            datei.WriteString(LPCTSTR(hstr));

            datei.WriteString("#####Packetfolge##### \n");

            for(i=0;i<anz_item;i++)
            {
                  len = m_Datenpackete.GetLine(i,itemstr,MAX_ITEM_STR);
                  datei.WriteString(strcat(itemstr,"\n"));
            }

            datei.WriteString("###################### \n");
            datei.WriteString("#####Ergebnisse###### \n");

        m_TA.GetWindowText(hstr);
            hstr = "TA=" + hstr + "\n";
            datei.WriteString(LPCTSTR(hstr));

        m_Pmin.GetWindowText(hstr);
            hstr = "Pmin=" + hstr + "\n";
            datei.WriteString(LPCTSTR(hstr));

        m_Vorlauf.GetWindowText(hstr);
            hstr = "V=" + hstr + "\n";
            datei.WriteString(LPCTSTR(hstr));

        m_P0.GetWindowText(hstr);
            hstr = "P0=" + hstr + "\n";
            datei.WriteString(LPCTSTR(hstr));

        m_Vorperiode.GetWindowText(hstr);
            hstr = "T0=" + hstr + "\n";
            datei.WriteString(LPCTSTR(hstr));
      }
}

void CCalcmmDlg::OnDateiOut2()
{
  // ergebnisse in *.out
      CFileException ferror;
      CStdioFile datei;
      CString outdatei;
      CString hstr(inputDatei);

      outdatei = hstr.Left(hstr.Find(".")+1) + "out";

      if(datei.Open(LPCTSTR(outdatei),CFile::modeWrite | CFile::modeCreate
,&ferror))
      {
        m_TA.GetWindowText(hstr);
```

```
            hstr = "TA=" + hstr + "\n";
            datei.WriteString(LPCTSTR(hstr));

        m_Pmin.GetWindowText(hstr);
            hstr = "Pmin=" + hstr + "\n";
            datei.WriteString(LPCTSTR(hstr));

        m_Vorlauf.GetWindowText(hstr);
            hstr = "V=" + hstr + "\n";
            datei.WriteString(LPCTSTR(hstr));

        m_P0.GetWindowText(hstr);
            hstr = "P0=" + hstr + "\n";
            datei.WriteString(LPCTSTR(hstr));

        m_Vorperiode.GetWindowText(hstr);
            hstr = "T0=" + hstr + "\n";
            datei.WriteString(LPCTSTR(hstr));

      }

}



// Implementierungsdatei functions.cpp
/////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "calcmm.h"
#include "calcmmDlg.h"

#include "math.h."
#include "usermain.h"

/////////////////////////////////////////////////////////////////////
//Funktionen

void CalculateTA_MM1(void);
void CalculateV_MM1(void);
void CalculatePmin_MM1(void);

void CalculateTA_DM1(void);
void CalculateV_DM1(void);
void CalculatePmin_DM1(void);

void CalculateT0(void);
void CalculateP0(void);

double AngebotM(double t);
double BedarfM(double t);
double AngebotD(double t);
double BedarfD(double t);


double AObenMinusEinsM(double x);
double BObenMinusEinsM(double x);
double AObenMinusEinsD(double x);
double BObenMinusEinsD(double x);


double DeltaM(double);
double DeltaD(double);
```

```
double GetDSF(long i);

////////////////////////////////////////////////////////////////////
//#################################################################
void CalculateTA_MM1(void)
{
      resultm = Dgesamt / inputQ;
      if (Dgesamt != 0) resultTA = anz_item * inputQ/ Dgesamt * inputTB;
}

//#################################################################
void CalculateTA_DM1(void)
{
      resultm = Dgesamt / inputQ;
      if (Dgesamt != 0) resultTA = anz_item * inputQ/ Dgesamt * inputTB;
}
//#################################################################
void CalculateV_MM1(void)
{
  double x;
      double delta;

      resultV = DeltaM(0);
      for(x=1;x<=Dgesamt;x++)
      {
        delta = DeltaM(x);
        if(delta>resultV) resultV = delta;;
      }
}

//#################################################################
void CalculateV_DM1(void)
{
  double x;
      double delta;

      resultV = DeltaD(0);
      for(x=1;x<=Dgesamt;x++)
      {
        delta = DeltaD(x);
        if(delta>resultV) resultV = delta;;
      }
}

//#################################################################
void CalculatePmin_MM1(void)
{
  double te;
      double t;
      double value;

      // te berechnen
      te = ceil(resultm-1)*resultTA-resultV;

