Simulation von Warteschlangennetzen mit BNETD

Martin Borriss*
TU Dresden
Institut für Betriebssysteme, Datenbanken und Rechnernetze

1. Dezember 1995

Zusammenfassung

Mit BNETD (Bedienungsnetze der TU Dresden) wurde ein System zur analytischen und simulativen Lösung von Warteschlangennetzen entwickelt. BNETD folgt dem Trend der Integration von exakter und approximativer Analyse, Modellierung, Verifizierung und Simulation in ein Programmsystem[1][2].

Wir beschreiben im ersten Teil das Design und die Implementation des Paketes. Insbesondere wird die Simulationskomponente[1] sowie Aspekte der Entwicklung der graphischer Modellierungsunterstützung näher erläutert.

Im zweiten Teil demonstrieren wir die Leistungsfähigkeit der Simulationskomponente anhand von zwei Problemen zu ATM-Netzen. Hierbei werden die Pufferanordnung innerhalb eines ATM-Switches untersucht und die Übertragung von Multimediadaten über ein ATM-Netz modelliert.

1 Entwurf und Implementation

1.1 Einleitung

Seit Beginn der achtziger Jahre wurde an der TU Dresden an einem Werkzeug zur Analyse und Simulation von Warteschlangennetzen gearbeitet [3]. Die hier vorgestellte Version beruht auf einem vollständigen Neuentwurf in den Jahren 1993/94. BNETD stellt ein integriertes Werkzeug dar, welches Modellierung, Simulation und Analyse innerhalb eines einzigen Programmsystems ermöglicht. Folgende Kriterien wurden beim Entwurf betont:

- Intuitive und einfache Bedienbarkeit,
- Kompaktheit und Integrierung der Komponenten.

Der vorgesehene Verwendungszweck von BNETD ist vorrangig der Einsatz als Praktikumssystem in der Hochschulausbildung[2], insbesondere an der TU Dresden. Demzufolge wurden - wegen der leichten Zugänglichkeit - PCs als Hardwareplattform gewählt mit MS-DOS als Betriebssystem. Die Entwicklung von BNETD wurde über einen Zeitraum von etwa einem Jahr vollzogen. Das hier vorgestellte BNETD ist vollständig in C++ geschrieben.

1.2 Entwurf

BNETD ist modular strukturiert, die Entwicklung der einzelnen Module erfolgte im wesentlichen zeitlich parallel. Als Schnittstelle zwischen den einzelnen Modulen wurden zwei grundlegende Datenstrukturen definiert:

- **Modellbeschreibung**

  Alle von BNETD zu behandelnden Modelle können durch diese Datenstruktur beschrieben werden. Sowohl Analyse- als auch Simulationskomponente verwenden diese Struktur als Eingabeparameter.

- **Ergebnisschnittstelle**

  Leistungsgrößen, ermittelt durch analytische, numerische oder simulative Lösung, werden mittels dieser Struktur zurückpropagiert. Die Uniformität der Ergebnisschnittstelle ermöglicht die Weiterverarbeitung und Aufbereitung der Ergebnisse.

Zusätzlich wurde eine Funktionsschnittstelle definiert, die für die Analysekomponente notwendig ist

*Email: Martin.Borriss@inf.tu-dresden.de

[1] entwickelt im Rahmen einer von Prof.K.Irmscher (TU Bergakademie Freiberg) betreuten Diplomarbeit

[2] Im Wintersemester 95/96 ist der Einsatz an der TU Dresden und an der TU Bergakademie Freiberg geplant.

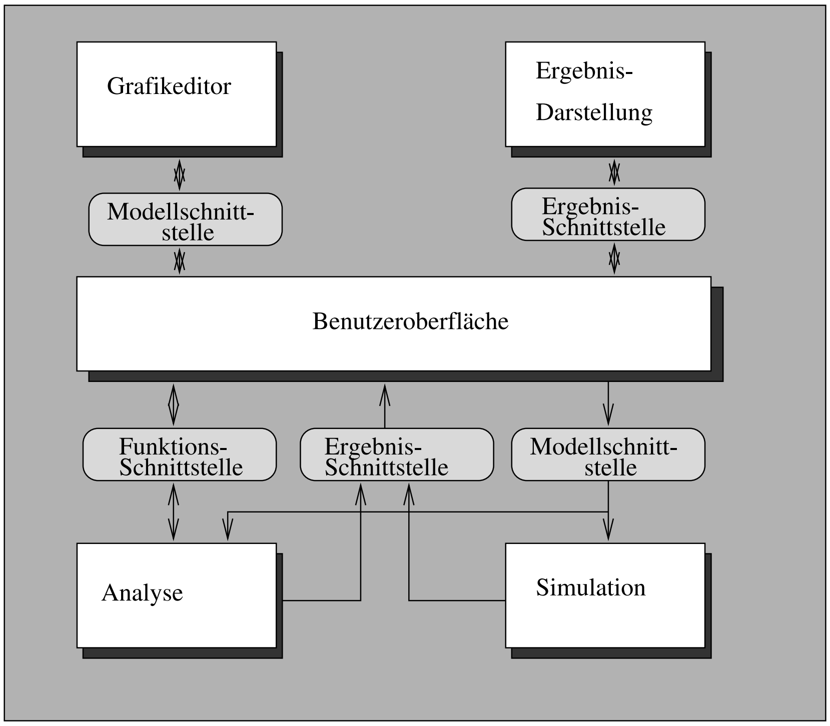

Abbildung 1: Struktur von BNETD

(Ermittlung von Zustands- und Randwahrscheinlichkeiten). Der Sachverhalt ist in Abb. 1 dargestellt.

Im folgenden wird das Design der einzelnen Module beschrieben.

1.3 Nutzerschnittstelle

Die Akzeptanz von Werkzeugen hängt sehr wesentlich von der Gestaltung der Nutzerschnittstelle ab. Benutzer legen Wert auf „einfache" Werkzeuge, die leicht zu nutzen sind und mit einem Minimum an gedruckter Dokumentation und Einarbeitung auskommen. Plattformen sollten „klein" sein, und das Werkzeug muß die Flexibilität aufweisen, innerhalb einer einheitlichen Umgebung verschiedene Lösungen (analytisch, numerisch und simulativ) bereitzustellen. Weiterhin ist eine Unterstützung bei der Präsentation und Aufbereitung der teilweise großen Menge von Ergebnisgrößen unabdingbar [2].

Beim Neuentwurf von BNETD wurde diesen Kriterien entsprochen, Ideen für zusätzliche Eigenschaften wurden zeitgleich mit der Implementation umgesetzt. Zur Zeit der Realisierung der Oberfläche [4] entsprach die pseudographische Darstellung [5] (siehe Abbildung 3) den Möglichkeiten der Hardware - aus heutiger Sicht wäre eine rein graphische Oberfläche eher angemessen.

Die wichtigsten Charakteristika seien kurz zusammengefaßt:

- **Pseudographische windoworientierte Oberfläche (SAA/CUA konform).**
- **Integrierter graphischer Editor**

  Der graphische Editor von BNETD (Abbildung 2) ermöglicht die schnelle und intuitive Modellierung der Modellstruktur mit ihren wichtigsten Merkmalen (Plazierung von Modellelementen, Übergängen und Übergangswahrscheinlichkeiten jeweils für mehrere Auftragsklassen). Die Parametrisierung (Warteschlangenstrategien, Verteilungen) erfolgt anschließend dialogorientiert.
- **Graphische Modelldarstellung**

  Anwendern steht jederzeit ein detaillierter Überblick über das aktuelle Modell zur Verfügung. Dieser wird wie folgt realisiert:

  - *Statusfenster*

    Das Statusfenster stellt alle Knoten eines Modelles und ihre aktuellen Eigenschaften im Überblick dar, innerhalb die-

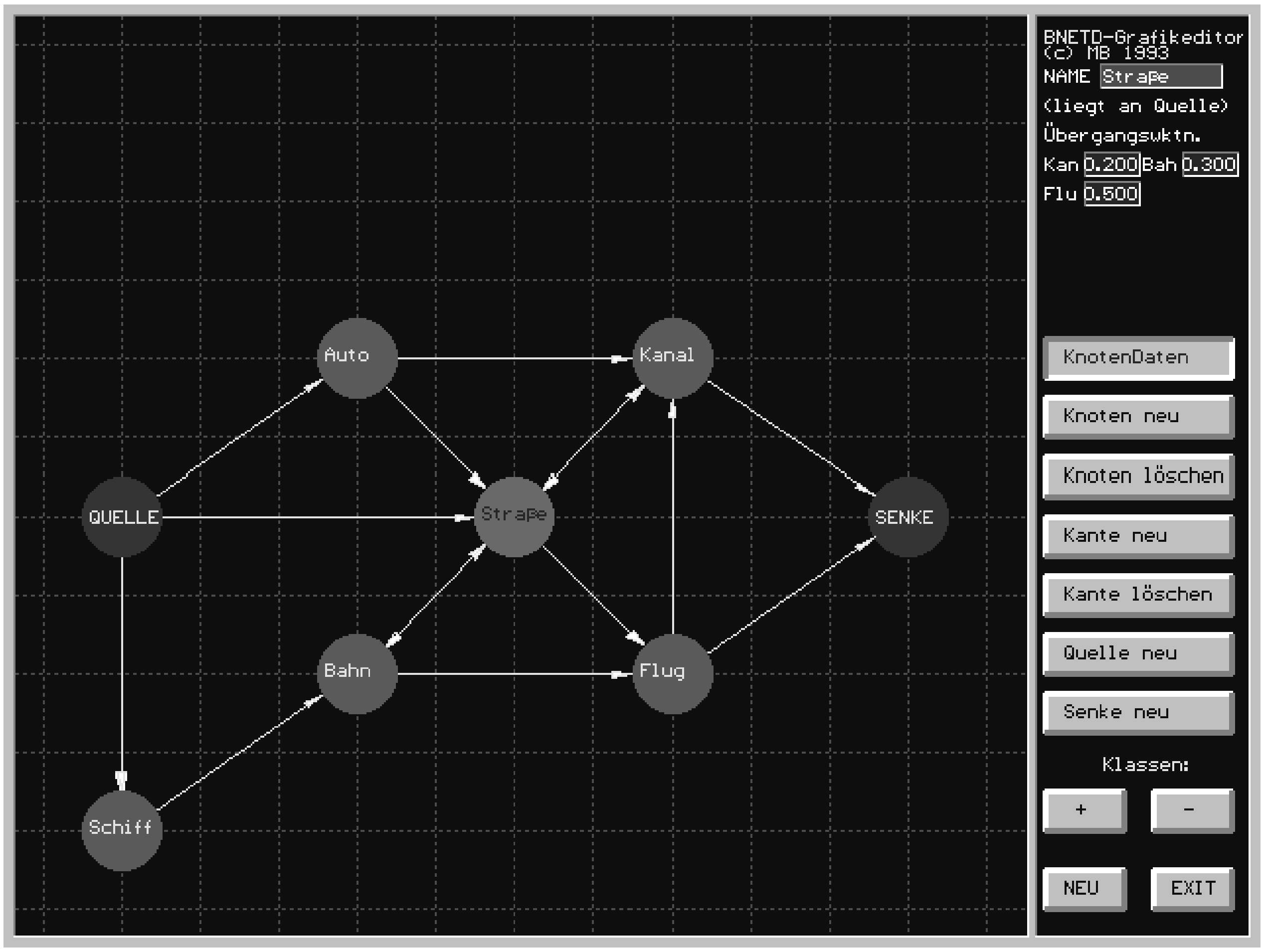

Abbildung 2: Grafischer Editor

ses Fensters sind gleichfalls Detailinformationen zu einzelnen Modellelementen verfügbar.