      // Pmin berechnen
      resultPmin = resultP0;
      for(t=1;t<=te;t++)
      {
        value = AngebotM(t+resultV)-BedarfM(t-inputTB);
        if(value>resultPmin) resultPmin = value;
      }
}

//#################################################################
```

```
void CalculatePmin_DM1(void)
{
  double te;
      double t;
      double value;

      // te berechnen
      te = ceil(resultm-1)*resultTA-resultV;

      // Pmin berechnen
      resultPmin = resultP0;
      for(t=1;t<=te;t++)
      {
        value = AngebotD(t+resultV)-BedarfD(t-inputTB);
        if(value>resultPmin) resultPmin = value;
      }
}

//#####################################################################
void CalculateT0(void)
{
      if (resultTA != 0) resultT0 = resultTA - (long)resultV %
(long)resultTA;
}

//#####################################################################
void CalculateP0(void)
{
      resultP0 = (floor(resultV/resultTA)+1)*inputQ;
}

//#####################################################################
double AngebotM(double t)
{
      double i;

      if(t<0) return(0);
      if(t>=ceil(resultm-1)*resultTA) return(Dgesamt);

      for(i=1;i<=ceil(resultm-1);i++)
      {
        if(((i-1)*resultTA<=t) && (t<(i*resultTA))) return (i*inputQ);
      }

      //andernfalls
      return(-1);
}
//#####################################################################
double AngebotD(double t)
{
      double i;

      if(t<0) return(0);
      if(t>=ceil(resultm-1)*resultTA) return(Dgesamt);

      for(i=1;i<=ceil(resultm-1);i++)
      {
        if(((i-1)*resultTA<=t) && (t<(i*resultTA))) return (i*inputQ);
      }

      //andernfalls
      return(-1);
}
//#####################################################################
```

```
double BedarfM(double t)
{
  long j;

      if(t<0) return(0);
      if(t>=(double)anz_item*inputTB) return(Dgesamt);

      for(j=1;j<=anz_item;j++)
      {
        if(((j-1)*inputTB<=t) && (t<(j*inputTB))) return (DSFolge[j]);
      }
  return(-1);
}
//#####################################################################
double BedarfD(double t)
{
  long j;

      if(t<0) return(0);
      if(t>=(double)anz_item*inputTB) return(Dgesamt);

      for(j=1;j<=anz_item;j++)
      {
        if(((j-1)*inputTB<=t) && (t<(j*inputTB))) return (GetDSF(j));
      }
  return(-1);
}
//#####################################################################
double AObenMinusEinsM(double x)
{
      double i;
      double h;
      double result = 0;

      h = ceil(resultm-1);

      for(i=0;i<h;i++)
      {
            if(((i*inputQ)<x) && (x<=(i+1)*inputQ)) result = i*resultTA;
      }

      if (((h*inputQ)<x) && (x<=Dgesamt)) result = h*resultTA;

      return(result);
}
//#####################################################################
double AObenMinusEinsD(double x)
{
      double i;
      double h;
      double result = 0;

      h = ceil(resultm-1);

      for(i=0;i<h;i++)
      {
            if(((i*inputQ)<x) && (x<=(i+1)*inputQ)) result = i*resultTA;
      }

      if (((h*inputQ)<x) && (x<=Dgesamt)) result = h*resultTA;

      return(result);
}
//#####################################################################
```

```
double BObenMinusEinsM(double x)
{
      long j = 0;

      for(j=0;j<anz_item;j++)
      {
            if (DSFolge[j] != DSFolge[j+1])
            {
                  if((DSFolge[j]<x)&&(x<=DSFolge[j+1])) return(j * in-
putTB);
            }
      }
  return(0);
}

//#####################################################################
double BObenMinusEinsD(double x)
{
      long j = 0;
  double dj;
      double dj1;

      for(j=0;j<anz_item;j++)
      {
            dj = GetDSF(j);
            dj1 = GetDSF(j+1);
            if (dj != dj1)
            {
                  if((dj<x)&&(x<=dj1)) return(j * inputTB);
            }
      }
  return(0);
}
//#####################################################################
double DeltaM(double x)
{
            return(AObenMinusEinsM(x)-BObenMinusEinsM(x));
}

//#####################################################################
double DeltaD(double x)
{
            return(AObenMinusEinsD(x)-BObenMinusEinsD(x));
}

//#####################################################################
//#####################################################################
//#####################################################################
//2. modell ##########################################################
double BquerM(double t)
{
  return(mRate*t);
}

//#####################################################################
double BquerD(double t)
{
  return(mRate*t);
}