– *Graphische Modellrepräsentation*

  In BNETD ist es möglich, das aktuelle Modell graphisch darstellen zu lassen. Um mit einer einzigen Modellrepräsentation auszukommen, muß das System die Fähigkeit besitzen, aus einer formalen Modellbeschreibung (Anzahl Knoten und Klassen, Definition der Übergänge) eine graphische Darstellung zu erzeugen. Dazu wurde ein heuristischer Algorithmus entwickelt, der die Anordnung der Knoten hinsichtlich Kantenlänge und Übersichtlichkeit verbessert [4].

- **Kontextsensitive Hilfefunktion**

  Die Oberfläche wurde mit dem Ziel entwickelt, eine schriftliche Dokumentation überflüssig zu machen. Alle Eigenschaften des Systems, insbesondere solche, die nicht selbsterklärend sind, werden 'on-line' erläutert. Die Hilfefunktion beinhaltet auch die Erklärung der Grundlagen der Warteschlangentheorie.

- **Ergebnisrepräsentation**

  Erzielte Ergebnisse (analytisch oder simulativ) werden in einheitlicher Form innerhalb der pseudographischen Oberfläche in Dialogfenstern wiedergegeben. Eine Komponente zur graphischen Darstellung der ermittelten Leistungsgrößen existiert gleichfalls.

- **Verifikation**

  Da die Definition des Modelles nicht programmgesteuert, sondern vollständig anwendergesteuert vor sich geht, erschienen Funktionen zur Verifikation des Modells sinnvoll. So wird überwacht, inwiefern Verteilungen von Ankunfts- und Bedienströmen die notwendigen

Parameter aufweisen. Weiterhin wird das Warteschlangennetz auf „Fallen“ oder „Tote Knoten“ überprüft, welche Strukturmängel darstellen.

Es wurden im Laufe der Entwicklung von BNETD noch kleinere Anpassungen vorgenommen, insbesondere wurde auch eine englischsprachige Version der Oberfläche erstellt [6].

1.4 Analysekomponente

Die Analysekomponente beinhaltet Algorithmen sowohl zur exakten als auch zur approximativen Lösung von Warteschlangenmodellen. Sofern das Netz bestimmmten Bedingungen genügt, kann ein Einsatz der Analysekomponente schnell zu einem genauen Ergebnis führen. Die Implementation der Algorithmen in BNETD ist beschrieben in [7] und [8]. Die aktuelle Version von BNETD beinhaltet die folgenden Algorithmen:

- **Exakte Lösung**

  Gordon/Newell-Theorem, Jackson-Theorem für offene Netze und Faltungsalgorithmus.

- **Approximative Lösung**

  Summationsmethode.

Für einen Überblick über weitere existierende analytische, numerische und approximative Verfahren siehe [9].

1.5 Simulationskomponente

Genügt das Modell nicht den Bedingungen, die eine approximative oder exakte Lösung mittels der Analysekomponente ermöglichen, ist der Einsatz der Simulationskomponente das Mittel der Wahl.

Entwurf und Implementation der Simulationskomponente wird detailliert diskutiert in [10]. Die besonderen Fähigkeiten der Simulationskomponente liegen im Umgang

- mit begrenzten Warteschlangenkapazitäten und Blockierungen,
- mit Prioritäten und
- mit nichtexponentiellen Verteilungen von Ankunfts- und Bedienzeit.

1.5.1 Implementation einer Klassenbibliothek

Die Klassenbibliothek stellt „Bausteine“ für die Entwicklung der BNETD-Simulationskomponente zur Verfügung. Im einzelnen sind dies Warteschlangen, Bedieneinrichtungen, Verzweiger, Quellen und Senken, Listenklassen und Nachrichten. Dadurch kann ausgehend von einer universellen Lösung eine spezielle Komponente gebaut werden.

Die Hierarchie ist einfach erweiterbar (beispielsweise durch Ableitung einer Warteschlange, welche eine andere Entnahmestrategie realisiert). Die Klassen der Bibliothek stellen potentiell eine höhere Funktionalität bereit, als von BNETD benötigt wird. (In BNETD ist jeder Bedieneinrichtung genau eine Warteschlange zugeordnet. Das ist durch die Klassenbibliothek *nicht* forciert.) Dieses wird ermöglicht durch die strikte Trennung der Funktionalität der einzelnen Klassen. Die dadurch erreichte Flexibilität sollte potentielle Performance-Vorteile eines „monolithischen“ Knotenobjektes (Integration von Warteschlangen- und Verzweigerfunktionalität) mehr als aufwiegen.

Insgesamt wird der Ansatz realisiert, der die einzelnen Klassen der Simulation als autonome Objekte auffaßt, die mittels Nachrichten kommunizieren. Die jeweilige Reaktion auf Nachrichten ist in der Ereignisbehandlungs-Methode definiert.

Die Klassen der Simulationsbibliothek führen jeweils Statistiken über ihren Zustand, aus denen schließlich Leistungsgrößen bestimmt werden können.

Die Hierarchie der Zufallszahlengeneratoren baut auf einem multiplikativen linearen Kongruenzgenerator auf:

$$y_n = (ay_{n-1}) \bmod m$$

mit a=1607 und m=2147483647. Andere Verteilungen werden durch entsprechende Transformationen erzeugt.

Die Simulationsobjekte verfügen über die Fähigkeit, auf Blockierungen des nachfolgenden Objektes geeignet zu reagieren. Die einfachste Variante ist, im Fall von blockierten Nachfolgern Forderungen einfach zu verwerfen (*Verlustsysteme*). Abbildung 4 zeigt das Kommunikationsprotokoll Sender - Verteiler - Empfänger im Falle der Blockierung der empfangenden Warteschlange.

Wie zu sehen ist, wird die Forderung, welche logisch noch einen Bedienkanal des Senders belegt, physisch im Verteilerobjekt gepuffert.

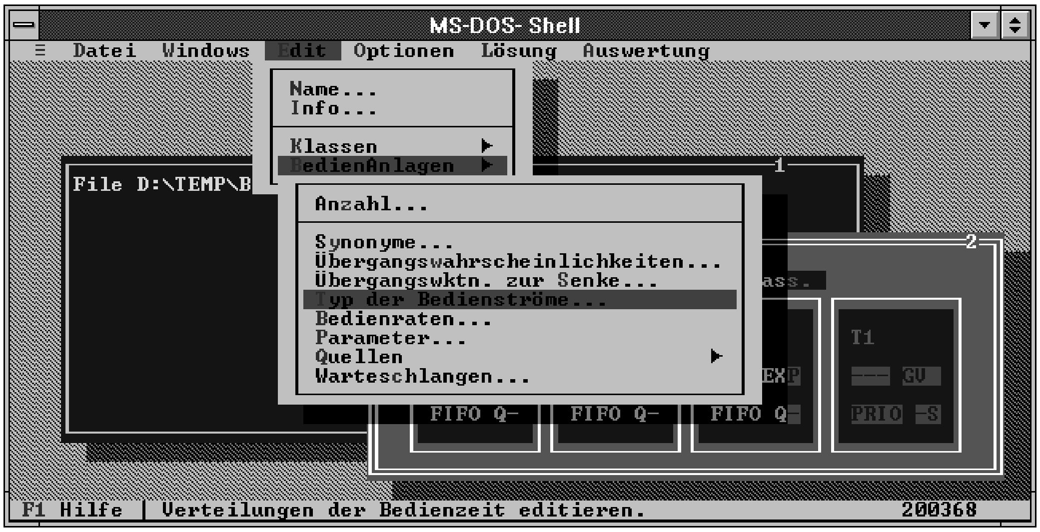

Abbildung 3: Programmoberfläche von BNETD

Ein Problem bei der Realisierung einer solchen ereignisorientierten Simulation ist das mögliche Auftreten von **Deadlocks**. Die Voraussetzung hierfür ist das Vorhandensein von Zyklen im Modell, und daß Kapazitäten der Elemente dieses Zyklus sämtlich begrenzt sind. Deadlocks können durch Strukturänderung vermieden werden (durch Einfügen einer Warteschlange mit unendlicher Kapazität). Ein anderer - in unseren Augen eleganterer - Weg bestand darin, Deadlocks zur Laufzeit zu erkennen. Dieser Algorithmus arbeitet deterministisch und bringt praktisch keine Performanceeinbußen. Deadlocks werden unmittelbar nach ihrer Entstehung erkannt. Wesentlich ist gleichfalls, daß bis zu diesem Zeitpunkt vorliegende Statistiken als Leistungsgrößen genutzt werden können. Die algorithmische Deadlockerkennung berücksichtigt mehrere Modellklassen und ist - gerade für größere Modelle - sinnvoll, da nicht alle Deadlock-Gefährdungen ohne weiteres durch „Hinschauen“ erkannt werden können.

Der Algorithmus gliedert sich wie folgt[10]:

1. **Feststellen potentieller Deadlockgefahr vor Simulationsbeginn**

   Hierbei wird mittels einiger Matrixoperationen über die aus der Übergangsmatrix gewonnne Adjazenzmatrix des Netzgraphen festgestellt, ob sich im Modell Zyklen befinden, die einen Deadlock enthalten können. Sollte keine Deadlockgefährdung vorliegen, sind die weiteren Schritte selbstverständlich gegenstandslos.

2. **Verwaltung von aktuell blockierten Forderungen in einer Matrix**

   Während der Simulation wird eine spezielle *Blockierungsmatrix* geführt, die Informationen über blockierte Forderungen beinhaltet, die Teil eines nach dem im ersten Schritt ermittelten gefährdeten Zyklus sind.

3. **Algorithmische Erkennung von Deadlocks**

   Wird während der Laufzeit festgestellt, daß sämtliche Bedienkanäle einer Bedienanlage durch blockierte Forderungen belegt sind, wird mittels einer speziellen Nachricht an den Simulatorkern ein Test auf einen Deadlock ausgelöst. Der zu diesem Zweck entwickelte rekursive Algorithmen kann anhand der Informationen in der Blockierungsmatrix feststellen, ob es sich um eine temporäre Blockierung oder um einen Deadlock handelt.

1.5.2 Entwicklung der speziellen Simulationskomponente von BNETD

Die Entwicklung einer speziellen Simulationskomponente wird geeignet durch die Entwicklung einer Klasse TSimulator, welche den Simulatorkern darstellt, realisiert. Diese ist für das Ansprechen der Schnittstellen, Initialisierung, Auswertung und Verifizierung verantwortlich.