//#####################################################################
void CalculateBplusBminusM(void)
{
      double value;
```

```
      double t;

      resultBplus = resultBminus = 0;

      for(t=0; t<=Tgesamt; t++)
      {
            value = BedarfM(t)-BquerM(t);
            if(value > resultBplus) resultBplus = value;
            else
            {
              if(value < resultBminus) resultBminus = value;
            }

      }
}

//#####################################################################
void CalculateBplusBminusD(void)
{
      double value;
      double t;

      resultBplus = resultBminus = 0;

      for(t=0; t<=Tgesamt; t++)
      {
            value = BedarfD(t)-BquerD(t);
            if(value > resultBplus) resultBplus = value;
            else
            {
              if(value < resultBminus) resultBminus = value;
            }

      }
}

//#####################################################################
void CalculateV_MM2(void)
{
      //möglichst nur ganze millisekunden
      if(mRate != 0) resultV = floor(resultBplus / mRate) ;
}

//#####################################################################
void CalculatePmin_MM2(void)
{
  double P;
      P = resultBplus-resultBminus+inputQ+2*mRate*inputTB-1;
  if(P<0) resultPmin = floor(P)*(-1);
      else resultPmin = ceil(P);
}

//#####################################################################
void CalculateV_DM2(void)
{
      //möglichst nur ganze millisekunden
      if(mRate != 0) resultV = floor(resultBplus / mRate) ;
}

//#####################################################################
void CalculatePmin_DM2(void)
{
  double P;
      P = resultBplus-resultBminus+inputQ+2*mRate*inputTB-1;
```

```
  if(P<0) resultPmin = floor(P)*(-1);
      else resultPmin = ceil(P);
}
//#####################################################################
//#####################################################################
//#####################################################################
//Dateiarbeit

bool DateikopfAuswerten(char datei[])
{
      CString line;
      CStdioFile objdatei( datei,CFile::modeRead);
      char *stopstr;

      // # testen
      if(objdatei.ReadString(line))
      {
            if((line.Left(1)!=CString("#"))) return(false); // falscer Da-
teianfang
      }
      else  return(false); // datei zu kurz

      // Dgesamt zuerst
      if(objdatei.ReadString(line)) Dgesamt = strtod(line,&stopstr);
      else  return(false); // datei zu kurz

      // anz_item
      if(objdatei.ReadString(line)) anz_item = strtod(line,&stopstr);
      else  return(false); // datei zu kurz

      // inputQ
      if(objdatei.ReadString(line)) inputQ = strtod(line,&stopstr);
      else  return(false); // datei zu kurz

      // inputTB
      if(objdatei.ReadString(line)) inputTB = strtod(line,&stopstr);
      else  return(false); // datei zu kurz

  return (true);
}

//#####################################################################
void SFDateiSchreiben(char datei[_MAX_PATH])
{
      CStdioFile objdatei( datei,CFile::modeRead);
      CString line;
      CString helpstr(datei);
      CFile DSFolgeDatei;
      CString sfdatei;
      CFileException ferror;
      char *stopstr;

      double value = 0;
      double summe = 0;

      objdatei.ReadString(line);  //erste "#..." linie

  do objdatei.ReadString(line);
      while((line.Left(1)!=CString("#"))); //zweite "#..." linie

// dateiname der ausgabe
      sfdatei = helpstr.Left(helpstr.ReverseFind(92)+1) +
CString("daten.sum");
      strcpy(outputDatei,LPCTSTR(sfdatei));
```

```
      if(DSFolgeDatei.Open(outputDatei,CFile.modeWrite | CFi-
le::modeCreate,&ferror))

      {
            DSFolgeDatei.Write(&summe,sizeof(value));
            for(double i=1;i<=anz_item;i++)
            {
                  // alle Datenpackete
                  objdatei.ReadString(line);
                  value = strtod(line,&stopstr);
                  summe+=value;
                  DSFolgeDatei.Write(&summe,sizeof(value));
            }
            DSFolgeDatei.Close();
      }
}

//#####################################################################
double GetDSF(long i)
{
      double ret = -1;
      CFileException ferror;
      CFile DSFolgeDatei;

      if(DSFolgeDatei.Open(outputDatei,CFile::modeRead,&ferror))
      {
                  DSFolgeDatei.Seek(i*sizeof(double),CFile::begin);
                  DSFolgeDatei.Read(&ret,sizeof(ret));
                  return (ret) ;
      }
      return(-1);
}

//#####################################################################
```

5.4 Datenträger