Im einzelnen sind folgende Schritte notwendig:

1. Bildung eines Modelles mit den durch die Simulationsbibliothek zur Verfügung gestellten

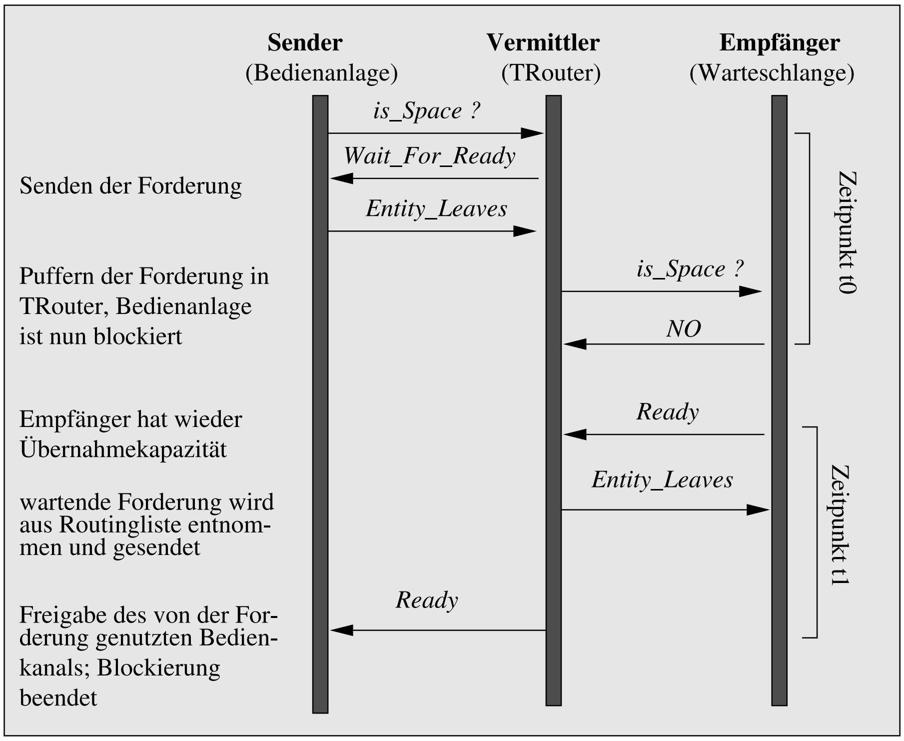

Abbildung 4: Kommunikation der Simulationsobjekte

Klassen aus der formalen Modellbeschreibung. Das ist Aufgabe der Methode TSimulator::Init.

2. „Anstoßen" der Simulation. Die Methode TSimulator::Run initialisiert die Ereignisliste.

3. Ermittlung der Leistungsgrößen aus den in den einzelnen Modellelementen geführten Statistiken. Die Methode TSimulator::Done führt dieses durch und übergibt die Leistungsgrößen an die Ergebnisschnittstelle. Gleichfalls werden aufgetretene Warnungen oder Fehler an die Nutzerschnittstelle propagiert.

4. Je nach angestrebter Verwendung empfiehlt sich eine Verifizierung für das formale Modell.

Die Simulation kann jederzeit gestoppt werden, die bis zu diesem Zeitpunkt ermittelten Werte stehen als Resultat zur Verfügung.

1.6 Einsatzgebiet und Leistungsumfang

BNETD wurde als universelles Werkzeug für die Lösung von Problemen, die sich mittels klassischer Warteschlangentheorie darstellen lassen, konzipiert. Diese grundlegende Funktionalität wurde um Fähigkeiten zur Behandlung von Blockierungen erweitert.

Der Schwerpunkt wurde bewußt auf einfache und intuitive Handhabbarkeit gelegt. Für ein großen Anteil von Problemen, die sich bei der einführenden Beschäftigung mit Warteschlangenmodellen ergeben, ist BNETD ein geeignetes Werkzeug.

1. **Unterstützte Modelle**

   BNETD behandelt einfache Warteschlangensysteme sowie offene und geschlossene Warteschlangennetze.

2. **Modellierung als Netz von Knoten**

   BNETD faßt Warteschlangensysteme als Netz von Knoten auf. Ein derartiger Knoten besteht aus Warteraum und Bedieneinrichtung, die im folgenden beschrieben sind. Übergänge zwischen den Knoten werden durch Übergangswahrscheinlichkeiten beschrieben.

3. **Unterstützung mehrerer Forderungsklassen**

   Bis zu fünf verschiedene Forderungsklassen können definiert sein. Diese können sich beispielsweise durch Übergänge, Prioritäten, Verteilungen und zugehörige Parameter unterscheiden. Klassenübergänge werden nicht unterstützt.

4. **Bedieneinrichtungen**

   Bedieneinrichtungen können mehrkanalig sein. Als Verteilungen der Bedienzeit werden Exponentialverteilung, Erlangverteilung, Normalverteilung, Gleichverteilung und deterministische Verteilung angeboten.

5. **Verteilung des Ankunftsstromes**

   Für die Modellierung offener Systeme können Ankunftsströme nachgebildet werden, die poissonsch, normalverteilt, erlangverteilt, gleichverteilt oder deterministisch sind.

6. **Warteschlangen**

   Warteräume können sowohl unbegrenzt als auch begrenzt sein. Als Entnahmestrategien sind FCFS, LCFS, Priority und Random implementiert.

Erweiterungen, die sich in BNETD als nützlich erweisen könnten, sind: Klassenübergänge, Vereinigen oder Teilen von Forderungen, Darstellung von Abhängigkeiten (Zuliefervorgänge), Lastabhängigkeiten, preemptive Strategien.

Obwohl das aktuelle BNETD nicht hinsichtlich einer speziellen Anwendung entwickelt wurde, war am ehesten mit Anwendungen im Bereich Leistungsbewertung von Rechnersystemen zu rechnen. Wie der zweite Teil des Beitrages zeigt, sind die Möglichkeiten von BNETD nicht darauf beschränkt.

1.7 Einordnung des Systems

BNETD sieht sich als universelles, bedienerfreundliches integriertes Werkzeug vorrangig zur Behandlung von Problemen der klassischen Warteschlangentheorie. Inspiriert durch eine frühere Version von PEPSY [11] und durch das an der TU Dresden entwickelte Simulationssystem TOMAS [12], will BNETD vorrangig bezüglich Bedienbarkeit und Kompaktheit neue Möglichkeiten aufzeigen. Mit der Implementation auf dem PC stößt BNETD in eine Nische, da der Großteil der bekannten vorhandenen Tools UNIX-basiert ist. BNETD verfügt über Algorithmen zur Prüfung der Modellstruktur, verfolgt einen eigenen Ansatz bezüglich Blockierungen (Erkennung von Deadlocks zur Laufzeit) und nutzt Heuristiken zur graphischen Darstellung einer Modellbeschreibung.

Die Simulation wurde - wegen des breiten Einsatzspektrums und der Vielzahl der ermittelbaren Leistungsgrößen - im bisherigen Einsatz bevorzugt. Das oft gebrauchte Argument, demzufolge für eine hohe Genauigkeit eine zeitlich entsprechend umfangreiche Simulation vonnöten ist, wird durch schnellere Hardware zunehmend relativiert.

Abschließend sei auf Arbeiten an der Universität Dortmund (HIT,MACOM,QPN) [13], auf das schon erwähnte PEPSY (Universität Erlangen) [11] - welches sich u.a. durch eine Vielzahl implementierter analytischer Verfahren auszeichnet - , auf das kommerzielle Tool QNAP II [14] und auf [19] verwiesen.

2 BNETD im Einsatz

Im zweiten Teil des Beitrages soll die Anwendbarkeit von BNETD auf Problematiken betreffend ATM Netzwerken gezeigt werden. Die erstmalige Modellierung von ATM-spezifischen Problemen mit BNETD erfolgte in [6].
Die erste Anwendung betrifft ein Problem des Designs von ATM Switches, die zweite Anwendung modelliert den Transport verschiedenartiger Datenströme mit unterschiedlichen Quality of Service Parametern über ATM. Einführungen in diese Problematiken sind in [15] und [16] enthalten.

2.1 Input Queues versus Output Queues

2.1.1 Problemstellung

In ATM erfolgt das Routing von Paketen (ATM-Zellen) durch *Switches*. Entsprechend von im Header der Zelle enthaltenen Informationen werden Zellen an den vorgesehenen Ausgabeport geleitet. Die Route der Zelle wurde beim Verbindungsaufbau festgelegt und wird in den Switches verwaltet. Sofern die Pfade gleichzeitig den Switch betretender Zellen unabhängig sind, können Zellen parallel weitergeschaltet werden (*Space Division Switching* , realisiert durch *Crossbar*-Architekturen).

Für den Fall, daß zwei Zellen einen Switch gleichzeitig betreten und zum gleichen Output-Port geleitet werden sollen, muß eine Behandlungsstrategie bestimmt werden. Zwei prinzipielle Möglichkeiten, dies zu realisieren sind die Pufferung von Zellen

- an den Eingangsports des Switches (*Input Queueing*) oder
- an den Ausgangsports des Switches (*Output Queueing*).

Intuitiv ist verständlich, daß im Falle von Input Queues der Fall auftreten kann, daß eine (aus obigem Grund) an einem Eingangsport wartende Zelle

eine hinter ihr wartende Zelle blockiert, welche eigentlich „freie Bahn“ hätte. Dieser Effekt wird als *Head Of Line (HOL) Blocking* bezeichnet.
Eine formale Analyse (siehe [17]) bestätigt die höhere Leistungsfähigkeit von Output-Puffern in Verbindung mit Crossbar-Architekturen. Der HOL-Effekt soll im ersten Anwendungsbeispiel simulativ nachvollzogen werden.

2.1.2 Modellierung

Abbildung 5 zeigt die verwendete Abstraktion eines ATM-Switches. Es handelt sich um einen Crossbar-Switch, dessen aggregierte Bandbreite 2.56 Gbit/s [3] beträgt und über drei Eingangsports und vier Ausgangsports verfügt. Diese werden jeweils mit 622 MBit/s[4] physischen Leitungen verbunden.

Auf jeden Input-Port werden vier Datenströme mit exponentiell verteilter Zwischenankunftszeit gesendet (λ=0.295 Zellen/μs pro Forderungsklasse - dies entspricht einer durchschnittlich genutzten Bandbreite von 500 MBit/s). Die Modellierung des Eingangsstromes als poissonsch stellt eine Abstraktion dar. Die Annahme, daß Zellankünfte unabhängig sind, ist optimistisch [16], soll aber dennoch für das vorgestellte Modell genügen.

Die Warteschlangen des Input-Ports stellen Input-Queues dar, weiterhin wird eine Glättung des Eingangsstromes erreicht (traffic smoothing).

Die Bedienung in den Input-Ports reflektiert die Zeit, die zum Weiterleiten an die Output-Ports benötigt wird. In diesem Beispiel soll die aggregierte Bandbreite von 2.56 Gbit/s durch eine Bedienrate von 2.01 Zellen/μs für jeden Input-Port nachgebildet werden. Die Verzweigung der Zellen ist fest durch die Zugehörigkeit zu einem Datenstrom (Forderungsklasse 1-4) gegeben. Zellen des Stromes Eins werden an Output-Port Eins geleitet usw.

Output-Ports verfügen gleichfalls über je eine Warteschlange (Output-Queues). Ist die Kapazität dieser Warteschlangen unbegrenzt, fungieren die Input-Queues nur als Einrichtung zur Glättung des Datenstromes. Dagegen wurde eine auf Input-Queues basierende Switch-Architektur durch eine Begrenzung dieser Kapazität auf eine Zelle modelliert.

Die Bedieneinrichtungen der Output-Queues bedienen Forderungen mit einer Rate von 1.47 Zellen/μs (entsprechend der Bandbreite eines 622 MBit-Kanals).

[3] z.B. DEC GigaSwitch; aktuelle Switches liegen in diesem Bereich
[4] nach SONET OC-12

2.1.3 Resultate

Folgende Resultate (Zeiten in μs) wurden - bei einer Simulationsdauer von 10 min auf einem Pentium 90 - ermittelt:

	Input	**Output**
$\bar{T}_v$	2.56	2.00
$\mathcal{D}$	3.54	3.57
ρ	0.646	0.600
Inputport 1		
ρ	0.704	0.593
$\bar{L}_w$	1.16	0.434
$L_{w_{max}}$	25	12
$\bar{T}_{block}$	0.102	0
Outputport 1		
ρ	0.602	0.608
$\mathcal{D}$	0.887	1.192
$\bar{L}_w$	0.249	0.413
$L_{w_{max}}$	1	10

($\bar{T}_v$ - Verweilzeit, $\mathcal{D}$ Durchsatz, ρ Auslastung, $\bar{L}_w$ mittlere Warteschlangenlnge und $\bar{T}_{block}$ mittlere Blockierungsdauer)

Neben globalen Ergebnisgrößen sind jeweils ausgewählte Leistungsgrößen eines Eingangs- und Ausgangsports wiedergegeben.

2.2 Multimediaströme über ATM

2.2.1 Problemstellung und Modellierung

Die Anforderungen an Netzwerke haben sich gewandelt. Stand zur Entwicklungszeit des bekannten TCP/IP-Protokollstacks eine zuverlässige Datenübertragung im Mittelpunkt, sind es heute größere Anforderungen bezüglich Bandbreite und Verzögerungszeit. Kontinuierliche Medien wie Audio und Video verlangen zusätzlich Performance-Garantien. In der vorgestellten Anwendung wird die Übertragung von drei verschiedenartigen Datenströmen über eine physikalische Verbindung mit 1.5 Mbit/s Bandbreite (eine T1- Telefonleitung) unterhalb von ATM simuliert. Es soll untersucht werden, inwiefern sich durch ein statisches Prioritätenschema Zusagen bezüglich der Dienstgüte machen lassen können.

Der Sachverhalt wurde wie folgt abstrahiert und modelliert: Drei Rechner senden Datenströme mit verschiedenen Charakteristika über eine physikalische Leitung (statistisches Multiplexing der Zellen). In den Hosts tritt eine konstante Paketierverzögerung auf (durch das Zerlegen von zum Beispiel Videoframes in Zellen). Die Warteschlange

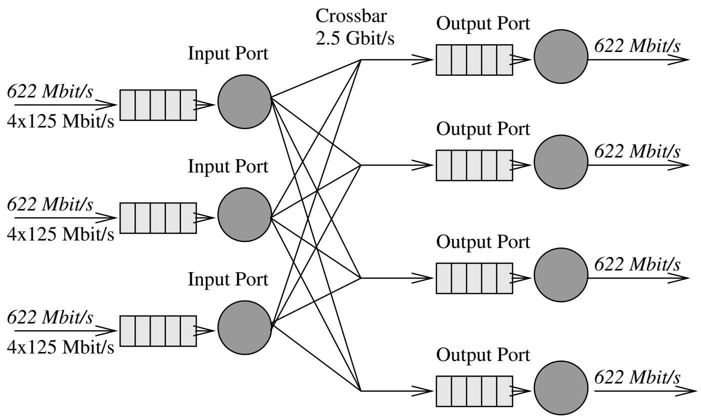

Abbildung 5: Modellierung der Switch-Struktur

vor der physikalischen Leitung realisiert ein 'traffic shaping'[5]. Eine Warteschlange, die nach FCFS arbeitet, realisiert bereits das 'Leaky Bucket' Konzept [16]. Durch die Vergabe von Prioritäten an die einzelnen Forderungsklassen entspricht die Funktion einem 'Multiple Leaky Bucket'.

Folgende Datenströme wurde modelliert:

1. *Audiodaten in CD-Qualität mit konstanter Bitrate von 128 Kb/s.*

   Die Ankunft des Audiostromes sei deterministisch mit einer Ankunftsrate λ=0.302 ms. Audiodaten haben die höchste Priorität.

2. *Videodaten (nach H.261 kodiert) mit variabler Bitrate, im Mittel 424 Kb/s.*

   Die Ankunft des Videostromes seien normalverteilt mit (μ=1,σ=0.2). Der Videostrom besitzt normale Priorität. [18] enthält eine Untersuchung zur statistischen Verteilung des Datenumfangs komprimierter Videoströme.

3. *Interaktive Daten mit einer mittleren Datenrate von 424 Kb/s.*

   Die Ankunft des Audiostromes ist poissonsch mit λ=1. Diese Daten haben die niedrigste Priorität.

Die als Übertragungsmedium genutzte 1.5Mbit/s-Verbindung wird durch eine Bedieneinrichtung mit μ=3.5 Zellen/ms realisiert. Die Entnahme aus der Warteschlange erfolgt nach statischen Prioritäten.

2.2.2 Resultate

Erwartungsgemäß ergab sich bezüglich Verweil- und Wartezeiten eine deutliche Differenzierung der Ströme nach ihrer Priorität. Erwähnenswert sind die relativ geringen Unterschiede in den Leistungsgrößen von Audio- und Videostrom gegenüber dem interaktiven Datenstrom. Im folgenden seien einige relevante ermittelte Leistungsgrößen dargestellt (Pentium-90, Simulationsdauer 30 min, Basiszeiteinheit Millisekunde):

Leistungsgrößen T1-Kanal	
Gesamt	
ρ	0.6595
$\mathcal{D}$	2.3023
$\bar{T}_w$	0.2001
$\bar{p}_w$	0.6116
$\bar{L}_w$	0.4606
Audiostrom	
ρ	0.0865
$\mathcal{D}$	0.3021
$\bar{T}_w$	0.0856
$\bar{p}_w$	0.5878
$\bar{L}_w$	0.0259
Videostrom	
ρ	0.2854
$\mathcal{D}$	0.9965
$\bar{T}_w$	0.1113
$\bar{p}_w$	0.5708
$\bar{L}_w$	0.1109
Datenstrom	
ρ	0.2875
$\mathcal{D}$	1.0037
$\bar{T}_w$	0.3226
$\bar{p}_w$	0.6593
$\bar{L}_w$	0.3238

($\bar{T}_w$ - mittl. Wartezeit, $\mathcal{D}$ Durchsatz, ρ Auslastung, $\bar{L}_w$ mittlere Warteschlangenlnge und $\bar{p}_w$ Wartewahrscheinlichkeit)

[5] Anpaßung von Datenströmen an die Kapazität des Netzwerks

3 Ausblick

Mit BNETD wurde ein universelles, integriertes Werkzeug zur Lösung von Problemen vorrangig der klassischen Warteschlangentheorie vorgestellt. BNETD vereint graphische Modellerstellung, Oberfläche und Ergebnisauswertung mit einer Analysefunktion und einer leistungsfähigen Simulation. Ein wesentlicher Gesichtspunkt des Entwurfs war eine einfache und intuitive Nutzbarkeit des Systems.

Das Design von BNETD, insbesondere der Simulation, wurde näher vorgestellt. Ein Algorithmus zur Erkennung von Deadlocks in der Simulation wurde eingeführt.

Die breite Anwendbarkeit von BNETD wurde mit der Erläuterung der Modellierung zweier aktueller Probleme der Computerkommunikation dokumentiert.

Literatur

[1] *Pooley,R.*: Performance Analysis Tools in Europe; 1995 *it+ti - Informationstechnik und Technische Informatik 3/95*

[2] *Smith,C.*: The Evolution of Performance Analysis Tools; 1995 *it+ti Informationstechnik und Technische Informatik 3/95*

[3] *Irmscher,K.*: Analyseverfahren geschlossener Bedienungsnetze (Theoretische Grundlagen des Programmsystems BNETD); 1984 *Schriftenreihe „Informationsverarbeitung in Hoch- und Fachschulwesen“, Ministerium für Hoch- und Fachschulwesen, Berlin*

[4] *Borriss,M.*: Window-Oberfläche für BNETD; 1993 *Grosser Beleg, TU Dresden*

[5] *Borland GmbH*: Turbo Vision C++; 1991 *Starnberg*

[6] *Borriss,M.*: Simulating ATM Networks; 1995 *Student report, Rensselaer Polytechnic Institute, Troy,NY,USA*

[7] *Pahlisch,S.*: Exakte Analyse von Produktformnetzen; 1994 *Großer Beleg, TU Dresden*

[8] *Rading,A.*: Analysealgorithmen für Bediensysteme und -netze; 1993 *Großer Beleg, TU Dresden*

[9] *Bolch,G.*: Analyse von Rechensystemen; 1982 *Teubner Stuttgart*

[10] *Borriss,M.*: Simulation von Bedienungsnetzen und -systemen in BNETD; 1994 *Diplomarbeit,TU Dresden*

[11] *Bolch,G; Roessler,M; Zimmer,R; Jung,H; Greiner,S.*: Leistungsbewertung mit PEPSY-QNS und MOSES; 1995 *it+ti Informationstechnik und Technische Informatik*

[12] *Frank,M; Neuthe,St.*: Fertigungssysteme modellieren und simulieren; 1990 *Zeitschrift für wirtschaftliche Fertigung und Automatisierung, Jahrgang 85/6, Seiten 326-333*

[13] *Bause,F.,Sczittnick,M.*: Design von Modellierungstools zur Leistungsbewertung; 1995 *it+ti Informationstechnik und Technische Informatik*

[14] *Veran,M.,Potier,D*: QNAP 2: A Portable Environment for Queueing System Modelling; 1985 *In: Potier,D. (ed.):Proc. of Modelling Techniques and Tools for Computer Performance Evaluation*

[15] *De Prycker,M*: Asynchronous Transfer Mode 2nd. ed.; 1994 *Ellis Horwood*

[16] *Partridge,C*: Gigabit Networking; 1994 *Addison Wesley*

[17] *Karol,M.J.,Hluchy,M.G.,Morgan,S.P.*: Input versus Output Queueing on a Space-Division Packet Switch; 1987 *IEEE Transactions on Communications Vol.35 No.12*

[18] *Enssle,J*: Modelling And Statistcal Multiplexing Of VBR MPEG Compressed Video In ATM Networks; 1994 *Proc. of 4th Open Workshop on High Speed Networks, Brest*

[19] *Beilner,H.,Bause,F.(Ed.)*: Quantitative Evaluation of Computing and Communication Systems (Lecture Notes in Computer Science 977); 1995 *Springer Heidelberg*